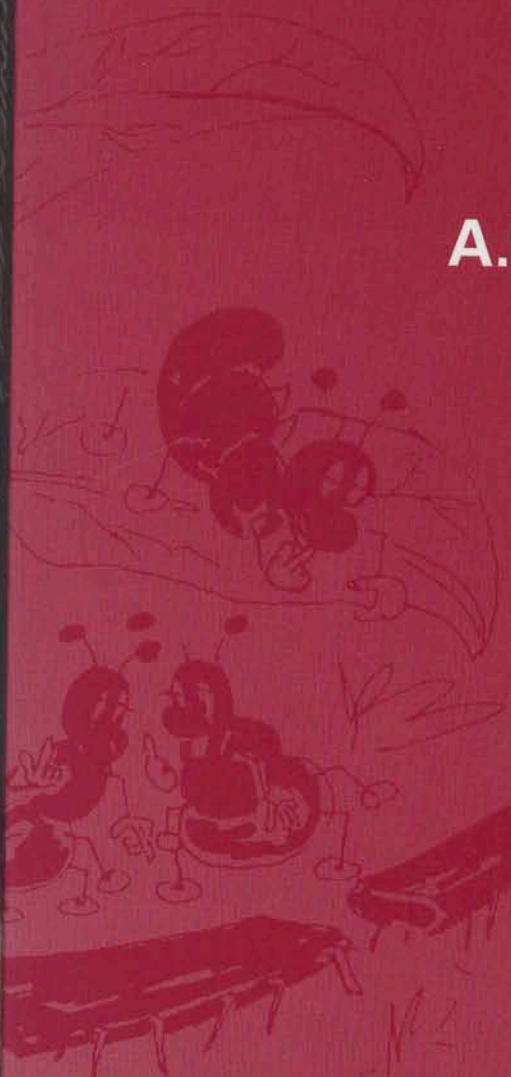


wordt niet uitgeleend

DIGITALE TECHNIEK

van probleemstelling
tot realisatie
deel 1

A.P. Thijssen, H.A. Vink



S
THIJSSSEN

5e druk

S

Thyssen

7

710836

3.6 8463

DR 2992464

Studieboeken

Digitale Techniek

van probleemstelling tot realisatie

deel 1



Digitale Techniek

van probleemstelling tot realisatie

deel 1



Digitale Techniek

van probleemstelling tot realisatie

deel 1



ir. A.P. Thijssen, ir. H.A. Vink

5e geheel herziene druk

Delft University Press

CIP-gegevens Koninklijke Bibliotheek Den Haag

Thijssen, A.P.

Digitale Techniek : van probleemspecificatie tot realisatie / door A.P. Thijssen en H.A. Vink

Delft : Delft University Press

Met lit. opg., reg.

ISBN 90-407-1793-1

NUGI 832

Trefw.: digitale techniek

© VSSD

Eerste druk 1982

Vijfde druk 1999

Uitgegeven door:

Delft University Press

Mekelweg 4, 2628 CD Delft

tel. 015 278 3254, telefax 015 278 1661, e-mail dup@dup.tudelft.nl

In opdracht van:

Vereniging voor Studie- en Studentenbelangen te Delft

Poortlandplein 6, 2628 BM Delft

tel. 015 - 2782124, telefax 015 - 2787585, e-mail: vssd@tudelft.nl

internet: www.oli.tudelft.nl/vssd/hlf/handleidingen/html

Alle rechten voorbehouden. Niets uit deze uitgave mag worden verveelvoudigd, opgeslagen in een geautomatiseerd gegevensbestand, of openbaar gemaakt, in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door fotokopieën, opnamen, of op enige andere manier, zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photo-copying, recording, or otherwise, without the prior written permission of the publisher.

ISBN 90-407-1793-1

Voorwoord

Na vele jaren trouwe dienst van de vorige druk zijn we toe aan een nieuwe druk. Hoewel de schakelalgebra hetzelfde is gebleven, er zijn geen regels bijgekomen, en aan het vak nog steeds de statische discipline ten grondslag ligt is het onderwijs in de Digitale Techniek allesbehalve statisch gebleken. Er zijn diverse redenen op te noemen voor een grondige herziening van de tekst. Allereerst is er de technologische ontwikkeling geweest. De CMOS technologie is de grote winnaar gebleken. Daarnaast zijn de prestaties van de logische schakelingen drastisch omhoog gegaan, zowel voor wat betreft de schakelsnelheid van de componenten als voor het aantal componenten op een chip. Dit betekent dat het onderwijs in de Digitale Techniek niet meer beperkt kan worden tot de strikt logische benadering van het digitale ontwerpen.

Steeds meer moet er op het logisch ontwerpniveau rekening gehouden worden met de onderliggende, meer fysisch georiënteerde, ontwerp niveaus. Een onderwerp als timing moet extra aandacht krijgen om de gewenste betrouwbaarheid en reproduceerbaarheid van snelle CMOS schakelingen te kunnen garanderen. Het verschijnsel clock skew en het creëren van marges hiervoor staan recent zeer in de belangstelling. Ook komt de ontwerper door de technologische ontwikkelingen meer in aanraking met de grenzen van wat er technisch nog mogelijk is. Naarmate er meer componenten op een chip geplaatst kunnen worden neemt bijvoorbeeld de dissipatie van de werkende chip toe. Gedeeltelijk kan dit probleem opgelost worden door over te gaan op lagere voedingsspanningen. Dit gaat echter echter onder meer ten koste van de ruismarges, die rond de logische signaalniveaus gedefinieerd moeten zijn. En kleinere ruismarges kunnen weer negatieve gevolgen hebben voor de betrouwbare werking van een schakeling. Een logisch ontwerper, die deze afhankelijkheden niet onderkent, zal vaker dan in het verleden verkeerde ontwerpbeslissingen nemen.

Testbaarheid van schakelingen is een ander voorbeeld van een vakgebied dat recent tot ontwikkeling is gekomen. De voornaamste reden hiervoor is de toegenomen complexiteit van de schakelingen. Zonder ingebouwde voorzieningen voor testbaarheid kan een geproduceerde schakeling niet of niet economisch op zijn juiste werking geverifieerd worden.

Bij het samenstellen van deze druk hebben we ervoor gekozen de grensvlakken tussen het logisch ontwerp en het meer fysisch georiënteerde deel van het ontwerp in de tekst op te nemen. Dit betekent een toename van de omvang. Deel 1 is gewijd aan de combinatoriek. Deel 2 behandelt het ontwerpen van sequentiële schakelingen. Ten behoeve van een korte introductie cursus in de Digitale Techniek is achter in beide delen een lijst opgenomen van paragrafen, welke hiervoor kunnen dienen. Dit maakt het boek ook geschikt voor informatica-type opleidingen. Ten behoeve van het onderwijs is bij de auteurs tevens een serie overheads te verkrijgen.

In deze druk, die in eerste instantie verschijnt ten behoeve van het onderwijs aan de Technische Universiteit te Delft, voor de colleges Digitale Techniek, is veel ervaring verwerkt die is opgedaan tijdens externe cursussen in het Nederlandse bedrijfsleven. De vaak zeer pittige discussies met de deelnemers aan deze cursussen bij onder meer de Stichting Post-HTO van de Haagse Hogeschool, Lucent Technologies Nederland b.v., het Philips Natuurkundig Laboratorium - CTT te Eindhoven en Philips Semiconductors te Nijmegen en met deelnemers afkomstig uit veel andere bedrijven, hebben in belangrijke mate bijgedragen tot de inhoud. Zoals vaak geldt ook hier dat de praktijk de beste

leerschool is. Veel problemen, die tijdens de cursussen kwamen boven drijven, hebben geresulteerd in meer inzicht in wat voor de praktijk belangrijke onderwerpen zijn. Samen met een goede cursus VHDL, een veel gebruikte beschrijvingstaal voor het ontwerpen van digitale hardware, heeft de lezer een voldoende basis voor het ontwerpen van digitaal werkende schakelingen.

Natuurlijk bevat dit boek fouten. Wij vragen de gebruiker daarom dringend eventuele opmerkingen en suggesties voor verbeteringen aan ons door te geven. Elke eerste aanmelding van een fout, hoe klein ook, wordt beloond met een bon. Vijf bonnen geven recht op een goede fles wijn, of op een sapjespakket. Deze regeling heeft bij vorige drukken uitstekend gewerkt. Zoals u heeft kunnen constateren zijn de wijnprijzen recent gestegen, vanwege de grotere vraag. Het zal ons een genoegen zijn de nodige flessen uit te reiken. Stuur uw commentaar toe aan A.P. Thijssen, werkeenheid CARDIT, Faculteit ITS, TU Delft, Mekelweg 4, 26 28 CD Delft of stuur een mailtje aan A.P.Thijssen@its.tudelft.nl.

Wij wensen u succes met uw studie in dit boeiende vakgebied.

Delft/Eindhoven, november 1998

Loek Thijssen

Hans Vink

Inhoud

0 Inleiding	13
<i>Tijdcontinue/intensiteitcontinue signaalbewerking</i>	14
<i>Tijddiscrete/intensiteitcontinue signaalbewerking</i>	15
<i>Tijddiscrete/intensiteitdiscrete signaalbewerking</i>	16
<i>Binaire signaalbewerking</i>	17
<i>Het ontwerptraject</i>	21
Literatuur	22
Opgaven hoofdstuk 0	23
1 Getalrepresentaties en codes	
1.1 Het decimale en het binaire talstelsel	25
<i>Talstelsels</i>	26
<i>Fracties</i>	27
<i>Aantal cijfers</i>	27
<i>De BCD representatie</i>	28
1.2 Andere getalrepresentaties	29
<i>De hexadecimale representatie</i>	29
<i>De octale representatie</i>	29
<i>De excess-3 representatie</i>	30
<i>m-uit-n codes</i>	31
1.3 Grondtal conversie	31
1.4 De conversie BIN-OCT-HEX	33
1.5 Codes voor alfanumerieke karakterverzamelingen	35
<i>De ASCII code</i>	35
<i>Parity</i>	36
<i>De EBCDIC code</i>	36
1.6 Equidistante codes	37
1.7 Binair rekenen	40
1.8 Getalrepresentaties in Z	43
<i>Rekenen in teken-en-modulus</i>	44
1.9 De two's complement representatie	45
<i>De representatie</i>	46
<i>Het tekenbit</i>	48
<i>Alternatieve introductie two's complement</i>	49
1.10 De optelling in het two's complement	49
1.11 Optellen en aftrekken in het two's complement	51
<i>De representant van het tegengestelde getal</i>	52
1.12 Capaciteitsoverschrijding in het two's complement	53
<i>Samenvatting van het two's complement</i>	55
<i>De two's complement cirkel</i>	55
1.13 De one's complement representatie	55
1.14 De floating point representatie	57
<i>ANSI/IEEE 32-bit single format</i>	58
Literatuur	59
Opgaven hoofdstuk 1	60

2 De schakelalgebra

2.1	De logische werking van een component	67
	<i>Keuze van de proposities</i>	69
	<i>Logische variabelen en Booleaanse variabelen</i>	70
	<i>Logische nullen en enen</i>	70
2.2	Bewerkingen in de symbolische logica	71
	<i>De logische operator EN</i>	71
	<i>De logische operator OF</i>	73
	<i>De logische operator NIET</i>	73
	<i>De inclusieve OF en de exclusieve OF</i>	74
	<i>Volgorde van bewerkingen</i>	75
2.3	De schakelalgebra	75
	<i>Volgorde van bewerkingen</i>	76
	<i>Rekenregels voor constanten</i>	76
	<i>Rekenregels voor logische variabelen</i>	77
	<i>Het bewijzen van schakelwetten</i>	81
2.4	Uitbreidbaarheid van schakelwetten	81
2.5	De mintermvorm van logische functies	83
	<i>De mintermvorm</i>	84
	<i>De som-van-produktenvorm</i>	85
	<i>Venn diagrammen</i>	86
2.6	De maxtermvorm van logische functies	87
	<i>Verband tussen de mintermvorm en de maxtermvorm</i>	87
2.7	Logische bewerkingen op functies	90
2.8	Decompositie van functies	92
	<i>Herhaalde decompositie</i>	94
	<i>Volgorde bij decompositie</i>	96
2.9	Toepassing van decompositie op kontaktschakelingen	97
2.10	De ON set, OFF set en DC set van logische functies	99
	<i>Afhankelijkheid van logische variabelen</i>	101
2.11	De keuze van proposities en logische variabelen	102
	<i>Ondubbelzinnigheid van proposities</i>	103
2.12	Voorbeeld opstellen van formules	104
	Literatuur	107
	Opgaven hoofdstuk 2	107

3 Bewerkingen op logische functies

3.1	Karnaughdiagrammen	115
	<i>Waarheidstabellen en Karnaughdiagrammen</i>	115
	<i>De constructie van Karnaughdiagrammen</i>	117
3.2	Reductie van logische functies	120
	<i>Don't cares</i>	122
3.3	Het systematisch uitlezen van Karnaughdiagrammen	123
3.4	De somvorm en de produktvorm van logische functies	127
	<i>De produkt-van-sommenvorm</i>	128
	<i>Recept conversie somvorm $\leftrightarrow$ produktvorm</i>	129
3.5	Toepassingen van Karnaughdiagrammen 1	130

	<i>Logische bewerkingen op functies</i>	130
	<i>Onderzoek van specificaties</i>	131
	<i>Logische schakelingen met meer dan één uitgang</i>	132
	<i>Volgorde van de randschriften</i>	133
3.6	Toepassingen van Karnaughdiagrammen 2	134
3.7	Cube representatie van logische functies	136
3.8	De eenvoudigste vorm van een formule	138
3.9	Het genereren van priemimplicanten	139
3.10	Het bepalen van een dekking	142
3.11	Uitbreidingen op het Quine-McCluskey algoritme	146
	<i>Iteratieve consensus</i>	146
	<i>Multiple-output networks</i>	149
3.12	Binary Decision Diagrams	151
3.13	Het opstellen van BDD's	154
	<i>Vereenvoudigingsregels</i>	155
	<i>Volgorde van de variabelen</i>	156
	<i>Don't cares in BDD's</i>	157
3.14	Van BDD naar som-van-produktenvorm	158
	Literatuur	161
	Opgaven hoofdstuk 3	162

4 Inleiding tot het ontwerpen van combinatorische schakelingen

4.1	Het ontwerpproces	169
	<i>Indeling ontwerpproces</i>	169
	<i>Het ontwerptraject van een opteller</i>	170
	<i>Scope van deze tekst</i>	172
4.2	Het logisch ontwerp van een n-bit full adder	173
	<i>Van numerieke naar logische specificatie</i>	174
	<i>Half adder en full adder</i>	176
	<i>Capaciteitsoverschrijding</i>	176
4.3	Ontwerpen met de operatoren {AND, OR, NOT}	177
	<i>Twolthree-level NOT-AND-OR schakelingen</i>	177
	<i>Twolthree-level NOT-OR-AND schakelingen</i>	178
	<i>Passingsproblemen</i>	179
	<i>Open ingangen</i>	181
	<i>Don't care signaalwaarden</i>	182
4.4	Ontwerpen met de operatoren {NAND, NOR}	182
	<i>NAND en NOR, universeel toepasbare componenten</i>	182
	<i>Twolthree-level NAND-NAND schakelingen</i>	183
	<i>Twolthree-level NOR-NOR schakelingen</i>	184
4.5	Ontwerpen met fanin beperkingen	186
	<i>Opbouw van n-input AND en OR poorten met NAND en NOR poorten</i>	187
	<i>Multilevel decompositie met NAND en NOR poorten</i>	188
	<i>Het decompositieprobleem</i>	191
4.6	Ontwerpvrijheid	192
4.7	Permissible functions	194
	<i>Het optimaliseren van combinatorische schakelingen</i>	198

	<i>Procedure voor het bepalen van permissible functions</i>	200
4.8	De afbeelding op hardware	201
	<i>Het signaalmodel: scheiding van logica en tijd</i>	201
	<i>Positieve en negatieve logica</i>	202
	<i>De logische formule bij een gegeven component</i>	203
	<i>Dualiteit</i>	206
	<i>Individuele logica</i>	207
4.9	Symbolen voor poorten	207
	<i>Vorm van de symbolen</i>	208
	<i>Functiesymbolen</i>	209
	<i>Het aanduiden van negaties</i>	209
	<i>Embedded symbols</i>	211
	<i>Signaaloverdracht op fysisch niveau</i>	212
4.10	Ontwerpverificatie en produktverificatie	214
	<i>Ontwerpverificatie</i>	214
	<i>Produktverificatie of testen</i>	216
	<i>Correct-by-design</i>	216
4.11	Het testen van combinatorische schakelingen	216
	<i>Defecten en fouten</i>	217
	<i>Een foutmodel voor combinatorische poortschakelingen</i>	218
4.12	Testen op basis van het stuck-at-0/1 foutmodel	218
	<i>Testen van multilevel combinatoriek</i>	220
	<i>Instelbaarheid en observeerbaarheid</i>	221
4.13	Permissible functions en Stuck-at testbaarheid	222
	<i>Literatuur</i>	225
	<i>Opgaven hoofdstuk 4</i>	226
5	Het fysisch ontwerpniveau	
5.1	Het schakelen van spanningen	237
	<i>Schakelaars</i>	238
5.2	De MOS transistor als schakelaar	241
	<i>De MOS transistor</i>	242
	<i>MOS transistor types</i>	243
	<i>De CMOS invertor</i>	245
5.3	De opbouw van CMOS poorten	246
	<i>Complexe poorten</i>	249
	<i>Extra voorzieningen</i>	250
5.4	De pass Transistor en de transmission gate	251
5.5	Uitgangsschakelingen	253
	<i>De 3-state uitgang</i>	253
	<i>De open-drain (open-collector) uitgang</i>	255
5.6	Overzicht van TTL logische families	256
	<i>De bipolaire transistor</i>	257
	<i>Poorten in de TTL technologie</i>	259
	<i>TTL logische families</i>	261
5.7	Overzicht van CMOS logische families	262
	<i>Lagere spanningen</i>	264

5.8	Logische familie-aangelegenheden	264
	<i>Ruismarges</i>	265
	<i>Voedingsspanningen</i>	267
	<i>Ground bounce</i>	268
	<i>Schmitt-trigger ingangen</i>	269
	<i>Compatibele impedanties</i>	271
	<i>Fanin en fanout</i>	271
	<i>Conclusies ten aanzien van het compatibel zijn</i>	272
5.9	Tijdmodellen	273
	<i>Het tijdmodel voor een logisch signaal</i>	273
	<i>Het tijdmodel voor een verbinding</i>	275
	<i>Het tijdmodel voor een poort</i>	277
5.10	Timing van multilevel combinatoriek	279
	<i>Het modelleren van modulair opgebouwde schakelingen</i>	281
	<i>Multilevel combinatoriek en dissipatie</i>	283
5.11	De toepassing van (tijd)modellen	283
	<i>Met en is weten</i>	284
	<i>Een best case / worst case model of een statistisch model?</i>	285
	<i>Modellering van het dynamisch gedrag van poorten</i>	285
5.12	Defecten en foutmodellering	286
	<i>Defect statistics</i>	290
	<i>Sequentiële fouten</i>	292
	Literatuur	293
	Opgaven hoofdstuk 5	295

6 Het logisch ontwerpniveau

6.1	Standaardcomponenten	307
6.2	Carrypropagatie in optellers	308
	<i>Carry generate en carry propagate hulpfuncties</i>	308
	<i>Optimalisatie van het full adder ontwerp</i>	311
6.3	Carrypropagatie in 4-bit full adders	312
6.4	Arithmetic logic units	314
6.5	Parallelvermenigvuldigers	317
6.6	Modulair opgebouwde vermenigvuldigers	320
6.7	Het dynamisch gedrag van poorten in optellers	322
	<i>Carry-skip adders</i>	322
	<i>End-around carry</i>	324
6.8	Data-overdracht	326
	<i>Selectoren/multiplexers</i>	327
	<i>Decoders/demultiplexers</i>	328
	<i>priority encoders</i>	329
6.9	Decompositie van logische functies met selectoren	330
	<i>Minimale BDD's en logica met selectoren</i>	332
	<i>Logisch functies in andere componenten dan poorten</i>	334
6.10	Foutendetectie en foutencorrectie	335
	<i>Foutendetectie</i>	335
	<i>De parity check</i>	336

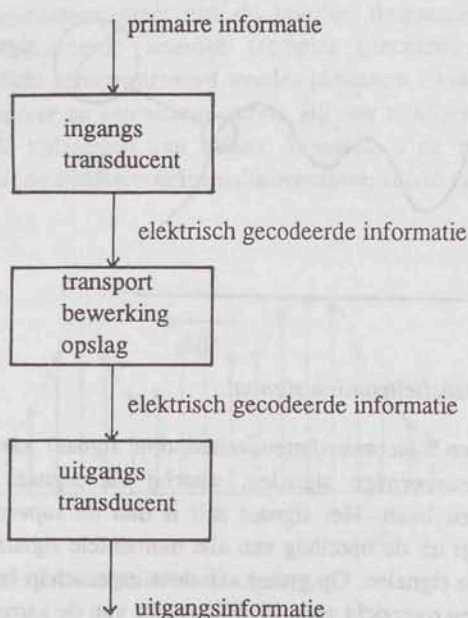
	<i>Parity generators and parity checkers</i>	337
	<i>Hamming afstand en foutendetectie/correctie</i>	340
6.11	Samenvatting	341
	Literatuur	343
	Opgaven hoofdstuk 6	344
7	Het structureel ontwerpniveau	
7.1	Probleemaanpak	355
7.2	Structurele aspecten van de optelling	358
	<i>Kenmerken van de operanden</i>	359
	<i>Worst case of gemiddelde propagatietijd</i>	360
	<i>Carry-completion optellers</i>	362
7.3	Pipeline optellers	364
7.4	De look-ahead carry generator	366
7.5	Overzicht van de optelling	369
	<i>Schema optelling / Domein en representatie van de getallen</i>	370
	<i>Schema optelling / Algoritmek keuze</i>	371
	<i>Schema optelling / Het logisch ontwerp van de schakeling</i>	372
		374
7.6	Ontwerpopdracht	375
	Literatuur	376
	Opgaven hoofdstuk 7	
8	Programmeerbare logica	
8.1.	Read only memories	381
	<i>Read only memories</i>	382
	<i>De interne structuur van een ROM</i>	382
	<i>Enkele details</i>	384
8.2.	Enkele toepassingen van ROM's	386
	<i>BCD-naar-7-segment decoder</i>	386
	<i>Don't cares</i>	388
	<i>Het vervangen van random logica</i>	389
	<i>Het corrigeren van een overdrachtskarakteristiek</i>	390
	<i>Instructiedecodering</i>	391
8.3.	Programmeerbare logische arrays	391
	<i>PAL's</i>	393
8.4.	Optimalisatie in array logica	396
	<i>Bit partitioning</i>	397
	<i>Folding</i>	400
	Literatuur	401
	Opgaven hoofdstuk 8	401
		409
	Appendix symbolen	
	Antwoorden van de opgaven	453
	Studie-advies	455
	Index	457

0

Inleiding

Waarom zoveel aandacht voor digitale/binaire signaalbewerking? Het antwoord op deze vraag kan kort zijn. Kijk om je heen en zie wat er de laatste 10-20 jaar veranderd is op het gebied van (personal) computers, in de telecommunicatie, bij consumentenelektronica en huishoudelijke apparatuur, digitale horloges, chipkaarten, enz. [Bloembergen, 1987; Dinklo, 1994]. Een verdere motivatie om aandacht te besteden aan de keuze voor digitale signaalbewerking lijkt dan ook nauwelijks nodig.

Voor een goed inzicht in het waarom van digitale signaalbewerking, vooral ook in de grenzen van de toepassingsmogelijkheden, is het noodzakelijk wat langer stil te staan bij de gestelde vraag. We doen dit aan de hand van een overzicht van enkele veel gebruikte vormen van informatiedragende signalen in de elektrotechniek. Een voorbeeld van een *informatiedragend signaal* is een signaal waarvan de *intensiteit* (niveau bij spanningen en sterkte bij stromen) niet van te voren vastligt en afhangt van de in het signaal opgeslagen informatie. (Er zijn ook andere manieren van informatie-opslag in signalen, bijvoorbeeld door frequentiemodulatie. Deze vormen laten we buiten beschouwing.)



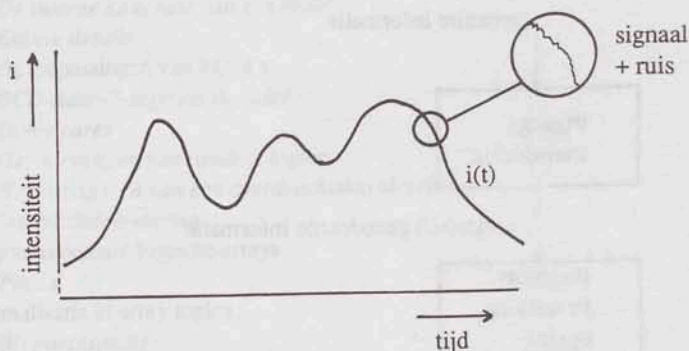
Figuur 0.1. Informatieverwerkend systeem.

Figuur 0.1 schetst de structuur van een elektronisch informatieverwerkend systeem. De primaire informatie (over temperatuur, druk, kracht, veld of beweging) wordt via een ingangstransducent omgezet in een elektrisch signaal, waarin de informatie gecodeerd is opgeslagen. Dit signaal wordt getransporteerd, bewerkt, opgeslagen en weer gereproduceerd in een informatieverwerkend systeem. Het resultaat is een al dan niet bewerkt elektrisch signaal, dat via een uitgangstransducent meestal weer wordt omgezet in een niet-elektrisch verschijnsel.

Bij het omzetten van informatie in een elektrisch signaal kunnen we kiezen uit verschillende signaaltvormen om de informatie te representeren. Deze zijn globaal onder te verdelen in tijdcontinue of tijddiscrete en in intensiteitcontinue of intensiteitsdiscrete signaaltvormen. Drie van de vier combinaties bekijken we hieronder wat uitgebreider.

Tijdcontinue/intensiteitcontinue signaalbewerking

Bij een *tijdcontinu/intensiteitcontinu signaal* is de intensiteit (momentele signaalwaarde ten opzichte van een bepaald referentieniveau) op elk moment van het interval waarin het signaal toegepast wordt van belang (tijdcontinu) en kan de intensiteit een willekeurige waarde in een bepaald interval aannemen (intensiteitcontinu). De informatie-inhoud is opgeslagen in de *vorm van het signaal*. Zie figuur 0.2. Het is derhalve van belang de vorm zo goed mogelijk te behouden bij het bewerken van tijdcontinue/intensiteitcontinue signalen. Iedere verandering van de vorm van het signaal kan de informatie-inhoud aantasten. De toepassing bepaalt hoeveel vervorming kan worden toegestaan.



Figuur 0.2. *Tijdcontinu/intensiteitcontinu signaal.*

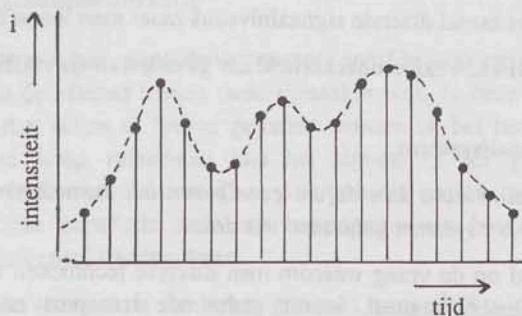
Elk signaal en dus ook een tijdcontinu/intensiteitcontinu signaal kan men opgebouwd denken uit een aantal sinusvormige signalen, waarbij elk signaal een verschillende frequentie, fase en intensiteit heeft. Het signaal zelf is dan de *superpositie* (momentele waarde van het signaal volgt uit de optelling van alle momentele signaalwaarden) van alle samenstellende sinusvormige signalen. Op grond van deze eigenschap bezit een signaal een *frequentiespectrum*, dit is een overzicht van alle frequenties van de samenstellende signalen [Papoulis, 1980; Oppenheim, 1983]. In theorie kan een frequentiespectrum zeer breed zijn. Bij praktische toepassingen benut men een deel van het spectrum en kapt men de rest af. Bij

telefonie loopt het frequentiespectrum traditioneel van 300 Hz tot 3400 Hz. Dit is voldoende om spraak over te dragen. Bij muziek loopt het frequentiespectrum van circa 30 Hz tot 20 KHz. De verschillen in bandbreedte hebben te maken met de eisen aan het systeem. Bij telefonie is het criterium een goede verstaanbaarheid. Bij muziek liggen de eisen veel hoger.

De fysica leert dat in elektrische systemen altijd verstoringen van signalen optreden. Deze komen onder andere voort uit de (statistische) fluctuaties van stromen in geleiders, *ruis* genoemd [Davidse, 1991]. Ook elektromagnetische koppeling met naburige geleiders zorgt voor een zekere beïnvloeding, die we *overspraak* noemen. Al deze verstoringen beïnvloeden de signaolvorm van tijdcontinue/intensiteitcontinue signalen. Een signaal kan dus nooit exact vormgetrouw overgedragen of bewerkt worden. Storingen in dezelfde frequentieband als het informatiedragende signaal zijn in veel gevallen niet of zeer moeilijk te onderscheiden van het signaal zelf. Storingen en signaal hebben min of meer dezelfde eigenschappen. Dit gegeven beperkt de afstand waarover tijdcontinue/intensiteitcontinue signalen getransporteerd kunnen worden, of de mate waarin zij gereproduceerd kunnen worden. Men heeft daarom gezocht naar andere signaolvormen, die beter bestand zijn tegen vervorming.

Tijddiscrete/intensiteitcontinue signaalbewerking

Volgens het *bemonsteringstheorema* van Shannon [Shannon, 1964] kan een elektrisch signaal volledig gekarakteriseerd worden door het signaal op equidistante tijdstippen te bemonsteren, mits de bemonstertijdstippen niet verder uit elkaar liggen dan de halve periode van de signaalcomponent met de hoogste frequentie in het spectrum van het signaal. Uit de bemonsterde waarden (samples genoemd) van het signaal kan het oorspronkelijke signaal gereconstrueerd worden [Shannon, 1964]. Een dergelijk bemonsterd signaal heet *tijddiscreet* en *intensiteitcontinu*. Bij een tijddiscreet signaal is de intensiteit slechts op bepaalde tijdstippen van belang. Ertussen is de signaalwaarde niet relevant. Figuur 0.3 beschrijft de tijddiscrete/intensiteitcontinue versie van het signaal uit figuur 0.2.



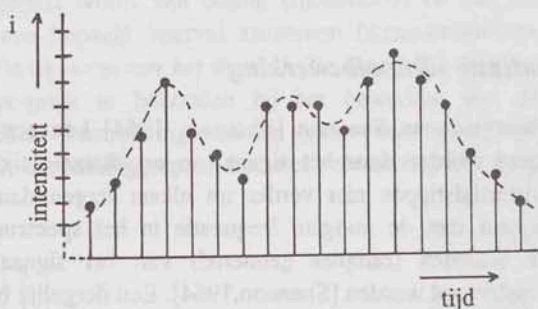
Figuur 0.3. Tijddiscreet/intensiteitcontinu signaal.

In een tijd- en intensiteitcontinu werkend systeem moet men er bij de bewerking van signalen rekening mee houden dat een signaal zowel in de tijd als in intensiteit continu

verandert. Bij een tijddiscreet signaal kunnen de samples na elkaar verwerkt worden. Samples kunnen ook in een aangepaste volgorde verzonden worden, mits na ontvangst de oorspronkelijke volgorde hersteld wordt. Kortom, meer vrijheid bij opslag en bewerkingen. Ruis en andere verstoringen hebben echter nog steeds invloed op de intensiteit van de samples en daarmee ook op de informatie-inhoud van het signaal.

Tijddiscrete/intensiteitdiscrete signaalbewerking

Het aantal intensiteitsniveaus van een tijddiscreet/intensiteitcontinu signaal is in principe oneindig groot. *Voor praktische toepassingen kan men volstaan met een beperkt aantal discrete niveaus, zonder een merkbaar kwaliteitsverlies.* De bemonsterde waarden worden dan afgebeeld op de dichtstbijzijnde discrete waarden. Zie figuur 0.4. Meestal gaat men niet verder dan een 16-bit kwantisatie, zijnde een indeling in $2^{16} = 65.536$ verschillende niveaus. Meer is dan niet zinvol, gezien de in het bronsignaal aanwezige onzekerheid als gevolg van onder meer ruis. Voor veel toepassingen volstaat men met minder bits. Een intensiteit-discreet signaal heet een *digitaal* signaal.



Figuur 0.4. Tijddiscreet/intensiteitdiscreet signaal.

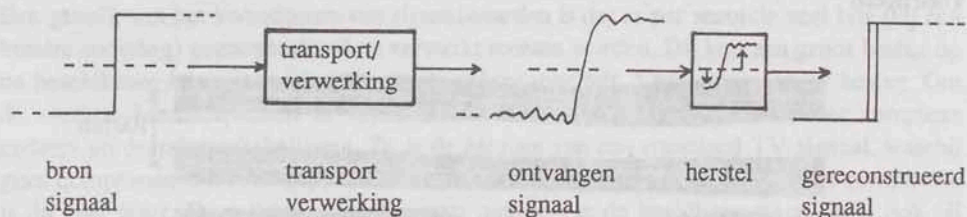
Bij een tijddiscreet/intensiteitdiscreet signaal verschillen de afgebeelde waarden van de bemonsterde waarden. Men noemt dit *kwantisatieruis*. Bij het beoordelen van het effect hiervan en bij de keuze van het aantal discrete signaalniveaus moet men letten op

- de van nature in het signaal aanwezige onzekerheid als gevolg van de reeds aanwezige ruis;
- de toepassing van het signaal/systeem.

Bij een voldoende groot aantal niveaus kan de als gevolg van het kwantiseren gemaakte fout binnen de toleranties van het systeem gehouden worden.

Figuur 0.5 geeft een antwoord op de vraag waarom men discrete technieken toepast. Elk signaal, dus ook een bemonsterd signaal, wordt gedurende transport en bewerking beïnvloed door wat we met ruis en overspraak hebben aangeduid, verzamelingen voor alle niet-gewenste en ook niet geheel te vermijden versturende invloeden. Bij een intensiteitdiscreet signaal nu is het mogelijk onderscheid te maken tussen de oorspronkelijke signaalwaarden en de op het signaal aanwezige ruis, mits het ruisniveau

niet te groot is. Als het ruisniveau minder is dan de helft van het verschil tussen twee discrete signaalniveaus kunnen storingen herkend worden en is het mogelijk de oorspronkelijke signaalwaarde te herstellen. Met het gereconstrueerde signaal kan de bewerking of het transport voortgezet worden. Merk op dat aldus de spanningsniveaus wel gereconstrueerd worden, maar de tijdstippen van de overgangen niet. We zullen later aantonen dat ook de beïnvloeding van de tijdstippen van de overgangen grotendeels kan worden gecompenseerd.



Figuur 0.5. Herstel van een intensiteitdiscreet signaal.

Bij intensiteitcontinue signalen spreekt men over de 'vervorming' als gevolg van verstoringen. Bij intensiteitdiscrete signalen verliest het begrip vervorming aan betekenis omdat elke vervorming beneden een bepaalde *drempelwaarde* weggefilterd kan worden. Bij intensiteitdiscrete signalen spreekt men daarom van de *foutkans*, de kans op een verkeerde klassifikatie, als maat voor de kwaliteit van het signaal of systeem.

Een belangrijke reden om over te gaan op tijd- en intensiteitdiscrete signaaltvormen is dat de invloed van ruis op intensiteitcontinue signalen cumuleert. Telkens als een intensiteitcontinu signaal bewerkt of getransporteerd wordt heeft ruis een versturende invloed. Daarnaast kan de momentele waarde van een intensiteitdiscreet signaal gemakkelijk opgeslagen worden door middel van een getal, als een bit string. Hierop zijn een groot scala van bewerkingen mogelijk, zoals het tijdelijk opslaan in een geheugen of het vergelijken met vorige waarden.

Binaire signaalbewerking

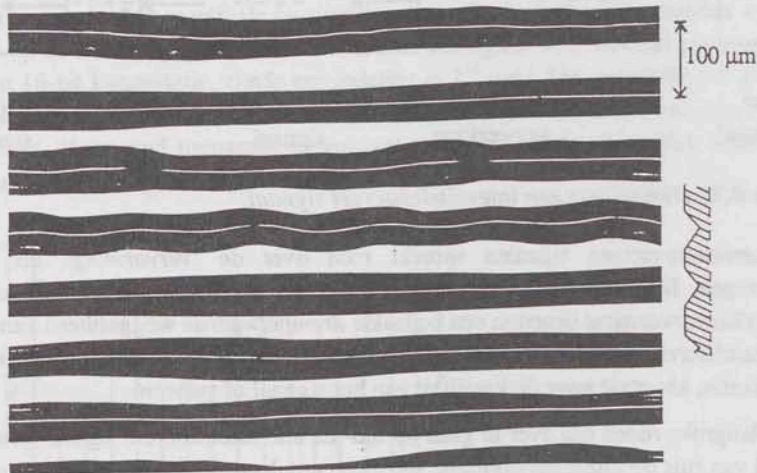
Het toelaatbare ruisniveau tijdens transport en/of bewerking van intensiteitdiscrete signalen hangt af van de afstand tussen twee signaalniveaus. Is deze te klein ten opzichte van het ruisniveau, dan zullen er fouten gemaakt worden bij het herstel van de niveaus. Immers, door een te hoog ruisniveau kan het signaal in het gebied rond de naastgelegen signaalwaarde komen. Betrouwbare discriminatie tussen twee niveaus is dan niet meer mogelijk. Deze conclusie stelt een bovengrens aan het aantal niveaus dat men kan onderscheiden en/of transporteren.

In de digitale techniek is de keuze gemaakt voor slechts *twee signaalniveaus*, een laag signaalniveau L en een hoog signaalniveau H. Deze keuze is vooral economisch bepaald. Bij slechts twee signaalniveaus is het mogelijk te werken met *zeer ruime toleranties* van de schakelende componenten. *Parameterveranderingen* als gevolg van temperatuurvariaties, wisselende belasting van een component, procesfluctuaties tijdens fabricage van

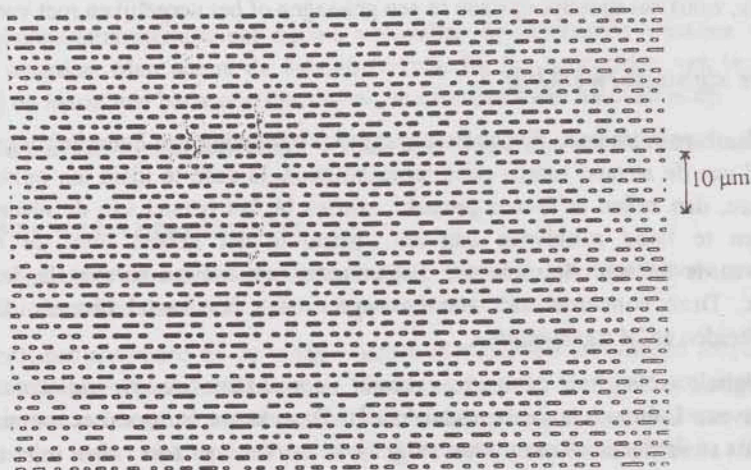
geïntegreerde circuits (IC's) en andere invloeden hebben dan binnen ruime grenzen geen invloed op de signaaloverdracht.

Bij slechts twee signaalniveaus is het noodzakelijk signalen en andere grootheden met een bepaald waardebereik te *coderen* met een voldoende aantal tweewaardige signalen. Hoofdstuk 1 gaat hier op in. Zo is een goed compromis mogelijk tussen *robuustheid* tegen verstoringen (twee niveaus) en *nauwkeurigheid* (veel discrete signaalniveaus).

Voorbeeld



Figuur 0.6. Grammofoonplaat (Foto W. Stravers).



Figuur 0.7. CD schijfje (10x meer vergroot) (Foto W. Stravers).

Figuur 0.6 laat zien hoe een muzieksignaal op de inmiddels klassieke grammofoonplaat is geregistreerd. De vorm van het signaal is gereproduceerd in de groef van de plaat. Bij het afspelen brengt de naald een spoeltje in een magnetisch veld in beweging, waardoor een met de vorm van de groef overeenkomend elektrisch signaal wordt opgewekt. Op een CD schijfje is de informatie daarentegen via wel/geen putje opgeslagen. Bij het aflezen hiervan ontstaat een signaal met twee signaalniveaus, met alle daaraan gekoppelde voordelen. Zie figuur 0.7. Ook zit er informatie in de duur van elk deelsignaal. [Pohlmann, 1988] □

Een gevolg van het kwantiseren van signaalwaarden is dat er per seconde veel bits (bij een binaire codering) getransporteerd en verwerkt moeten worden. Dit legt een groot beslag op de beschikbare *kanaalcapaciteit* en op de *rekencapaciteit*. Vaak een te groot beslag. Om de vereiste kanaalcapaciteit te verminderen kiest men voor compressie, met complexe codeer- en decodeerschakelingen. Zo is de *bit rate* van een standaard TV signaal, waarbij geen compressie wordt toegepast, 166 Mbit/s (dat zijn 166.000.000 bits/s). Met compressie is dit veel lager. Dergelijke technieken past men toe in de beeldbewerking, maar ook bij digitale audio.

Uit het bovenstaande mag niet de conclusie getrokken worden dat er voor tijdcontinue/intensiteitcontinue signaalbewerking (de analoge signaalbewerking) geen plaats meer is. Bij zeer grote bandbreedte van het frequentiespectrum, bij signalen met een zeer grote dynamiek, bij interface elektronica naar de analoge wereld en vele andere toepassingen blijft 'analoog' een reëel alternatief [Davidse, 1986; van Roermund, 1993]. Voor de minder kritische toepassingen gaat de voorkeur uit naar binaire signaalbewerking.

De keuze voor *binaire* (= tweewaardige) *signaalbewerking* heeft verstrekkende gevolgen. Alles wat we willen vastleggen of overdragen moet gecodeerd worden in binaire grootheden. Vaak ligt de keuze hoe dit te doen voor de hand. Bij getallen past men het *binaire talstelsel* toe, met als grondtal 2 (twee) in plaats van 10 (tien). Dit talstelsel is ook toepasbaar als we de waarde van een intensiteitsdiscreet signaal willen vastleggen. Met een 10-bit binaire code kunnen we 1024 signaalniveaus vastleggen en met een 16-bit binaire code 65.536.

Is niet zonder meer duidelijk met welke binaire code we een grootheid moeten vastleggen, dan is een tussenstap nodig. Via één of meer beweringen, die elk op zich slechts waar (TRUE) of niet waar (FALSE) kunnen zijn, wordt de grootheid of het verschijnsel gekenmerkt of geklassificeerd. Van elke bewering wordt nagegaan of deze daarbij waar of niet waar is. Waar stellen we in de digitale techniek voor door 1 en niet waar door 0. Aldus ontstaat een binaire code. Deze code heeft slechts betekenis als de beweringen, volgens welke de code opgesteld is, gegeven zijn en ondubbelzinnig geformuleerd zijn. Wat denkt u van de bewering

'De brug is open'

om de toestand van een beweegbare brug te omschrijven? Voor wie is de brug open? Deze bewering is dus niet geheel ondubbelzinnig.

Een bewering waarvan de waarheidswaarde FALSE (0) of TRUE (1) is, heet een *propositie*. Dit begrip komt uit de logica. Shannon heeft in 1938 [Shannon, 1938] voor het

eerst aangetoond dat het mogelijk is om de werking van schakelingen (in zijn geval elektromechanische schakelingen met kontakten) te beschrijven met behulp van begrippen uit de logica. Begrippen die eerder door Boole [Boole, 1854] geïntroduceerd waren. Het artikel van Shannon was het begin van een lange reeks toepassingen van de logica bij het ontwerpen van wat we daarom noemen *logische schakelingen*.

Voorbeeld

Kontakten in toetsen van een toetsenbord gaan geleiden indien er een kracht op de toets uitgeoefend wordt, in de vorm van het indrukken van de toets. Het verband tussen het indrukken van de toets en de werking van het kontakt beschrijven we met twee proposities

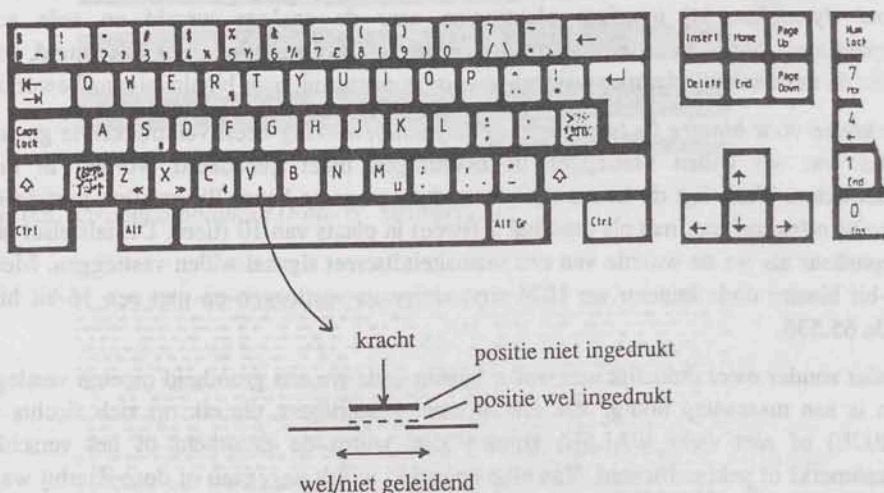
D 'De toets is ingedrukt'

S 'Het kontakt geleidt'

In het getekende geval geldt dat als D waar is ook S waar is en andersom, als D niet waar is dan is ook S niet waar. De formule

$$S = D$$

beschrijft dan formeel de werking van de getekende druktoets in figuur 0.8. □



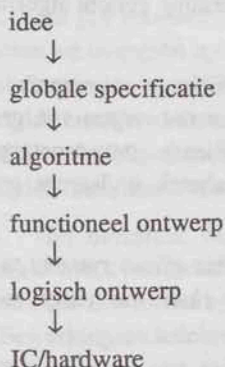
Figuur 0.8. Beschrijving van de werking van een toets (toetsenbord: IBM).

In dit voorbeeld is stilzwijgend een aantal veronderstellingen gemaakt. Zo mag de toets niet te hard worden ingedrukt. Te hard drukken vernietigt het toetsenbord. Te zacht drukken mag ook niet. Of te kort drukken. En vaak ook weer niet te lang. De onderliggende mechanische eigenschappen van het kontakt en de software waarmee het kontakt bemonsterd wordt spelen op het realisatieniveau een grote rol. Bij het *logisch ontwerp* van schakelingen met kontakten, waarbij het gaat om de *logische functionaliteit*, kunnen de mechanische eigenschappen voorlopig uit het gezichtsveld blijven.

Ook voor elektronische logische schakelingen geldt een aantal afspraken. Zo is er een bepaald gebied waarin een ingangsspanning als Hoog (H) wordt geïnterpreteerd en in een ander gebied als Laag (L). Ingangsspanningen mogen niet buiten de aldus gedefinieerde gebieden liggen. Houden we ons aan deze afspraken, dan kan een ontwerp met behulp van logische formules beschreven en verder uitgewerkt worden. Wanneer de grote lijnen van een ontwerp bekend zijn, dan volgt een stapsgewijze detaillering van het ontwerp en komen de fysische regels weer in zicht. Dan moet bijvoorbeeld uitgezocht worden hoe we moeten omgaan met veranderende signaalwaarden, tijdens een signaalovergang tussen het H en het L gebied.

Het ontwerptraject

Het *ontwerptraject* van een digitale schakeling bestaat uit een aantal opeenvolgende stappen:



Het ontwerp wordt stapsgewijs verfijnd totdat een realisatie van de schakeling mogelijk is. De eerste ontwerpstep, van het idee tot een *globale specificatie*, verloopt meestal langs intuïtieve weg. Deze specificatie, waarbij de ontwerpdoelstellingen worden vastgesteld, wordt meestal uitgedrukt in de natuurlijke taal.

Daarna zoekt men naar een algoritme, dat is een aantal min of meer exact beschreven regels die gevolgd moeten worden om het probleem op te lossen. Voor de beschrijving van het *algoritme* bedient men zich van formele talen en van de wiskunde. Men denkt in modules in de vorm van *black boxes* en *relaties* of *verbindingen* (interfaces) daartussen. De hierbij toegepaste wiskundige technieken zijn onder andere statistiek, wachtrijtheorie en grafentheorie.

Blijkt het algoritme te kloppen en zijn de geschatte prestaties (performance) voldoende, dan beeldt men het algoritme af op een *functioneel ontwerp*, in hardware termen zoals registers (data-opslag), tellers, logische operatoren en aritmetische operatoren. Bij deze afbeelding bewijst de automatentheorie zijn diensten.

Vervolgens worden de functies afgebeeld op de beschikbare *bouwblokken*. Dit kunnen eerder ontworpen hardware componenten zijn, series standaard IC's, maar ook bibliotheekcellen in een CAD systeem (computer aided design systeem). De theoretische

achtergronden voor dit afbeeldingsproces berusten op de schakelalgebra, dit is een kleine greep uit de logica c.q. de Boole algebra. Op dit moment komt ook de *timing verificatie* in zicht. Verloopt alles wel volgens de regels van de componenten? Is elk signaal tijdig beschikbaar? Daarnaast moet een schatting gemaakt worden van het benodigde vermogen, het benodigde oppervlak, etc.

Uiteindelijk moet de schakeling gerealiseerd worden. Alle elektronische eigenschappen van de componenten moeten nu in rekening gebracht worden. Op IC niveau worden de fysische eigenschappen vastgelegd in de vorm van *parameters*.

Het bovenstaande suggereert een strakke *top-down benadering* van het ontwerp van logische schakelingen. In de praktijk verloopt het ontwerpen via enkele iteraties over een of meer ontwerp niveaus. Met name is dit het geval indien op een lager ontwerp niveau blijkt dat een van de ontwerpdoelstellingen niet gehaald kan worden. Vaak zal dit de *performance* van een deelontwerp zijn. Een signaal is niet tijdig beschikbaar of een bewerking kost te veel tijd. Een aangepast en meer op parallel processing gericht algoritme kan dan uitkomst bieden.

Het is duidelijk dat tijdens dit heen en weer gaan tussen de verschillende ontwerp niveaus niet te veel details meegenomen kunnen worden, zeker niet bij het ontwerpen van grote systemen. Veel details maken de verificatie van de verschillende ontwerp stappen onoverzichtelijk en tijdrovend. Het vakgebied van de digitale techniek is daarom goed gekarakteriseerd met de volgende omschrijving [Ward, 1990]:

Digital engineering involves a self-imposed design discipline that allows systems to be analysed in the simple, abstract domain of logic rather than the vastly more complicated domain of underlying electronic principles.

Dit boek beperkt zich in hoofdlijnen tot de hiervoor met 'functioneel ontwerp' en 'logisch ontwerp' aangeduide ontwerp niveaus. Tevens gaan we in op aspecten van de afbeelding op IC's en andere vormen van hardware, waarvan kennis op de hogere niveaus gewenst is en de hogere niveaus daadwerkelijk beïnvloedt.

Literatuur

1. D. Bloembergen, *Consumentenelektronica: Snelle Modellen, Nooit Stuk*, De Ingenieur, april 1987, pp. 127-135.
2. G. Boole, *An Investigation into the Laws of Thought*, Dover Publications (Reprint of 1854).
3. J. Davidse, *Analoge Signaalbewerkingstechniek*, Delfse Uitgevers Maatschappij, Delft, 1991, hoofdstuk 6.
4. J. Davidse, *Analoge Elektronica in het IC tijdperk*, Elektronica, december 1986.
5. J. A. Dinklo, *Sleutelonderwerpen uit de Informatietechnologie*, Samsom Bedrijfsinformatie, Alphen, 1994.
6. A. V. Oppenheim, A. S. Willsky, and J. T. Young, *Signals and Systems*, Prentice-Hall, London, 1983, Ch. 4.
7. A. Papoulis, *Circuits and Systems, a Modern Approach*, Holt, Rinehart and Winston, Inc, Fort Worth, 1980.
8. K.C. Pohlmann, *Digitale Audio*, Kluwer Technische Boeken, Deventer, 1988.
9. A. H. M. van Roermund, *Over Integratie Gesproken*, Intreerede, Faculteit. ET, TU Delft, 1993.

10. C. E. Shannon, *A Symbolic Analysis of Relay and Switching Circuits*, Trans. Am. Inst. Electr. Eng., Vol. 57, 1938, pp. 713-724.
11. C. E. Shannon, and W. Weaver, *The Mathematical Theory of Communication*, Univ. of Illinois Press, Urbana, 1964.
12. S. A. Ward, and R. H. Halstead, *Computation Structures*, MIT Press, Cambridge, Massachusetts, 1990.

Opgaven hoofdstuk 0

0.1. Bij intensiteitcontinue signalen is ruis zeer moeilijk te onderscheiden van het signaal zelf. Dit komt omdat

- a. signaal en ruis min of meer dezelfde eigenschappen hebben.
- b. ruis de bemonstertijdstippen beïnvloedt.
- c. het ruisniveau veel kleiner is dan het signaalniveau.
- d. Geen van de vorige antwoorden is goed. Bij intensiteitcontinue signalen is ruis er juist gemakkelijk uit te filteren.

0.2. Om (bij een niet te hoog ruisniveau) ruis geheel uit een signaal te kunnen filteren moeten we overgaan op

- a. tijdcontinue/intensiteitcontinue signaalbewerking.
- b. tijddiscrete/intensiteitcontinue signaalbewerking.
- c. tijdcontinue/intensiteitcontinue of op tijddiscrete/intensiteitcontinue signaalbewerking.
- d. tijddiscrete/intensiteitdiscrete signaalbewerking.

0.3. Het omzetten van een intensiteitcontinu naar een intensiteitdiscreet signaal gaat gepaard met kwantisatieruis. Welke opmerking over de invloed van kwantisatieruis op een informatieverwerkend systeem is correct?

- a. Bewerkingstechnieken gebaseerd op intensiteitcontinue signalen zijn altijd nauwkeuriger.
- b. Bij een voldoende aantal signaalniveaus beïnvloedt kwantisatieruis de nauwkeurigheid van de signaalbewerking nauwelijks.
- c. Kwantisatieruis wordt geheel veroorzaakt door variaties in de bemonsterfrequentie. Deze moet daarom zeer constant zijn.
- d. Kwantisatieruis kan weggefilterd worden als het niveau ervan kleiner is dan de helft van het verschil tussen twee signaalniveaus.

0.4. De rol van verstorende invloeden op een intensiteitdiscreet signaal wordt uitgedrukt in de

- a. signaal-ruisverhouding.
- b. kwantisatieruisfactor.
- c. foutkans.
- d. kanaal- of opslagcapaciteit.

0.5. Binaire signaalbewerking is een vorm van discrete signaalbewerking. Welke van de volgende factoren speelt geen rol bij de keuze voor twee niveaus?

- a. Parameterveranderingen.
- b. De gewenste nauwkeurigheid bij bewerkingen.
- c. Procesfluctuaties bij het maken van IC's.
- d. Hoge bescherming tegen ruis.

0.6. Onderstaande beweringen hebben betrekking op de voordelen van digitale systemen boven systemen welke werken met analoge (continue) signalen/grootheden. Een van de beweringen is niet van toepassing. Welke?

- Bij digitale systemen behoeft de nauwkeurigheid van een uitkomst niet minder te zijn dan die van een bewerking in een systeem met continue grootheden, ondanks het feit dat bij het digitaliseren van signalen een afrondingsfout wordt gemaakt.
- Systemen die met discrete grootheden werken zijn automatisch betrouwbaarder dan systemen die met analoge grootheden werken.
- Digitaal werkende systemen zijn gemakkelijker reproduceerbaar dan analoog werkende systemen.
- Digitaal werkende systemen zijn minder gevoelig voor parameter variaties van componenten en spanningsbronnen.

0.7. Bij het vastleggen van de momentele waarde van een intensiteitcontinu signaal in een bitcombinatie wordt een afrondingsfout gemaakt (kwantisatieruis). Het te bemonsteren signaal kan variëren tussen 0 V en $V_{\max}$. De toegepaste omzetter beeldt de gemeten signaalwaarde af op de dichtstbijzijnde binaire waarde, waarbij zowel naar boven als naar beneden afgerond kan worden. Bij een registratie met tien bits is de hierbij optredende fout

- ca 0,2 % van de momentele signaalwaarde.
- ca 0,1 % van de momentele signaalwaarde.
- ca 0,05 % van de momentele signaalwaarde.
- afhankelijk van de momentele signaalwaarde kleiner dan, gelijk aan of (veel) groter dan de onder a t/m c genoemde percentages.

0.8. Bij het vastleggen van de momentele waarde van een intensiteitcontinu signaal in een bitcombinatie wordt een afrondingsfout gemaakt (kwantisatieruis). Het te bemonsteren signaal kan variëren tussen 0 V en $V_{\max}$. De toegepaste omzetter beeldt de gemeten signaalwaarde af op de dichtstbijzijnde binaire waarde, waarbij naar boven afgerond wordt. Om de hierbij optredende fout kleiner dan 0,1 % van $V_{\max}$ te kunnen houden vereist de binaire registratie minimaal

- negen bits.
- tien bits.
- elf bits.
- twalf bits.

Stelling 0

De digitale en de analoge techniek zijn tot elkaar veroordeeld.

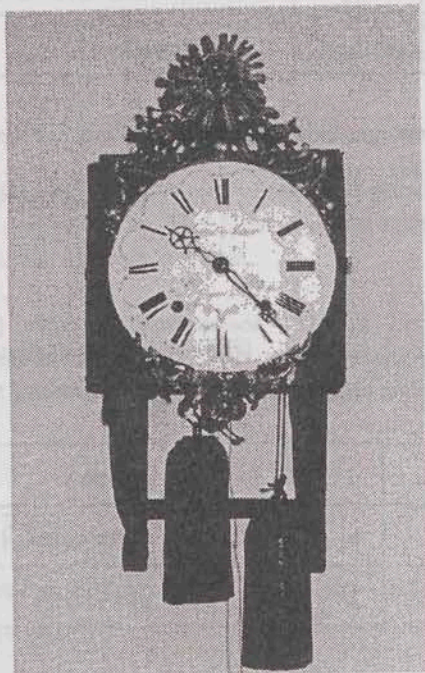
1

Getalrepresentaties en codes

1.1 Het decimale en het binaire talstelsel

Het *getalbegrip* maakt het ons mogelijk hoeveelheden aan te duiden en rekenkundige bewerkingen zoals optellen en aftrekken uit te voeren. Het voorstellen van getallen geschiedt niet altijd op dezelfde wijze. Vraag een kind naar zijn/haar leeftijd en er wordt een aantal vingers omhoog gestoken, één vinger voor elk jaar van zijn/haar leeftijd. De mogelijkheden van deze voorstellingswijze zijn beperkt en wel door het beschikbare aantal objecten, in dit geval tien vingers.

Door de eeuwen heen heeft men naar efficiëntere *getalrepresentaties* gezocht. Een voorbeeld hiervan is de wijze waarop Romeinen getallen voorstelden. Zij hadden aparte tekens die elk een zeker aantal aanduiden, bijvoorbeeld M ~ duizend en C ~ honderd. We vinden sporen hiervan nog terug op de wijzerplaten van antieke klokken. Zie voor een overzicht van de geschiedenis van het tellen [Boyer, 1968]. Het geven van 'namen' aan getallen wordt op een aardige manier besproken in [Scott, 1985].



Figuur 1.1. Uuraanduiding op antieke klokken.

Talstelsels

Het traditionele *decimale talstelsel* is een voorbeeld van een *positioneel talstelsel*. Het decimale talstelsel heeft als grondtal het getal 10 (tien) en de cijfers lopen van 0 t/m 9. (Het hoogste cijfer is gelijk aan het grondtal min 1.) De getallentheorie leert dat elk niet-negatief geheel getal hierin op precies één manier is voor te stellen. De getalrepresentatie in het decimale talstelsel is *uniek* [Weeg, 1960].

De *getalrepresentatie* in het decimale talstelsel bestaat uit een *geordende rij cijfers* (digits), waarbij aan elk cijfer een *gewicht* (weight) wordt toegekend en wel afhankelijk van de positie van het cijfer in de rij. Aldus wordt een gewogen codering verkregen. De gewichten zijn opklimmende machten van het *grondtal* (*base of radix*), te beginnen met 10^0 voor het rechter cijfer. Zo is

$$\begin{aligned} 723_{\text{DEC}} &= 7 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 \\ &= 700 + 20 + 3 \end{aligned}$$

Het subscript DEC geeft aan volgens *welk talstelsel* de rij cijfers moet worden geïnterpreteerd. Overeenkomstige regels gelden voor de interpretatie van talstelsels met een willekeurig geheel niet-negatief grondtal g :

$n-1$	$n-2$		1	0	← positie i
a_{n-1}	a_{n-2}		a_1	a_0	← cijfer a_i , $a_i \in \{0, 1, \dots, g-1\}$
g^{n-1}	g^{n-2}		g^1	g^0	← gewicht g^i

Het door n cijfers a_0 t/m a_{n-1} voorgestelde getal m stelt dan een hoeveelheid voor gelijk aan:

$$\begin{aligned} m &= a_{n-1}a_{n-2} \cdots a_1a_0 g \\ &= \sum_{i=0}^{n-1} a_i \cdot g^i \end{aligned} \tag{1.1}$$

In deze getalvoorstelling heet a_{n-1} het *hoogstwaardige cijfer* (*high-order* of *most significant digit*) en a_0 het *laagstwaardige cijfer* (*low-order* of *least significant digit*). Voor het in computers en digitale apparatuur veel toegepaste *binaire talstelsel* geldt:

$n-1$	$n-2$		1	0	← positie i
a_{n-1}	a_{n-2}		a_1	a_0	← cijfer a_i , $a_i \in \{0, 1\}$
2^{n-1}	2^{n-2}		2^1	2^0	← gewicht 2^i

waarbij

$$m = a_{n-1}a_{n-2} \cdots a_1a_0 \text{ BIN}$$

of

$$m = \sum_{i=0}^{n-1} a_i \cdot 2^i \quad (1.2)$$

Cijfers worden in het binaire talstelsel *bits* genoemd, een afkorting van *binary digit*.

De verzameling van de *natuurlijke getallen*, de *positive integers*, kan worden opgebouwd met behulp van de *axioma's van Peano* [Scott, 1985]. De bovenstaande talstelsels zijn weer een manier om de aldus gedefinieerde getallenverzameling voor te stellen.

Fracties

Op overeenkomstige wijze kunnen we fracties voorstellen. *Fracties* (delen van getallen kleiner dan 1) zijn voor te stellen middels *negatieve machten van het grondtal*. Zo stelt in een g -tallig talstelsel

$$0, a_{-1}a_{-2} \cdots a_{-k+1}a_{-k}$$

de fractie

$$\sum_{i=1}^{i=k} a_{-i} \cdot g^{-i} \quad (1.3)$$

voor. De hierbij te bereiken *nauwkeurigheid* wordt bepaald door g^{-k} . Bij fracties die geen veelvoud zijn van g^{-k} is een *afronding* noodzakelijk. Denk aan de voorstelling van $1/3$ in het decimale talstelsel, met bovendien een beperkt aantal cijfers!

Aantal cijfers

Voor de getalvoorstelling in talstelsels met verschillend grondtal g_1 respectievelijk g_2 zijn verschillende aantallen cijfers nodig. Het kleinste aantal cijfers is nodig bij het grootste grondtal. Bij een bereik van n decimale cijfers,

$$0 \leq \text{bereik} \leq 10^n - 1 \quad (1.4)$$

zijn m binaire cijfers nodig, met [Lewin, 1992]

$$2^m - 1 \geq 10^n - 1$$

of

$$m \geq \left\lceil \frac{n}{\log_{10} 2} \right\rceil = \left\lceil \frac{n}{0.301} \right\rceil \quad (1.5)$$

(De haken stellen afronding naar boven voor, op het eerstvolgende gehele getal.)

Bij drie decimale cijfers geldt

$$m \geq \lceil 9,9 \rceil \rightarrow m \geq 10 \text{ bits}$$

Voor de getalvoorstelling in computers werkt men standaard met veelvouden van 8 bits (*bytes*). In speciaal daarvoor ingerichte schakelingen geldt betrekking (1.5).

Bovenstaande getalrepresentaties zijn bedoeld voor *unsigned integers*, de niet-negatieve gehele getallen. De voorstelling van negatieve getallen in computers is niet zo eenvoudig. De behandeling hiervan stellen we uit tot paragraaf 1.8 en volgende.

De BCD representatie

Het grondtal 10 is belangrijk omdat we nu eenmaal gewend zijn hierin te rekenen. Het grondtal 2 is belangrijk vanwege het besluit in de digitale techniek van twee signaalniveaus uit te gaan. Een tussenvorm is de *BCD representatie*, de *Binary Coded Decimal* representatie, waarin men de facto blijft uitgaan van het decimale talstelsel, maar de cijfers binair noteert:

$$\begin{aligned} m &= a_2 a_1 a_0 \text{ DEC} \\ &= b_{11} b_{10} b_9 b_8 \ b_7 b_6 b_5 b_4 \ b_3 b_2 b_1 b_0 \text{ BCD} \end{aligned} \quad (1.6)$$

waarbij

$$a_0 \text{ DEC} = b_3 b_2 b_1 b_0 \text{ BIN}$$

$$a_1 \text{ DEC} = b_7 b_6 b_5 b_4 \text{ BIN}$$

$$a_2 \text{ DEC} = b_{11} b_{10} b_9 b_8 \text{ BIN}$$

Voorbeeld

$$937_{\text{DEC}} = 1001 \ 0011 \ 0111_{\text{BCD}} \quad \square$$

De gewichten van de bits in de BCD representatie lopen niet zuiver binair op. Zij verspringen bij de overgangen tussen de decimalen.

b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	bits BCD code
800				80				8				} gewichten (1.7) BCD code
400				40				4				
200				20				2				
100				10				1				

De BCD code is een voorbeeld van een *gewogen code*. Per decimaal cijfer hebben de bits een gewicht. Deze gewichten verhouden zich als 8 : 4 : 2 : 1. Daarom noemt men de BCD code wel een 8421 code. Bij andere decimale codes verhouden de gewichten zich als 2 : 4 : 2 : 1 [Wakerly, 1990; White, 1953].

Er bestaan ook talstelsels met een negatief grondtal. Wegens het geringe belang voor de praktijk behandelen we deze niet [Wadel, 1957].

1.2 Andere getalrepresentaties

Er bestaan nog andere getalrepresentaties. Enkele ervan volgen hierna. Sommige representaties, waaronder de hexadecimale representatie, zijn niet zozeer bedoeld om in te rekenen, maar dienen als kortschriftnotatie voor bit strings.

De hexadecimale representatie

In de *hexadecimale getalrepresentatie*, aangeduid als HEX, wordt een binaire getalrepresentatie in groepen van vier bits verdeeld, te beginnen bij het laagstwaardige bit. Elke groep van vier bits stelt dan (binair geïnterpreteerd) een getal tussen 0 en 15 voor. De gewichten van de groepen zijn oplopend $(2^4)^0$, $(2^4)^1$, $(2^4)^2$, ... Aldus ontstaat een getalrepresentatie met grondtal 16.

De hexadecimale representatie wordt gebruikt om *bit strings* aan te duiden. Elke groep van vier bits wordt door het bijbehorende hexadecimale cijfer voorgesteld. Omdat we met vier bits $2^4 = 16$ combinaties hebben en er decimaal slechts tien cijfers beschikbaar zijn worden de letters A t/m F aan de verzameling toegevoegd. Zie tabel 1.1.

0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

Tabel 1.1. Hexadecimale representatie van bit strings.

Voorbeeld

De bit string (of het binair gecodeerde getal)

1111100010110_{BIN}

wordt in de hexadecimale representatie genoteerd als

0001	1111	0001	0110	
↓	↓	↓	↓	1F16 _{HEX}
1	F	1	6	

Hiertoe zijn aan de linkerzijde drie nullen toegevoegd. □

De octale representatie

Zoals gezegd, de toepassing van de hexadecimale representatie beperkt zich tot het gebruik als kortschriftnotatie bij de voorstelling van bit strings. Tot enkele jaren terug paste men ook de *octale representatie* (OCT) toe. Hierbij werden groepjes van drie bits gevormd en elk groepje werd door het ermee corresponderende decimale cijfer (0 t/m 7) weergegeven.

Tegenwoordig werkt men standaard hexadecimaal met veelvouden van vier bits (*nibbles*). De octale representatie is in onbruik geraakt.

Tabel 1.2 geeft een overzicht van de representaties van de decimale getallen 0 t/m 20 in de hiervoor geïntroduceerde talstelsels. Boven elke kolom zijn de gewichten van de cijfers aangegeven.

DEC	BIN	OCT	HEX	BCD
10/1	16/8/4/2/1	8/1	16/1	80/40/20/10 8/4/2/1
00	00000	00	00	0000 0000
01	00001	01	01	0000 0001
02	00010	02	02	0000 0010
03	00011	03	03	0000 0011
04	00100	04	04	0000 0100
05	00101	05	05	0000 0101
06	00110	06	06	0000 0110
07	00111	07	07	0000 0111
08	01000	10	08	0000 1000
09	01001	11	09	0000 1001
10	01010	12	0A	0001 0000
11	01011	13	0B	0001 0001
12	01100	14	0C	0001 0010
13	01101	15	0D	0001 0011
14	01110	16	0E	0001 0100
15	01111	17	0F	0001 0101
16	10000	20	10	0001 0110
17	10001	21	11	0001 0111
18	10010	22	12	0001 1000
19	10011	23	13	0001 1001
20	10100	24	14	0010 0000

Tabel 1.2. Overzicht getallenrepresentaties.

De excess-3 representatie

In de *excess-3 representatie* wordt elk decimaal cijfer a_i weergegeven door de binaire code van $a_i + 3$. Zie tabel 1.3. De excess-3 code is wel toegepast bij het optellen van decimale getallen [Thijssen, 1990], om het carry transport bij de BCD optelling te versnellen.

decimaal cijfer	excess-3 representatie
0	0011 (3)
1	0100 (4)
2	0101 (5)
3	0110 (6)
4	0111 (7)
5	1000 (8)
6	1001 (9)
7	1010 (10)
8	1011 (11)
9	1100 (12)

Tabel 1.3. Excess-3 representatie.

m-uit-n codes

Behalve met genoemde codes worden getallen en andere karakterverzamelingen soms voorgesteld in *m-uit-n* codes. Met een 2-uit-5 code (in elke codecombinatie twee bits 1/drie bits 0) kunnen 10 combinaties gemaakt worden, genoeg om de decimale cijfers 0 t/m 9 voor te stellen.

Met een 3-uit-7 code kunnen 35 combinaties gemaakt worden. Een bekende 3-uit-7 code is het CCITT nr. 3 alfabet [CCITT, 1985], dat is toegepast voor radiotelegrafie [van Duuren, 1947]. Met het CCITT nr. 3 alfabet worden letters, cijfers en een aantal commando's gecodeerd.

Omdat de te coderen verzameling meer dan 35 karakters bevat heeft men in deze code twee combinaties gereserveerd voor een *letter/cijfer wissel*. Daarmee kan men instellen volgens welke van twee interpretatie-afspraken een ontvangen codecombinatie dient te worden geïnterpreteerd. Door de technische ontwikkelingen is dit soort trucs achterhaald.

De *m-uit-n* codes hebben *foutendetecterende eigenschappen*. Als na ontvangst van een 7-bit codecombinatie het aantal enen niet gelijk is aan *m* (drie bij een 3-uit-7 code), dan is er een fout opgetreden. Hierop kan een zekere bescherming tegen fouten worden gebaseerd.

Hiermee zijn de meest voorkomende codes voor rekenkundige en andere toepassingen besproken. Codes voor karakterverzamelingen waaronder de veel in computers toegepaste ASCII code bespreken we in paragraaf 1.5. Bij het vermenigvuldigen wordt de signed-digit representatie veel toegepast [Avizienis, 1961].

1.3 Grondtal conversie

In computers zijn de conversie van de decimale naar de binaire representatie en andersom veel voorkomende bewerkingen. Voor de conversie van een in het grondtal g_1 uitgedrukt getal *m* naar de representatie in het grondtal g_2 zijn verschillende algoritmes bekend. Bij een

ervan wordt het te converteren getal herhaald gedeeld door het grondtal g_2 met de rest r tussen [Couleur, 1958]

$$0 \leq r \leq g_2 - 1 \quad (1.8)$$

Deze deling wordt uitgevoerd in het oorspronkelijke talstelsel (grondtal g_1). De achtereenvolgens optredende resten zijn de cijfers voor de representatie in het talstelsel met het grondtal g_2 .

Voorbeeld

Converteer 47_{DEC} naar het binaire talstelsel. Dit gaat als volgt.

$$47 : 2 = 23 + \text{rest } 1 \rightarrow a_0 = 1$$

$$23 : 2 = 11 + \text{rest } 1 \rightarrow a_1 = 1$$

$$11 : 2 = 5 + \text{rest } 1 \rightarrow a_2 = 1$$

$$5 : 2 = 2 + \text{rest } 1 \rightarrow a_3 = 1$$

$$2 : 2 = 1 + \text{rest } 0 \rightarrow a_4 = 0$$

$$1 : 2 = 0 + \text{rest } 1 \rightarrow a_5 = 1 \Rightarrow 47_{\text{DEC}} = 101111_{\text{BIN}}$$

Toelichting:

Zij m een niet-negatief geheel getal, dat is weergegeven in het talstelsel met grondtal g . Dan geldt

$$\begin{aligned} m &= a_{n-1}a_{n-2} \cdots a_1a_0 g \\ &= a_{n-1} \cdot g^{n-1} + a_{n-2} \cdot g^{n-2} + \cdots + a_1 \cdot g^1 + a_0 \cdot g^0 \\ &= (a_{n-1} \cdot g^{n-2} + a_{n-2} \cdot g^{n-3} + \cdots + a_1) \cdot g + a_0 \end{aligned}$$

Dan is

$$m/g = a_{n-1} \cdot g^{n-2} + a_{n-2} \cdot g^{n-3} + \cdots + a_1 \quad (1.9)$$

het quotiënt q en is de rest r gelijk aan a_0 , met

$$0 \leq a_0 \leq g - 1 \quad (1.10)$$

Het verkregen *quotiënt* q heeft een gewicht g^1 en de *rest* r een gewicht g^0 . Deling van het quotiënt (1.9) door g levert een nieuw quotiënt

$$a_{n-1} \cdot g^{n-3} + a_{n-2} \cdot g^{n-4} + \cdots + a_2 \quad (1.11)$$

met gewicht g^2 en de rest

$$0 \leq a_1 \leq g - 1 \quad (1.12)$$

met een gewicht g^1 . Alle resten a_i , die bij herhaald delen door het grondtal g optreden, voldoen aan de grenzen in (1.12). De eerste rest heeft een gewicht g^0 , de volgende rest g^1 , daarna g^2 , enz. Het zijn de cijfers in het g -tallig talstelsel.

De deling van m door g kan in elk talstelsel worden uitgevoerd. De achtereenvolgens gevonden resten liggen tussen de grenzen (1.12) en kunnen direct en zonder enige conversie in het bedoelde talstelsel worden uitgedrukt. Deze procedure voor getalconversies kan bij elk grondtal worden toegepast. $\square$

Bij een ander algoritme worden de cijferverzameling en de gewichten van de cijfers van het talstelsel, waarin het te converteren getal is voorgesteld, uitgedrukt in het nieuwe talstelsel. Vervolgens worden alle produkttermen $a_i \cdot g^i$ in het nieuwe talstelsel uitgerekend. Daarna worden alle produkttermen opgeteld. Vooral voor de omzetting binair $\rightarrow$ decimaal is dit algoritme handig en wel omdat de cijfers dan 0 of 1 zijn. De evaluatie van de produkttermen is dan wel heel eenvoudig.

Voorbeeld

De omzetting van $m = 1011001_{\text{BIN}}$ naar de decimale representatie verloopt als volgt.

$$\begin{array}{rcl}
 m = 1011001_{\text{BIN}} & & \\
 \begin{array}{l} \text{---} \cdot 2^0 = 1 \\ \text{---} \cdot 2^1 = 0 \\ \text{---} \cdot 2^2 = 0 \\ \text{---} \cdot 2^3 = 8 \\ \text{---} \cdot 2^4 = 16 \\ \text{---} \cdot 2^5 = 0 \\ \text{---} \cdot 2^6 = 64 + \end{array} & & \\
 & & 89_{\text{DEC}}
 \end{array}$$

Bij deze conversie is een tabel nodig van decimaal gerepresenteerde machten van 2. Het is ook mogelijk deze tabel te genereren door de vorige macht van 2 te verdubbelen. $\square$

Bij de conversie van representaties van karakterverzamelingen maakt men wel gebruik van de *table lookup methode*. Van alle karakters wordt de nieuwe voorstelling bepaald of afgesproken en in een tabel opgeborgen. De oude codering dient hierbij (indirect) als adres van de tabel. Deze conversiemethode werkt zeer snel en wel omdat er tussendoor niet gerekend hoeft te worden. De methode wordt onder andere toegepast bij $\sin \alpha \leftrightarrow \cos \alpha$ conversies. Bij getallen is de table lookup methode omslachtig vanwege het grote geheugengebruik, dat evenredig is met het aantal/bereik van de te converteren getallen.

1.4 De conversie BIN-OCT-HEX

Tussen de binaire, de octale en de hexadecimale representatie verloopt de conversie zeer gemakkelijk. Dit komt omdat in deze talstelsels de grondtallen een macht van 2 zijn.

Voorbeeld

$$\begin{aligned}
 1F9A_{\text{HEX}} &= 1 \cdot 16^3 + F \cdot 16^2 + 9 \cdot 16^1 + A \cdot 16^0 \\
 &= 1 \cdot 2^{12} + F \cdot 2^8 + 9 \cdot 2^4 + A \cdot 2^0 \\
 &= 0001_{\text{BIN}} \cdot 2^{12} + 1111_{\text{BIN}} \cdot 2^8 + 1001_{\text{BIN}} \cdot 2^4 + 1010_{\text{BIN}} \cdot 2^0 \\
 &= 0001111110011010_{\text{BIN}} \quad \square \quad (1.13)
 \end{aligned}$$

De conversie van binair naar octaal berust op

$$\begin{aligned}
 m &= a_{n-1}a_{n-2} \dots a_1a_0_{\text{BIN}} \\
 &= a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \dots + a_1 \cdot 2^1 + a_0 \cdot 2^0 \\
 &= \dots + a_6 \cdot 2^0 \cdot 2^6 + (a_5 \cdot 2^2 + a_4 \cdot 2^1 + a_3 \cdot 2^0) \cdot 2^3 + (a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0) \cdot 2^0
 \end{aligned}$$

waaruit volgt

$$\begin{aligned}
 m &= \dots b_2 \cdot 2^6 + b_1 \cdot 2^3 + b_0 \cdot 2^0 \\
 &= \dots b_2b_1b_0_{\text{OCT}} \quad (1.14)
 \end{aligned}$$

waarbij

$$\begin{aligned}
 b_0 &= a_2a_1a_0_{\text{BIN}} \\
 b_1 &= a_5a_4a_3_{\text{BIN}} \\
 b_2 &= a_8a_7a_6_{\text{BIN}} \quad (1.15)
 \end{aligned}$$

enz. De octale cijfers kunnen direct gevonden worden door de bits van de binaire representatie in groepen van drie bits te verdelen en per groep deze bits binair te waarderen.

Eenzelfde redeneratie geldt voor de conversie van binair naar hexadecimaal:

$$\begin{aligned}
 m &= a_{n-1}a_{n-2} \dots a_1a_0_{\text{BIN}} \\
 &= \dots a_8 \cdot 2^0 \cdot 2^8 + (a_7 \cdot 2^3 + \dots + a_4 \cdot 2^0) \cdot 2^4 + (a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0) \cdot 2^0 \\
 &= \dots c_2 \cdot 2^8 + c_1 \cdot 2^4 + c_0 \cdot 2^0 \\
 &= \dots c_2 \cdot 16^2 + c_1 \cdot 16^1 + c_0 \cdot 16^0 \quad (1.16)
 \end{aligned}$$

voor het hexadecimale talstelsel, waarbij

$$\begin{aligned}
 c_0 &= a_3a_2a_1a_0_{\text{BIN}} \\
 c_1 &= a_7a_6a_5a_4_{\text{BIN}} \quad (1.17)
 \end{aligned}$$

enz.

Voorbeeld

$$\begin{aligned}
 101\,101\,111_{\text{BIN}} &= 101_{\text{BIN}} \cdot 2^6 + 101_{\text{BIN}} \cdot 2^3 + 111_{\text{BIN}} \cdot 2^0 \\
 &= 557_{\text{OCT}} \quad \square
 \end{aligned}$$

Voorbeeld

$$101101111_{\text{BIN}} = 0001_{\text{BIN}} \cdot 2^8 + 0110_{\text{BIN}} \cdot 2^4 + 1111_{\text{BIN}} \cdot 2^0 \\ = 16F_{\text{HEX}}$$

□

1.5 Codes voor alfanumerieke karakterverzamelingen

Een informatieverwerkend systeem moet naast getallen ook tekst, commando's, oproepen en signaleringen kunnen verwerken. Om dit type gegevens te kunnen verwerken zijn internationaal enkele codes voor *alfanumerieke karakterverzamelingen* gestandaardiseerd, waaronder de ASCII code.

De ASCII code

ASCII				commando's		karakters			
				b ₆		0	0	1	1
				b ₅		0	0	0	1
				b ₄		0	1	0	1
b ₃	b ₂	b ₁	b ₀						
0	0	0	0	NUL	DLE	SPACE	0	@	P
0	0	0	1	SOH	DC1	!	1	A	Q
0	0	1	0	STX	DC2	"	2	B	R
0	0	1	1	ETX	DC3	#	3	C	S
0	1	0	0	EOT	DC4	\$	4	D	T
0	1	0	1	ENQ	NAK	%	5	E	U
0	1	1	0	ACK	SYN	&	6	F	V
0	1	1	1	BEL	ETB	'	7	G	W
1	0	0	0	BS	CAN	(	8	H	X
1	0	0	1	HT	EM	)	9	I	Y
1	0	1	0	LF	SUB	*	:	J	Z
1	0	1	1	VT	ESC	+	;	K	[
1	1	0	0	FF	FS	,	<	L	\
1	1	0	1	CR	GS	-	=	M	]
1	1	1	0	SO	RS	.	>	N	^
1	1	1	1	SI	US	/	?	O	_

Tabel 1.4. De ASCII code

De *American Standard Code for Information Interchange* (ASCII code; spreek uit als askie) is een vaak toegepaste code voor computers en digitale apparatuur. De code is in tabel 1.4 gespecificeerd. De eerste twee kolommen betreffen commando's die op systeemniveau betekenis hebben. De overige kolommen duiden de codering aan van

hoofd/kleine letters, cijfers en een groot aantal tekens. Zo is de hoofdletter F gecodeerd (zie tabel 1.4) als

$$\square \quad F \leftrightarrow b_6b_5b_4b_3b_2b_1b_0 = 10001110$$

Voor de betekenis van de commando's verwijzen we naar [Wakerly, 1990]. De afspraken voor de ASCII code zijn vastgelegd in de standaard X.34-1968 van het American National Standards Institute.

Codes zoals de ASCII code zijn niet willekeurig toegekend. Zo is het gemakkelijk na te gaan of een ASCII karakter een cijfer voorstelt en wel door de codecombinatie te testen op

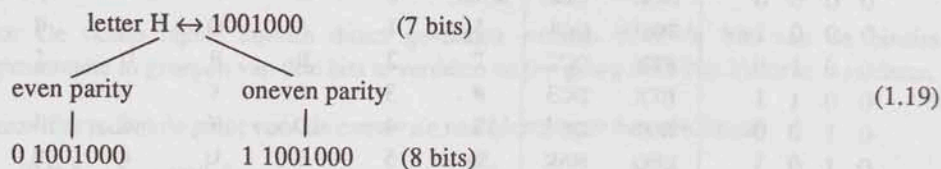
$$0110000_{\text{BIN}} \leq \text{ASCII karakter} \leq 0111001_{\text{BIN}} \quad (1.18)$$

Hiervoor zijn twee vergelijkingen nodig. Bij een random toewijzing van de codering van de cijfers zouden dit er maximaal tien zijn.

Parity

De ASCII code is een 7-bit code. Omdat men in computers gewoonlijk met *veelvouden van acht bits* werkt is er één bitpositie over. Deze bitpositie wordt meestal gebruikt om aan een ASCII karakter een *parity bit* toe te voegen, een bit dat het aantal enen in elk 8-bit woord even resp. oneven maakt. Zie hoofdstuk 6 voor een verdere toelichting.

Voorbeeld



Wanneer na ontvangst van een ASCII karakter met parity bit het aantal enen niet even resp. oneven is, dan is er zeker een fout opgetreden. De ASCII code + parity heeft hierdoor *foutendetecterende eigenschappen*. Zonder parity bit heeft de ASCII code deze niet. $\square$

De EBCDIC code

Naast de ASCII code bestaat de EBCDIC (Extended BCD Interchange Code). Zie tabel 1.5. De code kent meer karakters dan de ASCII code. Het is een 8-bit code. IBM heeft deze code toegepast. Het toevoegen van een parity bit aan een EBCDIC gecodeerd karakter heeft als nadeel dat er negen bits voor een karakter nodig zijn.

Hiermee hebben we de belangrijkste codes voor karakterverzamelingen gehad. In het verleden zijn veel andere codes toegepast, waaronder de *Morse code* voor het scheepvaartverkeer [Gilbert, 1969]. Zie verder [Gosling, 1980; Zeckendorf, 1973; Zemanek, 1965].

EBCDIC	b ₇	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	b ₆	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	b ₅	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	b ₄	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	b ₃ b ₂ b ₁ b ₀																
	0 0 0 0	NUL	DLE	DS		SP	&	_						{	}	\	0
	0 0 0 1	SOH	DC1	SOS				/		a	j	~		A	J		1
	0 0 1 0	STX	DC2	FS	SYN					b	k	s		B	K	S	2
	0 0 1 1	ETX	TM		—					c	l	t		C	L	T	3
	0 1 0 0	PF	RES	BYP	PN					d	m	u		D	M	U	4
	0 1 0 1	HT	NL	LF	RS					e	n	v		E	N	V	5
	0 1 1 0	LC	BS	ETB	UC					f	o	w		F	O	W	6
	0 1 1 1	DEL	IL	ESC	EOT					g	p	x		G	P	X	7
	1 0 0 0	GE	CAN							h	q	y		H	Q	Y	8
	1 0 0 1	RLF	EM							i	r	z		I	R	Z	9
	1 0 1 0	SMM	CC	SM		Ç	!	i	:								
	1 0 1 1	VT	CU1	CU2	CU3	.	\$	,	#								
	1 1 0 0	FF	IFS		DC4	<	*	%	@					}			
	1 1 0 1	CR	IGS	ENQ	NAK	(	)	_	'								
	1 1 1 0	SO	IRS	ACK		+	;	>	=					Y			
	1 1 1 1	SI	IUS	BEL	SUB	/	¬	?	"								EO

Tabel 1.5. De EBCDIC code.

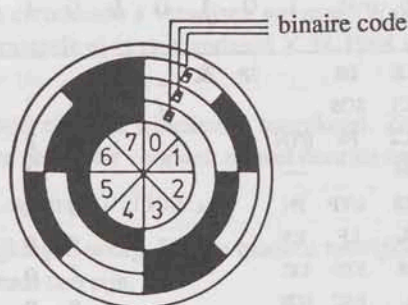
1.6 Equidistante codes

In elektro-mechanische toepassingen wordt voor de positiebepaling van een as wel een *codeschijf* toegepast (of bij een translatiemeting een *codelineaal*). Zo'n schijf bestaat uit een aantal verschillende segmenten, waarvan de codering door middel van lampjes of kontakten kan worden uitgelezen. Figuur 1.2 toont een codeschijf in de gewone binaire code.

Een *binaire codeschijf* heeft het voordeel dat de positie direct binair gecodeerd is. Verder heeft deze codeschijf grote nadelen en wel vanwege *overgangsverschijselen* op de overgangen van de segmenten. Bij de overgang van segment 7 (111) naar segment 0 (000) veranderen alle bits van de schijf. Staan de borstels resp. de lampjes niet precies uitgelijnd, dan kunnen bij de overgang van segment 7 naar segment 0 alle andere codecombinaties als overgangstoestand optreden. Welke codecombinaties in feite optreden hangt af van de uitlijning van de borstels/lampjes en de richting van de overgang. Ook bij andere overgangen treden dergelijke overgangsverschijselen op.

De codeschijf in figuur 1.3 heeft geen last van overgangsverschijselen dank zij de toegepaste progressieve code. Een *progressieve code* is een code waarin elk tweetal opeenvolgende codecombinaties in precies één bit verschilt. Een progressieve code is een

bijzonder geval van een *equidistante code*, een code waarin bij elke overgang tussen twee naburige posities of codewoorden altijd evenveel bits tegelijk veranderen.



Figuur 1.2. Codeschijf in de binaire code.

De *Gray code* of *gespiegeld binaire code* [Gray, 1953; Foss, 1954] is een progressieve code die ontstaat door 2^k eerder toegekende combinaties te spiegelen onder toevoeging van een extra bit. Dit bit is 0 voor de reeds toegekende codecombinaties en 1 voor de nieuwe combinaties. Zie tabel 1.6.



Figuur 1.3. Codeschijf in de Gray code (gespiegeld binaire code).

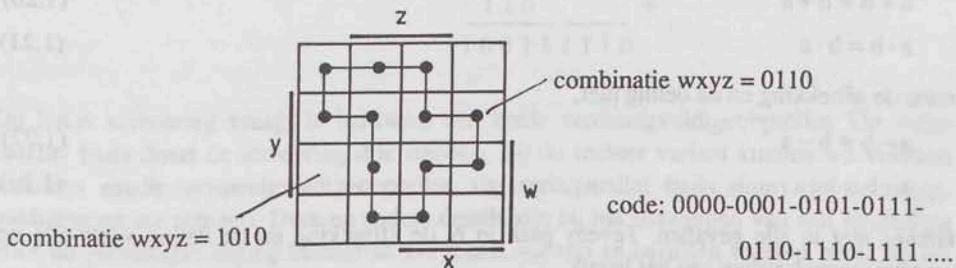
Bij het cyclisch doorlopen van een reeks codecombinaties is een progressieve codering slechts mogelijk bij een *even cycluslengte*. De cycluslengte hoeft niet noodzakelijk een macht van 2 te zijn. Zo kan een Gray code voor andere cycluslengtes ontstaan door het middenstuk (symmetrisch) weg te laten. Neem als voorbeeld de combinaties 0 t/m 4 gevolgd door 11 t/m 15 uit tabel 1.6. Ook deze cyclus van 10 combinaties is progressief gecodeerd.

Andere progressieve codes kunnen in een *Karnaughdiagram* gevonden worden. In een Karnaughdiagram wordt aan elk hokje een codecombinatie toegekend en wel afhankelijk of het hokje onder de met een variabele aangeduide streep valt (1) of niet (0). Zie de in figuur 1.4 benoemde hokjes als voorbeeld. Elke rondgang door zo'n Karnaughdiagram levert een

progressieve code op. Aldus kunnen progressieve codes van willekeurige even lengte geconstrueerd worden. Zie verder hoofdstuk 3 voor de details over de constructie van Karnaughdiagrammen.

patronen				combinatie
0	00	000	0000	0
1	01	001	0001	1
	11	011	0011	2
	10	010	0010	3
		110	0110	4
		111	0111	5
		101	0101	6
		100	0100	7
			1100	8
			1101	9
			1111	10
			1110	11
			1010	12
			1011	13
			1001	14
			1000	15

Tabel 1.6. Opbouw gespiegeld binaire code.



Figuur 1.4. Karnaughdiagram met progressieve code (12 combinaties).

Er bestaan ook *equidistante codes*, waarbij per overgang een ander aantal bits verandert, bijvoorbeeld telkens twee bits. Zie figuur 1.5 voor de constructie van een equidistante code waarbij twee bits tegelijk veranderen. We stappen daartoe met twee hokjes tegelijk door het diagram.

$$a \cdot (b + c) = a \cdot b + a \cdot c \quad (1.26)$$

Verder geldt voor rekenkundige bewerkingen een *prioriteitsvolgorde*. Deze is

machtsverheffen	<u>Hoogste prioriteit</u>
vermenigvuldigen	↓
delen	
worteltrekken	
optellen	↓
afrekken	<u>laagste prioriteit</u>

Met *haakjes* is het mogelijk de prioriteitsvolgorde te wijzigen.

Zoals gezegd, bovenstaande spelregels gelden ook in het binaire talstelsel. Het volgende voorbeeld toont het belang aan van de commutatieve wet bij de implementatie van de vermenigvuldiging.

Voorbeeld

Op grond van de commutatieve wet (1.21) mag de vermenigvuldiging van 110101_{BIN} met 110_{BIN} worden uitgevoerd als

$ \begin{array}{r} 110101 \\ \underline{110 \times} \\ 000000 \\ 110101 \\ \underline{110101 \ +} \\ 100111110 \end{array} $	of als	$ \begin{array}{r} 110 \\ \underline{110101 \times} \\ 110 \\ 000 \\ 110 \\ 000 \\ 110 \\ \underline{110 \ +} \\ 100111110 \end{array} $
---	--------	--

De linker uitvoering vraagt in hardware een brede vermenigvuldiger/opteller. Op serie-parallel basis duurt de uitvoering drie stappen. Bij de rechter variant kunnen we volstaan met een smalle vermenigvuldiger/opteller. Op serie/parallel basis duurt de vermenigvuldiging nu zes stappen. Deze en andere details zijn bij het ontwerpen van een schakeling voor de vermenigvuldiging belangrijk. De regels waarop ze berusten moeten reeds bij het opstellen van de algoritmes geverifieerd worden. Over top-down benadering gesproken! □

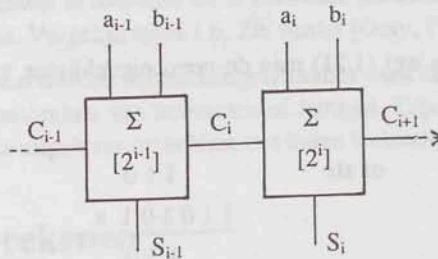
Met dit voorbeeld van de vermenigvuldiging zijn we automatisch op een voorbeeld van het rekenen in het binaire talstelsel gekomen. Tabel 1.7 specificeert de optelling van twee bits a_i en b_i van twee op te tellen binaire getallen A en B. In de tabel stellen a_i en b_i bits met gewicht 2^i voor en is C_i de inkomende carry (transport) met eveneens het gewicht 2^i . Het bit S_i is het te berekenen sombit, met gewicht 2^i . De uitgaande carry C_{i+1} heeft het gewicht 2^{i+1} . Figuur 1.6 schetst enkele secties van de optelschakeling. Het gewicht van elke sectie

staat tussen vierkante haken. (In tekeningen houden we meestal de standaard links-naar-rechts richting van de datastroom aan. Dit in tegenstelling tot de optelling 'op papier'.)

a_i	b_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$a_i / b_i / C_i / S_i$: gewicht 2^i
 C_{i+1} : gewicht 2^{i+1}

Tabel 1.7. Opteltabel.



Figuur 1.6. Optelsectie (2 bits).

Voorbeeld

$$\begin{array}{r}
 A \quad 1011101 \\
 B + \quad 1110100 \\
 \hline
 A+B \quad 11010001
 \end{array}$$

Deze optelling is met behulp van tabel 1.7 gemakkelijk te verifiëren. De laagstwaardige sectie met gewicht 2^0 bezit geen inkomende carry. Daarom heeft C_0 de 'default' waarde $C_0 = 0$. $\square$

Tot zover de overeenkomsten met het rekenen in N. Bij de implementatie van een bewerking in een schakeling moeten we bovendien rekening houden met praktische randvoorwaarden. Zo is in het binaire talstelsel bij n bits het *getalbereik* beperkt tot

$$0 \leq \text{bereik} \leq 2^n - 1 \quad (1.27)$$

voor niet-negatieve gehele getallen en tot

$$-2^{n-1} + 1 \leq \text{bereik} \leq 2^{n-1} - 1 \quad (1.28)$$

bij de binaire getalvoorstelling van positieve en negatieve gehele getallen met teken-en-modulus. Dit beperkte bereik geldt niet alleen voor de operanden in een bewerking, maar ook voor het resultaat [Huang, 1975]. Uit het vorige voorbeeld is gebleken dat bij de

optelling het resultaat één bit breder kan zijn dan de voorstelling van de operanden zelf. Indien het gereserveerde bereik hierin niet voorziet, dan treedt er capaciteitsoverschrijding op.

Capaciteitsoverschrijding volgt niet uit het algoritme voor de optelling, maar is een gevolg van de beperking van het aantal bits bij de implementatie in een schakeling. Het verschijnsel capaciteitsoverschrijding is implementatie-afhankelijk.

1.8 Getalrepresentaties in $\mathbb{Z}$

Negatieve gehele getallen worden op papier gewoonlijk weergegeven in *teken-en-modulus* (*signed-magnitude representation*). Hierbij wordt het positief/negatief zijn van een getal weergegeven met een teken $+/-$. De *modulus* geeft de absolute grootte van het getal aan. De binaire representatie van een geheel getal m uit $\mathbb{Z}$ in teken-en-modulus is een bitcombinatie

$$m \leftrightarrow a_{n-1}a_{n-2} \dots a_1a_0$$

en wel zodanig dat

$$\begin{array}{c}
 a_{n-1}a_{n-2} \dots a_1a_0 \\
 \swarrow \quad \searrow \\
 \begin{array}{l}
 a_{n-1} \text{ teken} \\
 | \\
 a_{n-1} = 0 \leftrightarrow + \\
 a_{n-1} = 1 \leftrightarrow -
 \end{array}
 \quad
 \begin{array}{l}
 a_{n-2}a_{n-3} \dots a_1a_0 \text{ BIN modulus} \\
 | \\
 |m| = a_{n-2}a_{n-3} \dots a_1a_0 \text{ BIN} \\
 = \sum_{i=0}^{n-2} a_i \cdot 2^i
 \end{array}
 \end{array}
 \quad (1.29)$$

Bij een bitcombinatie van n bits is het bereik van de voor te stellen getallen (1.28)

$$-2^{n-1} + 1 \leq m \leq 2^{n-1} - 1$$

Het bereik is naar beneden en naar boven toe beperkt. Bij rekenkundige bewerkingen kan daarom *capaciteitsoverschrijding* optreden. Het bereik is gemakkelijk uitbreidbaar door meer bits te nemen. Het tekenbit verandert daarbij van positie!

Voorbeelden

$$\begin{array}{ll}
 -6 \leftrightarrow 1110 \quad (n=4); & +21 \leftrightarrow 010101 \quad (n=6); \\
 -6 \leftrightarrow 10110 \quad (n=5); & +21 \leftrightarrow 0010101 \quad (n=7); \\
 -6 \leftrightarrow 100110 \quad (n=6); & +21 \leftrightarrow 00010101 \quad (n=8).
 \end{array}$$

Het linkerbit is altijd het tekenbit. □

Het *bereik* in teken-en-modulus is *symmetrisch*. Van elk getal behoort ook het tegengestelde ervan tot het bereik:

$$m \in \text{bereik (1.28)} \leftrightarrow -m \in \text{bereik (1.28)} \quad (1.30)$$

De bewerking aftrekken is daarom altijd gedefinieerd (mits er gezien de randvoorwaarde van het beperkte bereik geen capaciteitsoverschrijding optreedt).

De teken-en-modulus representatie kent verder een *dubbele representant* van 0, te weten

$$+0 \leftrightarrow 00 \dots 00 \quad (1.31.a)$$

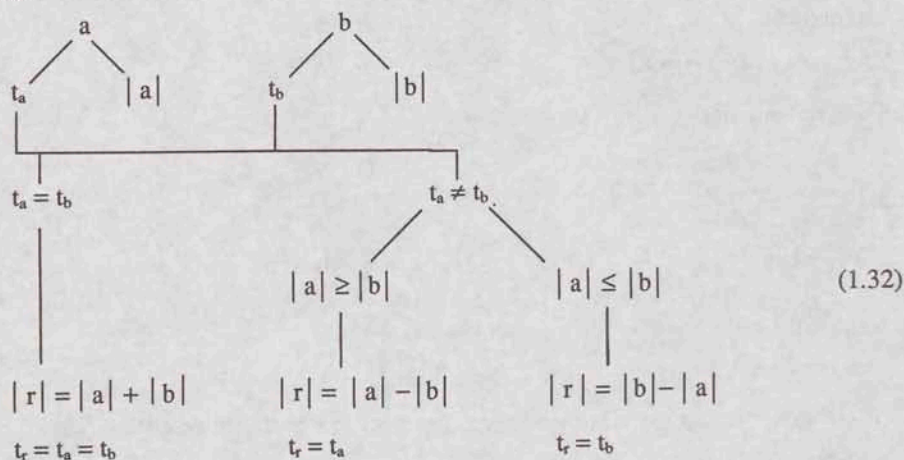
$$-0 \leftrightarrow 10 \dots 00 \quad (1.31.b)$$

De overige getallen hebben één representant. Met n bits kunnen we dus (maximaal) $2^n - 1$ getallen voorstellen.

Rekenen in teken-en-modulus

Voor de optelling is het schema (1.32) van toepassing.

Schema



Toelichting:

We splitsen de getallen a en b in teken t_a en t_b en modulus $|a|$ en $|b|$. Zijn de tekenbits gelijk, dan is de modulus van het resultaat, $|r|$, gelijk aan

$$|r| = |a| + |b|$$

Het tekenbit van het resultaat, t_r , is dan gelijk aan t_a of t_b . Op overeenkomstige wijze kunnen de andere twee takken geïnterpreteerd worden.

Voorbeeld

De binaire uitvoering van de optelling van $+13 + (-18)$ op zes bits verloopt als volgt.

$$\begin{aligned} a &\leftrightarrow 001101 & t_a &= 0 \\ & & |a| &= 01101_{\text{BIN}} \\ b &\leftrightarrow 110010 & t_b &= 1 \\ & & |b| &= 10010_{\text{BIN}} \end{aligned}$$

De tekenbits zijn ongelijk, we moeten de moduli aftrekken. Het teken van het verschil van $|a|$ en $|b|$ bepaalt of $|a|$ van $|b|$ dan wel $|b|$ van $|a|$ moet worden afgetrokken:

$$\begin{array}{r}
 |a| \quad 01101 \\
 |b| - \quad 10010 - \\
 \hline
 111011 \\
 \uparrow \\
 \text{leen met gewicht } -2^5
 \end{array}$$

Er is een leen met gewicht -2^5 , hetgeen betekent dat $|a| - |b| < 0$ was. Om $|r|$ te vinden moeten we in (1.32) de moduli verwisselen.

$$\begin{array}{r}
 |b| \quad 10010 \\
 |a| - \quad 01101 - \\
 \hline
 000101 \\
 \uparrow \\
 \text{geen leen}
 \end{array}$$

Volgens het schema is de teken-en-modulus representatie van het resultaat r :

$$100101 \leftrightarrow r = -5$$

□

Al met al levert dit geen prettige procedure op om in een digitale schakeling of in software uit te voeren. De schakeling zal relatief gecompliceerd zijn omdat

- de tekenbits op een andere wijze behandeld moeten worden dan de bits van de moduli;
- de schakeling bewerkingen moeten kunnen uitvoeren die corresponderen met optellen en aftrekken;
- het teken van het verschil van $|a|$ en $|b|$ bepaalt of $|b|$ van $|a|$ dan wel $|a|$ van $|b|$ moet worden afgetrokken.

Om deze redenen gaat men meestal over op een andere representatie van gehele getallen door bitcombinaties. Een ervan is het two's complement.

1.9 De two's complement representatie

Uit de begintijd van de digitale techniek stammen twee getalvoorstellingen, het two's complement en het one's complement. Bij deze twee getalvoorstellingen worden de positieve/negatieve getallen uit een bepaald bereik in $\mathbb{Z}$ afgebeeld op een ander niet-negatief domein [Sklar, 1972]. De uitvoering van rekenkundige bewerkingen in dit nieuwe domein kan veel gemakkelijker in een logische schakeling worden omgezet dan bij de teken-en-modulus representatie het geval is. We beperken ons hierna tot een samenvatting van de

eigenschappen en afspraken van het two's complement, de meest toegepaste representatie. In een later hoofdstuk volgt de formele onderbouwing van het two's complement rekenen.

De representatie

In het two's complement worden de gehele getallen $m \in \mathbb{Z}$ met het bereik

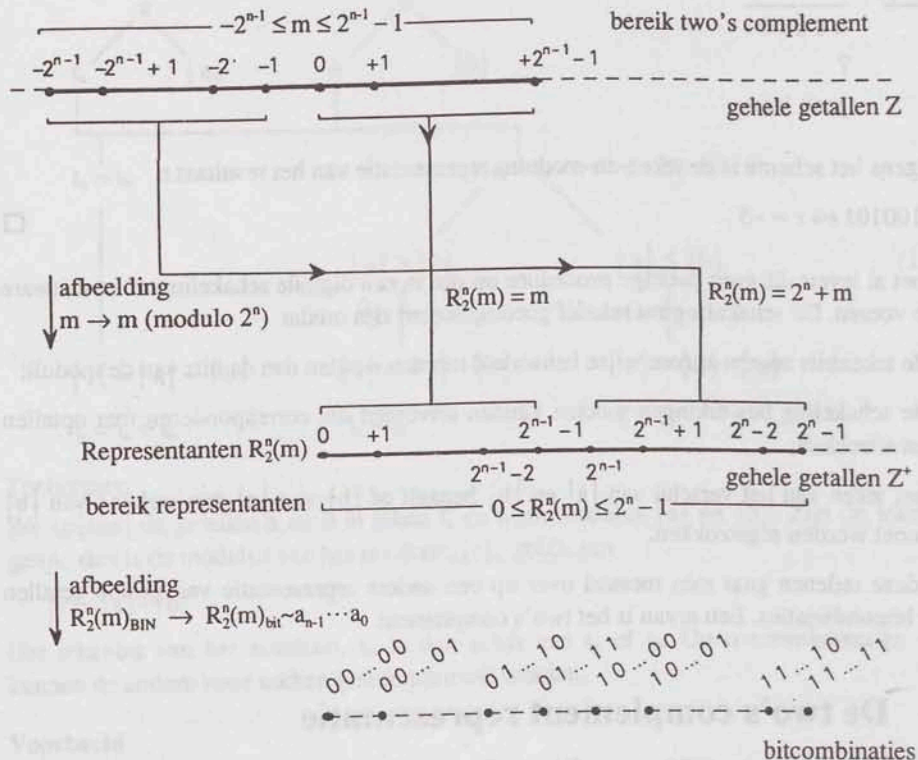
$$-2^{n-1} \leq m \leq 2^{n-1} - 1 \quad (\text{bereik } m) \quad (1.33)$$

gerepresenteerd door een (meestal binair gecodeerd) geheel getal $R_2^n(m)$, waarbij

$$0 \leq m \leq 2^{n-1} - 1 \iff R_2^n(m) = m \quad (1.34)$$

$$-2^{n-1} \leq m \leq -1 \iff R_2^n(m) = 2^n + m \quad (1.35)$$

Door deze transformatie zijn de getallen $R_2^n(m)$ *niet-negatief*. Zie ook figuur 1.7.



Figuur 1.7. De two's complement representatie.

In de representant $R_2^n(m)$ duidt m het te representeren getal aan, het superscript n geeft het aantal bits van de binaire representant aan of zo u wilt het bereik (1.33), en het subscript 2 duidt op het two's complement. Dit laatste omdat we ook andere getalvoorstellingen zullen introduceren, waaronder het one's complement.

Het zou te ver voeren nu al de keuze voor deze transformatie te motiveren. Voorlopig zien we (1.34) en (1.35) daarom als de 'definitie' van het two's complement.

De binaire voorstelling van de getallen $R_2^n(m)$ duiden we aan met $R_2^n(m)_{\text{BIN}}$:

$$R_2^n(m) \leftrightarrow R_2^n(m)_{\text{BIN}} = a_{n-1}a_{n-2} \cdots a_1a_0_{\text{BIN}} = \sum_{i=0}^{n-1} a_i \cdot 2^i \quad (1.36)$$

Tabel 1.8 geeft de two's complement representatie voor een 4-bit getalvoorstelling. Volgens (1.33) liggen de voor te stellen getallen m dan tussen

$$-8 \leq m \leq +7 \quad (\text{bereik bij vier bits}) \quad (1.37)$$

Zoals gezegd, de representant $R_2^n(m)$ van m is altijd *niet-negatief* (1.34/1.35).

Decimaal	Two's Complement		Decimaal	Two's Complement	
m	$R_2^4(m)$	$R_2^4(m)_{\text{BIN}}$	m	$R_2^4(m)$	$R_2^4(m)_{\text{BIN}}$
0	0	0 0 0 0	-1	15	1 1 1 1
1	1	0 0 0 1	-2	14	1 1 1 0
2	2	0 0 1 0	-3	13	1 1 0 1
3	3	0 0 1 1	-4	12	1 1 0 0
4	4	0 1 0 0	-5	11	1 0 1 1
5	5	0 1 0 1	-6	10	1 0 1 0
6	6	0 1 1 0	-7	9	1 0 0 1
7	7	0 1 1 1	-8	8	1 0 0 0

Tabel 1.8. De two's complement representatie bij vier bits.

Uit tabel 1.8 en het voorafgaande volgt dat we op verschillende niveaus over de in het two's complement gerepresenteerde getallen m kunnen praten:

- $R_2^n(m)$
het niet-negatieve gehele getal, dat ontstaat door een getal m uit het bereik (1.33) te transformeren volgens (1.34) of (1.35);
- $R_2^n(m)_{\text{BIN}}$
de binaire voorstelling van het getal $R_2^n(m)$ met n bits in het binaire talstelsel;
- $R_2^n(m)_{\text{bit}}$
de logische voorstelling van het binair gecodeerde getal $R_2^n(m)_{\text{BIN}}$, wanneer we een logische schakeling ontwerpen waarin de bits logische nullen en enen voorstellen.

Op het hoogste niveau, dat van de getalvoorstelling $R_2^n(m)$, zullen we de eigenschappen van de two's complement representatie formuleren en in een later stadium ook afleiden. Het niveau daaronder, dat van $R_2^n(m)_{\text{BIN}}$, dient als tussenniveau om de afbeelding naar logische nullen en enen voor te bereiden en te vergemakkelijken. Op het laagste niveau, het logische

niveau, ontwerpen we de schakelingen die voor de uitvoering van het two's complement rekenen nodig zijn. Zoals gezegd, dit volgt later.

Voorbeeld

Gaan we uit van het getal $m = -3$, dan zijn bij een 4-bit representatie

$$R_2^4(-3) = 2^4 - 3 = 13 \quad (\text{decimaal gecodeerd geheel getal} \geq 0)$$

$$R_2^4(-3)_{\text{BIN}} = 1101_{\text{BIN}} \quad (\text{binair gecodeerd getal} \geq 0)$$

$$R_2^4(-3)_{\text{bit}} = 1101 \approx \text{TTFT} \quad (\text{bit string in logische 0/1 of in F/T}) \quad \square$$

Opmerking

Uit het bereik (1.33) en op grond van de transformatieregels (1.34) en (1.35) volgt dat in het two's complement het *tegengestelde* van het getal -2^{n-1} niet is voor te stellen. Zie tabel 1.8 voor het getal -8 . Het getal $+8$ kan dan niet worden voorgesteld. Deze schoonheidsfout van het two's complement komt voort uit praktische overwegingen bij het ontwerp van logische schakelingen, die op het two's complement gebaseerd zijn. Men zou het two's complement ook op een symmetrisch bereik kunnen definiëren,

$$-2^{n-1} + 1 \leq m \leq +2^{n-1} - 1 \quad (1.38)$$

Op praktische gronden is gekozen voor het bereik (1.33). $\square$

Het tekenbit

Uit tabel 1.8 zien we dat de binaire representanten van niet-negatieve gehele getallen als linkerbit altijd een 0 hebben en dat van de binaire representanten van negatieve gehele getallen het linkerbit altijd 1 is. Deze eigenschap geldt voor elk two's complement bereik volgens (1.33), voor willekeurige n . Uit de formules (1.34) en (1.35) volgt dat

$$0 \leq m \leq 2^{n-1} - 1 \leftrightarrow R_2^n(m) = m \leftrightarrow 0 \leq R_2^n(m) \leq 2^{n-1} - 1 \quad (1.39)$$

en

$$-2^{n-1} + 1 \leq m \leq -1 \leftrightarrow R_2^n(m) = 2^n + m \leftrightarrow 2^{n-1} \leq R_2^n(m) \leq 2^n - 1 \quad (1.40)$$

Schrijven we deze grenzen voor $R_2^n(m)$ binair, dan vinden we

$$00 \dots 00_{\text{BIN}} \leq R_2^n(m) \leq 01 \dots 11_{\text{BIN}} \quad (1.39.a)$$

en

$$10 \dots 00_{\text{BIN}} \leq R_2^n(m)_{\text{BIN}} \leq 11 \dots 11_{\text{BIN}} \quad (1.40.a)$$

Hiermee is aangetoond dat we aan het linkerbit van de binaire voorstelling $R_2^n(m)_{\text{BIN}}$ kunnen aflezen wat het teken van het oorspronkelijke getal m was. Men noemt dit linkerbit daarom wel het *tekenbit* van de two's complement representatie (sign bit). (Strikt genomen is dit onzin. In het two's complement domein (1.39/1.40) komen uitsluitend niet-negatieve

gehele getallen voor. Het begrip 'teken' slaat op het *oorspronkelijke domein* (1.33) van de getallen m .)

Alternatieve introductie two's complement

In de literatuur [Davio, 1983] wordt het two's complement ook wel geïntroduceerd als een binaire representatie van gehele getallen,

$$m \leftrightarrow a_{n-1}a_{n-2} \cdots a_1a_0$$

waarbij het voorste bit a_{n-1} het *negatieve gewicht* -2^{n-1} krijgt en de overige bits a_i voor $n-2 \geq i \geq 0$ een positief gewicht 2^i krijgen. Aldus is

$$m = -a_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i \cdot 2^i \quad (1.41)$$

Voorbeeld

$$-37 = -2^6 + 2^4 + 2^3 + 2^1 + 2^0$$

De hiermee overeenkomende bit string is

$$-37 \leftrightarrow 1011011$$

Uit definitie (1.35) volgt:

$$\begin{aligned} -37 &\leftrightarrow R_2^7(-37) = 2^7 - 37 \\ &= 91_{\text{DEC}} \\ &= 1011011_{\text{BIN}} \end{aligned}$$

Dit levert dezelfde bit string 1011011 op. □

De door ons gekozen representatie (1.34/35) levert voordelen op bij de bewijsvoering van de rekenregels voor het two's complement.

1.10 De optelling in het two's complement

Voor de optelling in het two's complement geldt de volgende stelling [Thijssen, 1990].

Stelling 1.1

$$R_2^n(m_1 + m_2) = R_2^n(m_1) + R_2^n(m_2) \pmod{2^n} \quad (1.42)$$

met

$$-2^{n-1} \leq m_1, m_2 \leq 2^{n-1} - 1 \quad \square$$

Volgens deze stelling wordt de representant van de som $m_1 + m_2$ van twee gehele getallen m_1 en m_2 , $R_2^n(m_1 + m_2)$, gevonden door de representanten $R_2^n(m_1)$ en $R_2^n(m_2)$ van m_1 en m_2 op te tellen. Is de som van de representanten,

$$R_2^n(m_1) + R_2^n(m_2)$$

vervolgens groter dan of gelijk aan 2^n , dan moet het resultaat met 2^n verminderd worden tot het resultaat ligt tussen

$$0 \leq R_2^n(m_1) + R_2^n(m_2) \leq 2^n - 1 \quad (1.43)$$

Wiskundig duidt men dit aan als de *sommatie modulo 2^n* .

Interpreteren we stelling 1.1 binair, dan volgt:

$$\begin{array}{rcl} R_2^n(m_1)_{\text{BIN}} & \rightarrow & a_{n-1}a_{n-2} \cdots a_1a_0 \\ R_2^n(m_2)_{\text{BIN}} + & \rightarrow & b_{n-1}b_{n-2} \cdots b_1b_0 + \\ \hline [R_2^n(m_1) + R_2^n(m_2)]_{\text{BIN}} & \rightarrow & \begin{array}{c} S_{n-1}S_{n-2} \cdots S_1S_0 \\ \downarrow \text{ modulo } 2^n \end{array} \\ & & S_{n-1}S_{n-2} \cdots S_1S_0 \\ R_2^n(m_1 + m_2)_{\text{BIN}} & \leftarrow & \end{array} \quad (1.44)$$

We vatten dit resultaat samen als stelling 1.2, de binaire versie van stelling 1.1.

Stelling 1.2

$$\begin{aligned} R_2^n(m_1 + m_2)_{\text{BIN}} &= R_2^n(m_1)_{\text{BIN}} + R_2^n(m_2)_{\text{BIN}} \pmod{2^n} \\ &= S_{n-1}S_{n-2} \cdots S_1S_0_{\text{BIN}} \end{aligned} \quad \square \quad (1.45)$$

Tellen we de binaire representanten op in een n -bit binaire opteller, dan moet het uitgaande carrybit C_n (ook wel het sombit S_n genoemd) worden weggelaten.

Voorbeelden

$$\begin{array}{rcl} R_2^4(+3)_{\text{BIN}} & \rightarrow & 0011 \\ R_2^4(+2)_{\text{BIN}} + & \rightarrow & 0010 + \\ \hline [R_2^4(+3) + R_2^4(+2)]_{\text{BIN}} & \rightarrow & \begin{array}{c} 0101 \\ \downarrow \text{ modulo } 2^4 \end{array} \\ & & R_2^4(+5)_{\text{BIN}} = 0101_{\text{BIN}} \end{array}$$

$$\begin{array}{rcl}
 R_2^4(-3)_{\text{BIN}} & \rightarrow & 1101 \\
 R_2^4(-2)_{\text{BIN}} + & \rightarrow & \underline{1110} + \\
 [R_2^4(-3) + R_2^4(-2)]_{\text{BIN}} & \rightarrow & \begin{array}{r} 1101 \\ + 1110 \\ \hline 11011 \end{array} \\
 & & \downarrow \text{mod ulo } 2^4 \\
 & & R_2^4(-5)_{\text{BIN}} = 1011_{\text{BIN}}
 \end{array}$$

$$\begin{array}{rcl}
 R_2^4(+3)_{\text{BIN}} & \rightarrow & 0011 \\
 R_2^4(-2)_{\text{BIN}} + & \rightarrow & \underline{1110} + \\
 [R_2^4(+3) + R_2^4(-2)]_{\text{BIN}} & \rightarrow & \begin{array}{r} 0011 \\ + 1110 \\ \hline 10001 \end{array} \\
 & & \downarrow \text{mod ulo } 2^4 \\
 & & R_2^4(+1)_{\text{BIN}} = 0001_{\text{BIN}}
 \end{array}$$

Uit deze voorbeelden blijkt dat de optelling van de binaire voorstelling van de two's complement representanten $R_2^n(m_1)$ en $R_2^n(m_2)$ met een normale binaire opteller kan worden uitgevoerd. Vergelijk dit eens met de in paragraaf 1.8 beschreven optelprocedure voor de teken-en-modulus representatie.

1.11 Optellen en aftrekken in het two's complement

Het verschil $m_1 - m_2$ van twee gehele getallen m_1 en m_2 kan bepaald worden door het tegengestelde van m_2 , te weten het getal $-m_2$, bij m_1 op te tellen:

$$m_1 - m_2 = m_1 + (-m_2) \quad (1.46)$$

Stelling 1.2 (formule 1.43) geldt ongeacht het teken van m_1 en m_2 , zodat we mogen stellen:

Stelling 1.3

$$\begin{aligned}
 R_2^n(m_1 - m_2) &= R_2^n(m_1 + (-m_2)) \\
 &= R_2^n(m_1) + R_2^n(-m_2) \pmod{2^n}
 \end{aligned} \quad \square (1.47)$$

Zijn van m_1 en m_2 de representanten $R_2^n(m_1)$ en $R_2^n(m_2)$ gegeven, dan moeten we om stelling 1.3 te kunnen toepassen eerst de representant van $-m_2$, $R_2^n(-m_2)$, uit de representant $R_2^n(m_2)$ van m_2 afleiden. Hiervoor bestaat een eenvoudige procedure. Deze is als volgt.

De representant van het tegengestelde getal

De *representant van het tegengestelde* van een getal m , het getal $-m$, kan in het two's complement verkregen worden door van de binaire voorstelling $R_2^n(m)_{\text{BIN}}$ van $R_2^n(m)$ alle bits te complementeren ($0 \rightarrow 1$ en $1 \rightarrow 0$) en er dan $+1$ bij op te tellen. Levert deze optelling een $(n+1)$ -bit som op, dan moet de voorste 1 met gewicht 2^n geschrapt worden (we rekenen in het two's complement modulo 2^n !).

Voorbeeld

$$\begin{array}{rcl}
 R_2^4(+5)_{\text{BIN}} = 0101_{\text{BIN}} & \rightarrow & 0101 \\
 & & \downarrow \text{complement} \\
 & & 1010 \\
 & & \underline{0001} + (+1 \text{ erbij}) \\
 R_2^4(-5)_{\text{BIN}} = 1011_{\text{BIN}} & \leftarrow & 1011
 \end{array}$$

Vergelijk ook tabel 1.8. □

Het recept om de representant $R_2^n(m)$ in die van het tegengestelde getal $R_2^n(-m)$ om te zetten werkt altijd, behalve bij de ondergrens van het bereik van het two's complement. Als gevolg van de asymmetrie in het bereik (1.33) kan $R_2^4(-8)$ niet in $R_2^4(+8)$ worden geconverteerd. (Dit kan wel als we op een 5-bit representatie overgaan!) Voor het getal 0 werkt het recept overigens correct.

Voorbeeld

$$\begin{array}{rcl}
 R_2^4(+0)_{\text{BIN}} = 0000_{\text{BIN}} & \rightarrow & 0000 \\
 & & \downarrow \text{complement} \\
 & & 1111 \\
 & & \underline{0001} + (+1 \text{ erbij}) \\
 & & 1'0000 \\
 & & \downarrow \text{modulo } 2^n \\
 R_2^4(-0)_{\text{BIN}} = 0000_{\text{BIN}} & \leftarrow & 0000
 \end{array}$$
□

Passen we het 'recept' op -8 toe, bij vier bits, dan gaat er iets fout. Ga dit na. Welk antwoord wordt gevonden? Past dit antwoord in de two's complement systematiek?

Bij de optelling van negatieve getallen kunnen we in het two's complement handig gebruik maken van stelling 1.4.

Stelling 1.4

$$R_2^n(-m)_{\text{BIN}} = C_{R_2^n(m)_{\text{BIN}}} + 1 \pmod{2^n} \quad \square \quad (1.48)$$

Hiermee hebben we aangetoond hoe in het two's complement de aftrekking kan worden teruggebracht tot de optelling van de representant van het tegengestelde getal.

1.12 Capaciteitsoverschrijding in het two's complement

De som $m_1 + m_2$ van twee gehele getallen m_1 en m_2 uit het bereik (1.33) van het two's complement kan buiten het toegestane bereik vallen. Tellen we bijvoorbeeld +3 en +6 op, dan moet de som +9 zijn. In een 4-bit two's complement bereik (zie tabel 1.8) kunnen de getallen +3 en +6 wel en hun som +9 niet worden voorgesteld. Bij de optelling van +3 en +6 treedt *capaciteitsoverschrijding* op. Passen we bij capaciteitsoverschrijding toch de procedure uit paragraaf 1.10 toe, dan is het antwoord fout.

Voorbeeld

$$\begin{array}{rcl}
 R_2^4(+3)_{\text{BIN}} & \rightarrow & 0011 \\
 R_2^4(+6)_{\text{BIN}} & \rightarrow & 0110+ \\
 \hline
 [R_2^4(+3) + R_2^4(+6)]_{\text{BIN}} & \rightarrow & 01001 \\
 & & \downarrow \text{ modulo } 2^4 \\
 & & R_2^4(-7)_{\text{BIN}} = 1001_{\text{BIN}}
 \end{array}$$

Het gevonden antwoord is $R_2^4(-7)_{\text{BIN}}$, de representant van een negatief getal, terwijl de representant van een positief getal verwacht werd. $\square$

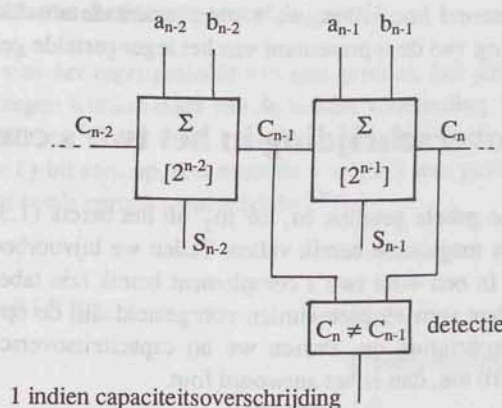
Bij dit voorbeeld kunnen we twee aantekeningen maken. Het verschil is 16. Bij vier bits rekenen we 'modulo 2^4 ' en dit is precies wat we kwijtgeraakt zijn. Er zit dus duidelijk enige systematiek achter. Het is hier echter niet de plaats om dieper op deze problematiek in te gaan. We noemen slechts de stelling dat het optreden van capaciteitsoverschrijding kan worden gedetecteerd aan:

Stelling 1.5

$$C_n \neq C_{n-1} \leftrightarrow \text{capaciteitsoverschrijding} \quad \square (1.49)$$

Onderzoeken we dit aan het vorige voorbeeld, dan is $C_3 = 1$ en $C_4 = 0$. De stelling klopt. Er is sprake van capaciteitsoverschrijding.

Figuur 1.8 geeft een toelichting op de plaats waar we C_n en C_{n-1} in een binaire opteller moeten aftakken. Met deze schakeling kan op een heel eenvoudige wijze een detectie van capaciteitsoverschrijding uitgevoerd worden. Rond alle ALU's in computers en processoren zit deze schakeling. De detectie-uitgang zit er standaard op, in de vorm van een overflow detectie. Deze detectie speelt een essentiële rol bij het uitvoeren van rekenkundige bewerkingen in software.



Figuur 1.8. Detectie capaciteitsoverschrijding.

Voorbeeld

$$\begin{array}{rcl}
 -3 & R_2^4(-3)_{\text{BIN}} & \rightarrow 1101 \\
 -6 & R_2^4(-6)_{\text{BIN}} & \rightarrow 1010 \\
 \hline
 -9 & [R_2^4(-3) + R_2^4(-6)]_{\text{BIN}} & \rightarrow \begin{array}{r} 10111 \\ \text{carry} \end{array} \\
 & & \downarrow \text{modulo } 2^n
 \end{array}$$

$$R_2^4(+7)_{\text{BIN}} = 0111_{\text{BIN}}$$

In dit voorbeeld is bij de optelling $C_3 = 0$ en $C_4 = 1 \Rightarrow$ capaciteitsoverschrijding. $\square$

In een optelschakeling kan capaciteitsoverschrijding gedetecteerd worden middels een test op de laatste (= uitgaande) carry en de voorlaatste carry. Dit kan met een EXOR poort. Zie ook paragraaf 2.2.

Als regel geldt dat, als er bij de optelling van twee representanten met gelijk tekenbit een representant met een daaraan tegengesteld tekenbit ontstaat, er capaciteitsoverschrijding is opgetreden. Ter completering merken we nog op dat bij de optelling van de representanten van getallen met verschillend teken nooit capaciteitsoverschrijding optreedt. Voor de bewijzen van het bovenstaande verwijzen we naar [Thijssen, 1990].

Opmerking

Capaciteitsoverschrijding kan worden voorkomen door van te voren naar het mogelijke bereik van de som te kijken en hieraan het 'aantal bits n ' van de two's complement representanten aan te passen. $\square$

Samenvatting van het two's complement

In het two's complement rekenen we 'modulo 2^n ' waarbij we, als een som groter dan $2^n - 1$ of kleiner dan 0 is, zoveel keer 2^n aftrekken of bijtellen totdat het resultaat tussen 0 en $2^n - 1$ ligt (de grenzen inbegrepen).

Het two's complement kan formeel beschreven worden als een (binair) rekensysteem, dat gedefinieerd is op de *verzameling restklassen modulo 2^n* . Een restklasse modulo 2^n is de verzameling van alle gehele getallen, die op een veelvoud van 2^n na aan elkaar gelijk zijn. De restklasse $[m]$ is:

$$[m] = \{g \mid g \in \mathbb{Z} \text{ en } g = m + k \cdot 2^n\} \quad \text{met } k \in \mathbb{Z} \quad (1.50)$$

In het two's complement wordt elke restklasse voorgesteld door de representant $R_2^n(m)$ ervan. Dit is het getal uit de restklasse dat ligt tussen

$$0 \leq \text{representant } R_2^n(m) \leq 2^n - 1 \quad (1.51)$$

In het two's complement (modulo 2^n) wordt er geen onderscheid gemaakt tussen het getal m en $m + k \cdot 2^n$. Voor onze toepassing van dit systeem op het binaire rekenen is het lastig dat er tijdens het rekenen telkens termen met gewicht 2^n opduiken of verdwijnen. Wanneer dit optreedt kan dat gedetecteerd worden aan *capaciteitsoverschrijding*. We hebben gezien dat capaciteitsoverschrijding gemakkelijk gedetecteerd kan worden in een schakeling. Dat maakt het mogelijk op een voor onze toepassingen zinvolle wijze om te gaan met het two's complement rekenen.

De two's complement cirkel

Optellen en aftrekken in het two's complement wordt vaak voorgesteld in een two's complement cirkel. De 4-bit versie hiervan staat in figuur 1.9. Bij optellen of aftrekken van een getal gaat men het benodigde aantal stappen in de gewenste richting langs de cirkel.

1.13 De one's complement representatie

In het two's complement worden de getallen uit het domein (1.33),

$$-2^{n-1} \leq m \leq 2^{n-1} - 1$$

door middel van de transformaties (1.34) en (1.35) naar het domein (1.39/40) getransformeerd:

$$0 \leq R_2^n(m) \leq 2^n - 1$$

In dit nieuwe domein verlopen de berekeningen optellen en aftrekken, wanneer we kijken naar de bijbehorende logische schakelingen, veel gemakkelijker dan bij de teken-en-modulus representatie. Reden om deze transformatie toe te passen. In het nieuwe domein wordt modulo 2^n gerekend, alles wat tijdens een bewerking buiten het domein (1.39/40)

valt wordt door aftrekking van een of meer keer 2^n binnen het domein gebracht. In het verleden is ook de *one's complement representatie* veel toegepast. Bij deze representatie worden de getallen uit het domein

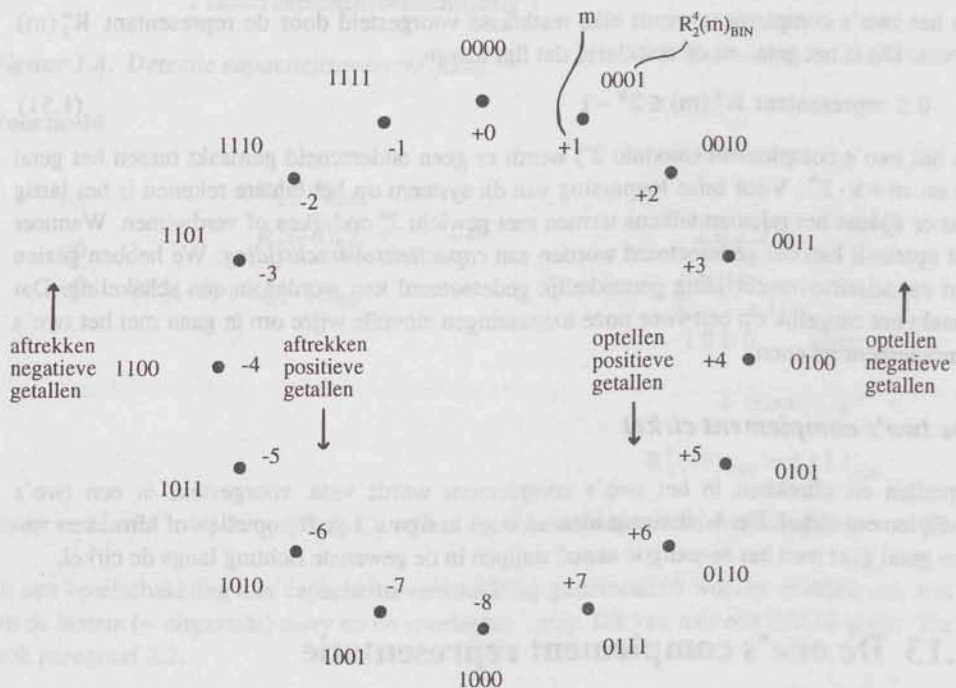
$$-2^{n-1} + 1 \leq m \leq 2^{n-1} - 1 \quad (\text{bereik one's complement}) \quad (1.52)$$

getransformeerd volgens

$$0 \leq m \leq 2^{n-1} - 1 \rightarrow R_1^n(m) = m \quad (1.53)$$

$$-2^{n-1} + 1 \leq m \leq 0 \rightarrow R_1^n(m) = 2^n - 1 + m \quad (1.54)$$

(In $R_1^n(m)$ duidt het subscript 1 op het one's complement.)



Figuur 1.9. De two's complement cirkel.

Het one's complement kent volgens de definitie (1.53/1.54) een *dubbele representant* van 0, $R_1^n(0)_{\text{BIN}} = 00 \dots 00_{\text{BIN}}$ en $R_1^n(0)_{\text{BIN}} = 11 \dots 11_{\text{BIN}}$. De optelling geschiedt volgens stelling 1.6.

Stelling 1.6

$$R_1^n(m_1 + m_2) = R_1^n(m_1) + R_1^n(m_2) \pmod{2^n - 1} \quad \square (1.55)$$

Voorbeeld

$$\begin{array}{rcll}
 -3 & R_1^4(-3)_{\text{BIN}} = 1100_{\text{BIN}} & \rightarrow & 1100 \\
 -2 + & R_1^4(-2)_{\text{BIN}} = 1101_{\text{BIN}} & + & 1101 \\
 -5 & [R_1^4(-3) + R_1^4(-2)]_{\text{BIN}} & \rightarrow & \begin{array}{r} 11001 \\ \text{---} \rightarrow 1+ \end{array} \\
 & R_1^4(-5)_{\text{BIN}} = 1010_{\text{BIN}} & \leftarrow & 1010
 \end{array}$$

Nu mogen we bij het optreden van $S_n = 1$ in de optelling geen term 2^n weglaten, maar slechts $2^n - 1$. In de binaire uitvoering betekent dit dat, als een uitgaande carry/sombit C_n/S_n wordt weggelaten op de hoogstwaardige positie, er ter compensatie op de laagstwaardige positie een 1 moeten worden bijgeteld. De carry C_n heeft dan het gewicht 2^0 . $\square$

Mede door deze extra optelling is het one's complement op de achtergrond geraakt. Voor een uitgebreidere behandeling van het one's complement zie [Thijssen, 1990].

1.14 De floating-point representatie

De tot nu toe behandelde getalrepresentaties waren *fixed-point representaties*. De komma (point) stond op een vaste plaats en wel meestal (impliciet) rechts van de cijfers of bits. Bij fracties was de plaats van de komma aan de linkerzijde (paragraaf 1.1).

Wanneer de komma op een vaste plaats staat zijn de bewerkingen onafhankelijk van de plaats van de komma. De plaats van de komma in het resultaat hangt af van het type bewerkingen (optellen ~ komma op dezelfde plaats en vermenigvuldigen ~ komma schuift aantal posities op) en niet van de waarde van de cijfers of bits.

Bij heel grote of heel kleine getallen kiest men voor een *drijvende komma* of *floating-point* representatie. Deze representatie kent een *mantisse* (*mantissa*) of *fractie* en een *exponent* (*exponent*). Een getal m wordt dan voorgesteld door

$$m = \pm \text{mantisse} \cdot \text{grondtal}^{\text{exponent}}$$

Voorbeeld

$$\begin{aligned}
 3.160.000.000 & \quad (\text{fixed-point}) \\
 = 316 \cdot 10^7 & \quad (\text{floating-point}) \\
 = 3,16 \cdot 10^9 & \quad (\text{floating-point}) \\
 = 0,316 \cdot 10^{10} & \quad (\text{floating-point})
 \end{aligned}$$

 $\square$

De *floating-point* representatie maakt het mogelijk in de getalrepresentatie het bereik te scheiden van de nauwkeurigheid. Zo is in het voorbeeld hierboven het getalbereik ca 10^{10} en de nauwkeurigheid slechts drie decimalen. Het nadeel is dat de *floating-point*

representatie niet uniek is. Een getal kan op meer dan één manier voorgesteld worden. Dit kan in de praktijk tot veel conversies leiden. Optellen van twee getallen kan bijvoorbeeld alleen wanneer we de exponenten eerst gelijk maken.

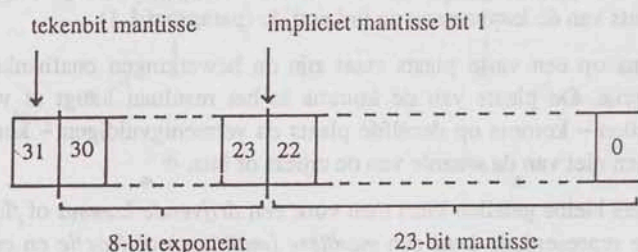
Voorbeeld

$$\begin{aligned} 0,3180 \cdot 10^{10} + 0,3101 \cdot 10^7 &= 318 \cdot 10^7 + 0,3101 \cdot 10^7 \\ &= 318,3101 \cdot 10^7 \\ &= 0,3183 \cdot 10^{10} \end{aligned}$$

Bij vier cijfers voor het bereik van de mantisse gaat iets van de nauwkeurigheid verloren. □

Bij gegeven aantal cijfers/bits per getal moet er een keuze gemaakt worden tussen het aantal cijfers voor de mantisse (meer ~ nauwkeuriger) en de exponent (meer ~ groter bereik). Het zal duidelijk zijn dat als verschillende fabrikanten van computers tot een verschillende keuze komen, er veel conversies nodig zijn om software van de ene computer over te zetten op de andere computer. Daarom zijn hierover internationaal afspraken gemaakt in de vorm van de ANSI/IEEE floating-point standaard [ANSI/IEEE, 1985]. Hierin staan verschillende getalformaten gespecificeerd. De ANSI/IEEE kent een 32-bit *single format* en een 64-bit *double format*. Beide formats kunnen uitgebreid worden tot een *extended format*. Als voorbeeld beschrijven we het 32-bit single format.

ANSI/IEEE 32-bit single format



Figuur 1.10. 32-Bit floating-point format.

Figuur 1.10 specificeert het 32-bit format. Het linkerbit (no. 31) is het tekenbit van de mantisse. Hiermee wordt het positief (0) of negatief (1) van het getal aangegeven.

De volgende acht bits zijn voor de exponent. Deze kan ook negatief zijn en zou dus een tekenbit moeten bevatten. In de standaard heeft men dit opgevangen door bij de exponent een vaste verschuiving (bias) van +127 op te tellen. Alle exponenten zijn dus 127 te groot. Aldus is geen tekenbit nodig.

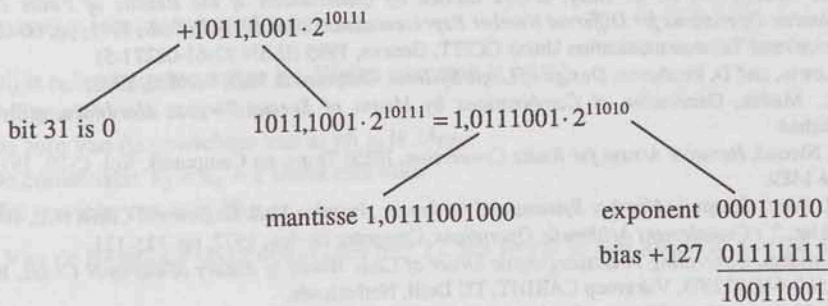
De rechter 23 bits stellen de mantisse voor. De binaire voorstelling van de mantisse wordt zover genormaliseerd totdat het linkerbit 1 is, dus tot

$$1, a_{-1}a_{-2}a_{-3}\dots$$

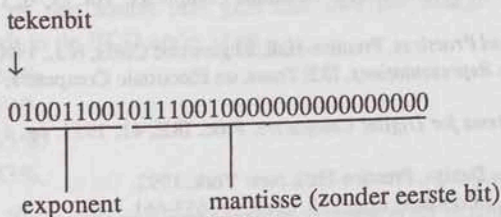
Dit linkerbit voor de komma wordt dan weggelaten in de 32-bit representatie. Zo is met 23 bits een 24-bit nauwkeurigheid mogelijk. (Ga na dat als gevolg van deze beslissing het getal 'nul' problemen geeft.)

Voorbeeld

Het binaire getal $+1011,1001 \cdot 2^{1011}$ is in floating-point format volgens figuur 1.10:



Standaard format:



De ANSI/IEEE standaard bevat naast de formats nog details ten aanzien van het voorstellen van nul en voor foutafhandeling. Het voert te ver hier op deze details in te gaan. Zie bijvoorbeeld [Wilkinson, 1992; Hill, 1978].

Hiermee besluiten we voorlopig het onderwerp getalrepresentaties en het binair rekenen. In vervolghoofdstukken komen de details van de bijbehorende schakelingen aan bod.

Literatuur

1. ANSI/IEEE Standard for Binary Floating-Point Arithmetic, Std. 754-1985, New York: IEEE 1985.
2. A. Avizienis, *Signed-Digit Number Representations for Fast Parallel Arithmetic*, IRE Trans. on Electronic Computers, Vol EC-10, 1961, pp. 389-400.
3. C. Boyer, *A History of Mathematics*, John Wiley, New York, 1968.
4. J. F. Couleur, *BIDEC - A Binary-to-Decimal or Decimal-to-Binary Converter*, IRE Trans. on Electronic Computers, Vol. EC-7, 1958, pp. 313-316.
5. M. Davio, J-P. Deschamps, and A. Thayse, *Digital Systems with Algorithm Implementation*, John Wiley, New York, 1983.
6. H. C. A. van Duuren, *Typendruk Telegrafie over Radioverbindingen*, Het PTT Bedrijf, Jaarg. 1, no. 2, 1947, pp. 35-41.
7. F. A. Foss, *The Use of a Reflected Code in Digital Control Systems*, IRE Trans. on Electronic Computers, Vol. EC-3, 1954, pp. 1-6.

8. E. L. Gilbert, *How Good is Morse Code*, Information & Control, Vol. 14, 1969, pp. 559-565.
9. P. E. Gosling, and Q. L. M. van Laarhoven, *Codes for Computers and Microprocessors*, Macmillan, London, 1980.
10. F. Gray, *Pulse Code Communication*, Patent 2,632,058, March 17, 1953.
11. H. J. Gray, Jr, P. V. Levanian, and M. Rubinooff, *An Analog-to-Digital Converter for Serial Computing Machines*, Proc. IRE, Oct. 1953, pp. 1462-1465.
12. F. J. Hill and G. R. Peterson, *Digital Systems, Hardware Organization and Design*, John Wiley, New York, 1978, Ch. 16.
13. T. C. Huang, and D. C. Huey, *Errors Caused by Quantization of the Results of Finite Length Arithmetic Operations for Different Number Representations*, Proc. IEEE ISCAS, 1975, pp. 60-65.
14. International Telecommunication Union CCITT, Geneva, 1985 (ISBN 92-61-02271-5)
15. D. Lewin, and D. Protheroe, *Design of Logic Systems*, Chapman & Hall, London, 1992.
16. F. L. Muller, *Generation of Combinations by Means of Special-Purpose Hardware*, still to be published.
17. J-D. Nicoud, *Iterative Arrays for Radix Conversion*, IEEE Trans. on Computers, Vol. C-20, 1971, pp. 1479-1489.
18. N. R. Scott, *Computer Number Systems and Arithmetic*, Prentice-Hall, Englewood Cliffs, N.J., 1985.
19. S. Sklar, *2's Complement Arithmetic Operations*, Computer Design, 1972, pp. 115-121.
20. A. Snijders, *Generating in Lexicographic Order of Code Words of Binary M-out-of-N Codes*, Report 051 560-44(1972)03, Vakgroep CARDIT, TU Delft, Netherlands.
21. A. P. Thijssen, H. A. Vink en C. H. Eversdijk, *Digitale Techniek 2*, Delftse Uitgevers Maatschappij, Delft, 1990 of Delft University Press, Delft, 1995.
22. L. B. Wadel, *Negative Base Number Systems*, IRE Trans. on Electronic Computers, Vol. EC-6, 1957, pp. 123.
23. J. F. Wakerly, *Digital Design Principles and Practices*, Prentice-Hall, Englewood Cliffs, N.J., 1990.
24. G. P. Weeg, *Uniqueness of Weighted Code Representations*, IRE Trans. on Electronic Computers, Vol. EC-9, 1960, pp. 487-489.
25. G. S. White, *Coded Decimal Number Systems for Digital Computers*, Proc. IRE, 41, 1953, pp. 1450-1452.
26. B. Wilkinson and R. Makki, *Digital System Design*, Prentice-Hall, New York, 1992.
27. L. J. Zeckendorf, *Character Sets en Codes*, Informatie, Jaarg. 15, 1973, pp. 653-661.
28. H. Zemanek, *Alphabets and Codes*, Elektronische Rechenanlagen, Vol. 7, 1965, pp. 239-258.

Opgaven hoofdstuk 1

1.1. Over talstelsels worden vier beweringen gedaan. Een ervan is fout. Welke?

- a. In een talstelsel met positief geheel grondtal g lopen de cijfers van 0 t/m $g-1$.
- b. Natuurlijke getallen m zijn op één manier voor te stellen in een willekeurig talstelsel met positief geheel grondtal $g \geq 2$.
- c. Het gewicht van het n^e cijfer is g^{n-1} .

x x x x

↑ vierde cijfer

- d. Bij twee talstelsels met grondtal g_1 en g_2 , met $g_1 < g_2$, verdient het talstelsel met het grondtal dat het dichtst bij 2 of een macht van 2 ligt de voorkeur.

1.2. Fracties kunnen in een talstelsel met grondtal g voorgesteld worden door

$$0, a_{-1}a_{-2} \cdots a_{-n+1}a_{-n} = \sum_{i=-1}^{-n} a_i \cdot g^i$$

In een binaire getalrepresentatie, met 8 bits achter de komma, kunnen niet alle fracties van een getal voorgesteld worden. Beelden we een fractie af op de dichtstbijzijnde binaire

waarde, dan maken we hierbij een fout. De maximale (absolute) waarde van deze fout is

- a. 2^{-7} .
- b. 2^{-8} .
- c. 2^{-9} .
- d. 2^{-10} .

1.3. Gegeven is een BCD gecodeerd getal van 12 bits:

$$m = a_{11}a_{10}a_9a_8 a_7a_6a_5a_4 a_3a_2a_1a_0 \text{ BCD}$$

Hierbij is a_0 het laagstwaardige bit. Welke uitspraak is juist?

- a. Het gewicht van a_6 is 40_{DEC} .
- b. De som van de gewichten van a_3 en a_4 is 24_{DEC} .
- c. De combinatie $a_3 = a_0 = 1$ komt niet voor.
- d. Het gewicht van a_3 is 10_{DEC} .

1.4. Van de mogelijke codecombinaties met 12 bits wordt er een aantal in een 12-bit BCD code

$$m = a_{11}a_{10}a_9a_8 a_7a_6a_5a_4 a_3a_2a_1a_0 \text{ BCD}$$

wel en een aantal niet gebruikt. Het percentage van alle codecombinaties dat gebruikt wordt in de BCD code is ca.

- a. 24%.
- b. 39%.
- c. 63%.
- d. 83%.

1.5. Twee niet-negatieve gehele getallen A en B zijn in de hexadecimale code gecodeerd als

$$A = 3789$$

$$B = 8917$$

De som van A en B, $S = A + B$, is in de hexadecimale code:

- a. C1A1.
- b. C0A0.
- c. 12706.
- d. anders dan onder a t/m c is aangegeven.

1.6. De representaties van de octale en de hexadecimale getallen worden aangeduid met N_{OCT} en N_{HEX} . Wanneer N_{OCT} vier cijferposities bezit (bijvoorbeeld 1723) en N_{HEX} drie cijferposities bezit (bijvoorbeeld 861), dan geldt voor hun bereik:

- a. Bereik $N_{\text{OCT}} = 1 \frac{1}{3}$ bereik N_{HEX} .
- b. Bereik $N_{\text{OCT}} = \text{bereik } N_{\text{HEX}}$.
- c. Bereik $N_{\text{OCT}} = \frac{1}{2}$ bereik N_{HEX} .
- d. Geen van deze antwoorden is juist.

1.7. Gegeven zijn niet-negatieve gehele getallen met n decimale cijfers. Om deze getallen binair te kunnen representeren zijn tenminste m bits nodig. Dan is m:

- a. $4n$.

- b. $10n$.
- c. $\lceil 3,322n \rceil$. (haken duiden het eerstvolgende gehele getal aan)
- d. Tussen n en m bestaat geen vast verband.

1.8. Het getal 1011011010_{BIN} in de gewone binaire code is

- a. een tweevoud, maar geen vijfvoud.
- b. geen tweevoud, maar wel een vijfvoud.
- c. noch een tweevoud, noch een vijfvoud.
- d. zowel een tweevoud als een vijfvoud.

1.9. De BCD representatie van 11100011_{BIN} is

- a. 0010 0010 0111.
- b. 1110 0011.
- c. 343.
- d. E3.

1.10. Gegeven is een getal m in de hexadecimale code:

$$m = 372_{\text{HEX}}$$

In de BCD code wordt m genoteerd als:

- a. 0010 0101 0000.
- b. 1000 1000 0010.
- c. 0011 0111 0010.
- d. 882.

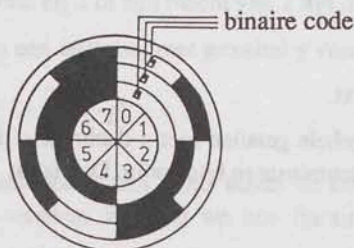
1.11. Bij transmissie van informatie wordt bij de zender soms een parity bit aan de data toegevoegd. Dit heeft tot doel het verkrijgen of verbeteren van de mogelijkheid om bij de ontvanger fouten te detecteren in de ontvangen data. Bij hoeveel van de volgende codes

- 5-bit telegraafcode (32 combinaties gebruikt)
- 3-uit-7 code (drie enen in elk woord van zeven bits)
- BCD code
- 7-bit ASCII code

is (met genoemd doel) het toevoegen van een parity bit nuttig?

- a. Eén.
- b. Twee.
- c. Drie.
- d. Vier.

1.12. Gegeven is een binaire codeschijf.



Zoals bekend treden er bij het uitlezen van deze codeschijf overgangsverschuiven op. Men probeert deze overgangsverschuiven op te lossen door, uitgaande van de huidige stand, uitsluitend de twee naastliggende standen te accepteren en de overige standen te negeren. Alle andere tussenstanden worden dan niet geaccepteerd en zo nodig wordt de informatie over de vorige stand gehandhaafd. Welke van de volgende beweringen is van toepassing?

- Er kunnen nog steeds uitleesfouten voorkomen.
- Behalve bij wisseling van draairichting zijn alle uitleesfouten geëlimineerd.
- Bij vaste draairichting komen geen uitleesfouten voor.
- Alle uitleesfouten zijn vermeden.

1.13. Over de gespiegeld binaire code worden drie beweringen gedaan.

- Een gespiegeld binaire code voor het coderen van 2^n opeenvolgende objecten kost in het algemeen meer bits dan de binaire code. Dit komt omdat bij de gespiegeld binaire code telkens maar één bit mag veranderen.
- Bij de gespiegeld binaire code verhouden de gewichten van de bits (van rechts naar links) zich als 1 : 2 : 4 : 8, enz.
- De gespiegeld binaire code is een typische 'rekencode', dat wil zeggen dat deze code veel wordt toegepast bij het uitvoeren van rekenkundige bewerkingen.

Van deze drie beweringen is/zijn er

- geen juist.
- één juist.
- twee juist.
- drie juist.

1.14. Twee getallen A en B zijn in de binaire teken-en-modulus representatie weergegeven als

$$A \leftrightarrow a_{n-1}a_{n-2} \dots a_1a_0$$

$$B \leftrightarrow b_{n-1}b_{n-2} \dots b_1b_0$$

Hierbij zijn a_{n-1} en b_{n-1} de tekenbits.

Over de schakeling om deze twee getallen op te tellen worden vier beweringen gedaan, waarvan er één juist is. Welke?

- Voor de optelling van A en B is een binaire opteller nodig van minimaal n secties.
- Voor de optelling van A en B is een binaire opteller/aftrekker nodig van minimaal n + 1 secties.
- Er treedt capaciteitsoverschrijding op als $C_n = 1$ is. (C_n is het transport uit de sectie die twee bits met gewicht 2^{n-1} optelt, alsmede C_{n-1} .)
- Geen van de vorige drie beweringen is (geheel) juist.

1.15. Vier getallen, welke zijn gegeven in de 16-bit teken-en-modulus representatie, moeten worden opgeteld. Om dan geen last te hebben van capaciteitsoverschrijding moet de representatie worden uitgebreid naar een (16+k)-bit representatie. Dan is

- k = 1.

- b. $k = 2$.
- c. $k = 3$.
- d. $k = 4$.

1.16. Het bereik van de gehele getallen m , welke met n bits in het two's complement kunnen worden voorgesteld, is

- a. $-2^n \leq m \leq 2^n - 1$.
- b. $-2^n - 1 \leq m \leq 2^n - 1$.
- c. $-2^n + 1 \leq m \leq 2^n - 1$.
- d. $-2^{n-1} \leq m \leq 2^{n-1} - 1$.

1.17. Om het decimale getal $+32_{\text{DEC}}$ in het two's complement voor te stellen zijn

- a. zes bits nodig.
- b. vijf bits en een apart tekenbit nodig.
- c. zeven bits nodig.
- d. acht bits nodig.

1.18. In het two's complement wordt het gehele getal m afgebeeld op $R_2^n(m)$.

Bij gebruik van acht bits wordt $m = -41$ afgebeeld op

- a. $R_2^8(m) = 215$.
- b. $R_2^8(m) = 214$.
- c. $R_2^8(m) = 185$.
- d. $R_2^8(m)$ is een ander getal.

1.19. Voor het getal -93 bevat de two's complement representant $R_2^8(-93)_{\text{BIN}}$ in totaal

- a. drie enen.
- b. vier enen.
- c. vijf enen.
- d. meer dan vijf enen.

1.20. In het two's complement wordt het gehele getal m afgebeeld op $R_2^n(m)$.

- a. Voor positieve m is $R_2^n(m)$ onafhankelijk van n .
- b. Voor negatieve m is $R_2^n(m)$ onafhankelijk van n .
- c. Zowel voor positieve als negatieve m is $R_2^n(m)$ onafhankelijk van n .
- d. Zowel voor positieve als negatieve m is $R_2^n(m)$ afhankelijk van n .

1.21. Van een aantal getallen is de two's complement representant in zes bits gegeven.

$$R_2^6(+6)_{\text{BIN}} = 001100_{\text{BIN}}$$

$$R_2^6(-6)_{\text{BIN}} = 111010_{\text{BIN}}$$

$$R_2^6(-0)_{\text{BIN}} = 111111_{\text{BIN}}$$

$$R_2^6(+16)_{\text{BIN}} = 100000_{\text{BIN}}$$

$$R_2^6(-11)_{\text{BIN}} = 110101_{\text{BIN}}$$

Hervan is/zijn er

- a. één fout.
- b. twee fout.
- c. drie fout.
- d. vier fout.

1.22. Van een in het two's complement gerepresenteerd geheel getal m is van $R_2^8(m)_{\text{BIN}}$ het linker bit $a_7 = 1$. Dan ligt het getal m tussen (grenzen zo nauwkeurig mogelijk):

- a. $-2^7 \leq m \leq 2^7$.
- b. $-2^7 \leq m \leq -2^0$.
- c. $-2^8 \leq m \leq 0$.
- d. andere grenzen.

1.23. Van twee gehele getallen is $m_1 = -23$ en $m_2 = +7$. In het two's complement worden deze getallen voorgesteld door een bit string of door de representant ervan (dat is het met de bit string corresponderende niet-negatieve gehele getal volgens de gewichten van de binaire code). De representant van de som van $-m_1$ en $-m_2$ is:

- a. $R_2^6((-m_1) + (-m_2)) = +48$.
- b. $R_2^6((-m_1) + (-m_2)) = -16$.
- c. $R_2^6((-m_1) + (-m_2)) = +16$.
- d. $R_2^6((-m_1) + (-m_2))$ heeft een andere dan de genoemde waarden.

1.24. Aan een 8-bit binaire opteller worden de getallen $+13$ en -34 in het two's complement aangeboden. Het resultaat $S_7S_6S_5S_4S_3S_2S_1S_0$ van deze optelling is:

- a. 1 1 1 0 1 0 1 1.
- b. 1 1 1 0 1 1 0 0.
- c. 1 1 1 0 1 0 1 0.
- d. 1 0 0 1 0 1 0 1.

1.25. Gegeven zijn de getallen $m_1 = +23$ en $m_2 = -56$. Tel hun binaire representanten op in het two's complement, met gebruikmaking van 8-bit representanten. Dan geldt voor het resultaat:

- a. $R_2^8(m_1 + m_2)_{\text{BIN}} = 01011110_{\text{BIN}}$ en $C_8 = 0$ (C_8 is de uitgaande carry).
- b. $R_2^8(m_1 + m_2)_{\text{BIN}} = 11011110_{\text{BIN}}$ en $C_8 = 1$.
- c. $R_2^8(m_1 + m_2)_{\text{BIN}} = 11011111_{\text{BIN}}$ en $C_8 = 0$.
- d. $R_2^8(m_1 + m_2)_{\text{BIN}} = 11011111_{\text{BIN}}$ en $C_8 = 1$.

1.26. In het two's complement wordt de representant $R_2^n(-m)_{\text{BIN}}$ van het tegengestelde van een getal m uit de representant $R_2^n(+m)_{\text{BIN}}$ verkregen door

- a. het tekenbit te inverteren.
- b. alle bits te inverteren.
- c. naar keuze eerst 1 bij het getal op te tellen en daarna alle bits te inverteren of als alternatief eerst alle bits te inverteren en er dan 1 bij op te tellen.
- d. Geen van deze drie antwoorden is goed.

1.27. Bij het optellen van twee negatieve gehele getallen m_1 en m_2 in het two's complement treedt capaciteitsoverschrijding op als voor hun som $m_1 + m_2$ geldt:

- a. $-2^n \leq m_1 + m_2 \leq -2^{n-1} - 1$.
- b. $-2^n + 2 \leq m_1 + m_2 \leq -2^{n-1} - 1$.
- c. $-2^n \leq m_1 + m_2 \leq -2^{n-1}$.
- d. $-2^n + 2 \leq m_1 + m_2 \leq -2^{n-1} - 1$.

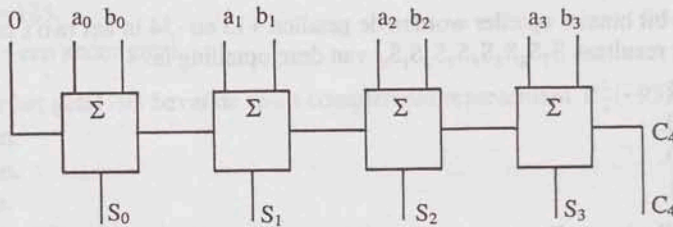
1.28. Aan een 4-bit opteller worden de getallen -4 en -5 in het two's complement aangeboden. Het resultaat $S_3S_2S_1S_0$ van de optelling is

- a. 0 1 1 0.
- b. 1 0 0 1.
- c. 1 0 0 0.
- d. 0 1 1 1.

1.29. Aan een 4-bit binaire opteller worden de binaire representanten $a_3a_2a_1a_0$ en $b_3b_2b_1b_0$ van twee getallen in het two's complement aangeboden. Indien $C_4 = 1$ blijkt te zijn, dan is het resultaat (denk aan capaciteitsoverschrijding) $S_3S_2S_1S_0$ goed mits

- a. C_4 wordt verwaarloosd.
- b. C_4 wordt verwaarloosd en er 1 bij het resultaat wordt opgeteld.
- c. C_4 wordt verwaarloosd en $C_3 = 1$ is.
- d. C_4 wordt verwaarloosd en $C_3 = 0$ is.

1.30.



Gegeven is een 4-bit binaire opteller, die gebruikt wordt voor het optellen van twee getallen $a_3a_2a_1a_0$ en $b_3b_2b_1b_0$. Deze getallen, gerepresenteerd in een complement code, worden aangeboden aan de gegeven opteller. Bij het verkregen resultaat moet nog 1 opgeteld worden bij toepassing van

- a. het one's complement steeds.
- b. het one's complement mits $C_4 = 1$ is.
- c. het two's complement steeds.
- d. het two's complement mits $C_4 = 1$ is.

Stelling 1

Zoals het klokje thuis telt, telt het nergens.

2

De schakelalgebra

2.1 De logische werking van een component

In de digitale techniek gaat men uit van twee signaalniveaus. Dit besluit vereist tweewaardige componenten. Ook moeten we de interface van een digitale schakeling met de buitenwereld baseren op tweewaardige signalen, althans aan de digitale zijde ervan. De keuze van de digitale grootheden of signalen ligt daarbij helaas niet altijd zo duidelijk voor de hand als bij het rekenen in het binaire talstelsel. In dit hoofdstuk ontwikkelen we wiskundig gereedschap om de afbeelding naar tweewaardig werkende componenten te vergemakkelijken. Dit gereedschap, de schakelalgebra, gebruiken we vervolgens om het ontwerp van logische schakelingen te optimaliseren.

Bij een mechanische schakelaar (kontakt) is er sprake van een *aktie* of *stimulus*, een kracht of een magnetisch veld, waarmee invloed op het kontakt wordt uitgeoefend. De uitgeoefende kracht beïnvloedt de *toestand* (stand) waarin het kontakt verkeert, het kontakt is open of gesloten. Dit is een *waarneembare eigenschap* van het kontakt en wel via het geleidingsvermogen ervan. Bij een *maakkontakt* resulteert de uitoefening van een kracht of een andere stimulus in een zeer goed geleidingsvermogen ($\sim$ kortsluiting) tussen de aansluitklemmen. Zonder kracht is er sprake van een open keten en is er geen geleidingsvermogen aanwezig. Bij een *verbreekkontakt* geldt het omgekeerde.

component	stimulus	effect
kontakt	kracht/magnetisch veld	zeer goed/geen geleidingsvermogen
bipolaire transistor	basisstroom/spanning	hoog/laag geleidingsvermogen
MOS transistor	gate spanning	hoger/geen geleidingsvermogen
fototransistor	licht	hoger/laag geleidingsvermogen
signaallampje	stroom	wel/geen licht uitstralend

Tabel 2.1. Overzicht van enkele componenten.

Veel componenten in de digitale techniek hebben overeenkomstige eigenschappen. Met een daartoe geschikte stimulus zijn de eigenschappen ervan te beïnvloeden. Meestal zal deze verandering het geleidingsvermogen betreffen en daarmee het vermogen om spanningen en stromen te beïnvloeden. Sommige componenten in het overzicht van tabel 2.1 zijn van nature al tweewaardig, zoals mechanische kontakten en signaallampjes.

Andere, zoals transistoren, laten binnen zekere grenzen een continue beïnvloeding van de eigenschappen toe en daarmee van de stroom erdoor. In de digitale techniek worden transistoren gewoonlijk in twee uiterste toestanden toegepast. Daardoor mogen we ze *beschouwen als tweewaardige componenten*.

Een maakcontact en een verbreekkontakt hebben een *complementaire werking* ten aanzien van de aangeboden stimulus. Ook bij de andere componenten komt dit voor. Een nMOS transistor gaat geleiden bij een positieve gate-source spanning, terwijl een pMOS transistor er dan juist mee ophoudt.

De *logische werking* van alle hiervoor beschreven componenten kunnen we beschrijven met een tweetal *proposities*, dit zijn uitspraken die waar of niet waar zijn. De ene propositie beschrijft de (aanwezigheid van de) stimulus aan de ingangskant, de andere het effect ervan op de component. Voor door middel van een kracht bediende kontakten is een geschikt tweetal proposities

'De toets is ingedrukt'

en

'Het kontakt geleidt'

Bij een nMOS transistor zijn de proposities

'De gate-source spanning is hoger dan de drempelspanning'

en

'De transistor heeft een (relatief) goed geleidingsvermogen'

Het verband tussen de twee proposities wordt in de digitale techniek vastgelegd in een *waarheidstabel*. Links geven we aan of de propositie, die de stimulus aan de ingangszijde definieert, waar is of niet waar is. Rechts wordt aangegeven of de bijbehorende propositie, die het effect op de eigenschappen van de component definieert, waar is. Voor een maakcontact leidt dit tot waarheidstabel 2.2 en voor een verbreekkontakt tot waarheidstabel 2.3. Doen we dit voor een nMOS en een pMOS transistor, dan ontstaan dezelfde tabellen. Alleen de formulering van de bijbehorende proposities in de kop van de tabel is anders.

toets ingedrukt	kontakt geleidt
niet waar	niet waar
waar	waar

Tabel 2.2. Waarheidstabel van een maakcontact.

toets ingedrukt	kontakt geleidt
niet waar	waar
waar	niet waar

Tabel 2.3. Waarheidstabel van een verbreekkontakt.

Keuze van de proposities

We zouden een kontakt ook kunnen specificeren met de proposities

'De toets is ingedrukt'

en

'Het kontakt geleidt *niet*'

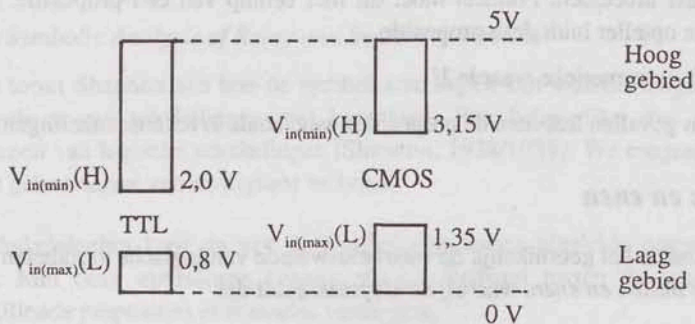
Voor een maakkontakt ontstaat dan tabel 2.4. Vergelijk ook tabel 2.2.

toets ingedrukt	kontakt geleidt niet
niet waar	waar
waar	niet waar

Tabel 2.4. Waarheidstabel.

We zien dat de waarheidstabel afhangt van de wijze waarop de proposities gedefinieerd zijn. Het is nodig hierover ondubbelzinnige afspraken te maken. Bij het hierna te introduceren IEC symbolensysteem is afgesproken proposities zo veel mogelijk 'positief' of 'bevestigend' te formuleren. In veel gevallen wordt daarmee verwarring voorkomen. Helaas kan dat niet altijd of is het niet duidelijk wat 'bevestigend' is. In dat geval moeten de gekozen proposities duidelijk aangegeven worden. We komen op deze problematiek in paragraaf 2.11 terug.

Voorbeeld



Figuur 2.1. Afspraken over de interpretatie van logische spanningsniveaus.

Figuur 2.1 specificeert hoe bij twee families van IC's de logische *ingangsspanningen* worden geïnterpreteerd. Er is een 'Hoog gebied' aangegeven en een 'Laag gebied'. Bevindt de ingangsspanning zich tussen deze twee gebieden in, dan is de werking van zo'n ingang en vaak ook de werking van het IC *niet gedefinieerd*. In feite geeft het Hoog en het Laag gebied aan waarin de ingangsspanning zich moet bevinden als de schakeling in rust is. Deze afspraken gelden dus niet als de schakeling in de overgangstoestand is. □

Een propositie abstraheert de werkelijkheid. Een werkelijkheid die vrijwel altijd veel meer mogelijkheden biedt dan de propositie kan beschrijven. Bij een logische toepassing van een component leggen we onszelf dus (veel) beperkingen op. Bijvoorbeeld door een transistor in twee uiterste toestanden toe te passen: dicht (lekstroom?) of goed (maximaal?) geleidend. Tussen deze uitersten in zit een gebied, waarin de transistor normaliter niet mag zijn. Alleen bij het omschakelen van de ene toestand naar de andere toestand is dit onvermijdelijk. Propositionen zijn dan ook uitsluitend waar/niet waar in de zogenaamde *statische toestand* van de component. In de *dynamische toestand*, de overgangstoestand, geldt de beschrijving met propositionen niet. Wat dit voor consequenties heeft bij het ontwerpen zullen we nog uitvoerig bespreken.

Logische variabelen of Booleaanse variabelen

Een propositie kan waar zijn (true of T) of niet waar zijn (false of F). De grootheden T en F noemen we de *waarheidswaarden van de propositie*. We kunnen aan elke propositie een variabele toekennen, die de waarheidswaarde van die propositie aangeeft. Zo'n variabele heeft twee waarden (T en F) en heet daarom een *logische variabele*. Vaak worden logische variabelen *Boolean variables* genoemd.

Een logische variabele is een bijzonder geval van een *binaire variabele*. Een binaire variabele is een variabele die precies twee waarden kan aannemen. Bij een binaire optelsectie bijvoorbeeld heeft elke ingang en elke uitgang de numerieke waarde 0 of 1, met het bijbehorende gewicht. Wordt aan elke ingang en uitgang nu een variabele toegekend die de waarde van die ingang of uitgang aangeeft, dan zijn dit binaire variabelen.

De stap van een binaire variabele naar een logische variabele is snel gezet. We kunnen ze één-op-één op elkaar afbeelden. Formeel moet dit met behulp van een propositie. In het geval van de binaire opteller luidt deze propositie

'Het bit ... heeft de numerieke waarde 1'

In vanzelfsprekende gevallen laat men de propositionen weg, zoals in rekenschakelingen.

Logische nullen en enen

In de digitale techniek is het gebruikelijk de waarheidswaarden van logische variabelen aan te geven met *logische nullen en enen*, waarbij de afspraak geldt dat

$$0 \leftrightarrow F \quad (\text{false}) \quad (2.1.a)$$

$$1 \leftrightarrow T \quad (\text{true}) \quad (2.1.b)$$

Dit zou in eerste instantie verwarring kunnen opleveren met het gebruik van numerieke enen en nullen bij onder andere binair rekenen. De praktijk wijst uit dat dit wel meevalt, mede gezien de toegepaste één-op-één afbeelding.

Samenvatting

Binnen de grenzen van het logische abstractieniveau kan de werking van componenten beschreven/gedefinieerd worden met proposities en de hiervan afgeleide logische variabelen. Dit schept de mogelijkheid de functie van een schakeling of groter systeem op logisch niveau te definiëren, waarmee veel van de in de logica ontwikkelde wiskundige technieken ter beschikking komen van de ontwerper. $\square$

Incidenteel past men ook meerwaardige logica toe. Zie hiervoor een speciale uitgave van het blad IEEE Computer, Special Issue on Multiple-Valued Logic, April 1988. We besteden hier verder geen aandacht aan.

2.2 Bewerkingen in de symbolische logica

De begrippen propositie en waarheidswaarde stammen uit de *symbolische logica*. Een belangrijke publicatie op dit gebied is

'An Investigation of the Laws of Thought'

van George Boole in 1854 [Boole, 1854]. Naar hem wordt de hierna te behandelen *schakelalgebra* de *Boole algebra* genoemd. De theorie van George Boole is later uitgebreid door onder andere Edward V. Huntington (1874-1952) met de Huntington Postulates en door August de Morgan (1806-1871). Naar De Morgan zijn enkele van de negatiewetten genoemd.

Een tweede belangrijke publicatie is die van Shannon uit 1938,

'A Symbolic Analysis of Relay and Switching Circuits'

Hierin toont Shannon aan hoe de symbolische logica kan worden toegepast op de analyse en synthese van schakelingen met contacten. Een belangrijke stap voorwaarts in het ontwerpen van logische schakelingen [Shannon, 1938/1939]. We mogen 1938 daarom zien als het geboortjaar van de digitale techniek.

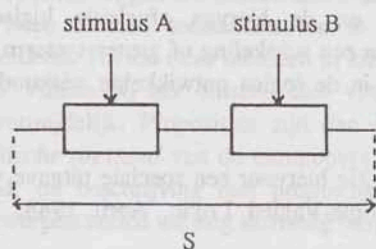
De schakelalgebra kent de van de symbolische logica afgeleide operatoren EN, OF en NIET. Met deze operatoren kunnen we het verband tussen de waarheidswaarde van verschillende proposities in formules vastleggen.

De logische operator EN

Figuur 2.2 geeft de *serieschakeling* weer van twee 'logische' componenten van hetzelfde type. Met behulp van twee stimuli A en B kunnen de componenten in geleiding gebracht worden. De stimulus heet dan actief. Het is duidelijk dat deze serieschakeling slechts kan geleiden als beide stimuli 'actief' zijn.

Bij deze serieschakeling behoort waarheidstabel 2.5. In tabel 2.5 duiden A en B proposities aan die aangeven dat de betreffende stimulus actief is, dat wil zeggen de waarde heeft die

de component in dit geval in de (hoog)geleidende toestand brengt. De propositie die het geleidingsvermogen van de hele schakeling beschrijft duiden we aan met S.



Keten S: werking beschreven door
'De keten S geleidt'

Figuur 2.2. Serieschakeling van logische componenten.

We zien uit tabel 2.5 dat de serieschakeling uitsluitend geleidend is als beide componenten geleidend zijn. In de symbolische logica wordt dit weergegeven met de *logische operator EN*,

$$S = A \text{ EN } B \quad (2.2)$$

Een voordeel van deze formele beschrijving van de serieschakeling is dat we nog geen uitspraak hoeven te doen over het soort componenten dat we gaan toepassen. De keuze ervan, kontakten of transistoren bijvoorbeeld, wordt uitgesteld tot een later stadium van het ontwerp.

De logische operator EN drukt uit dat de samengestelde propositie S dan en slechts dan waar is als A en B beide waar zijn. Tabel 2.5 mag dan ook gezien worden als de definitie van de logische operator EN.

A	B	S
F	F	F
F	T	F
T	F	F
T	T	T

(a) in waarheidswaarden F en T

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

(b) in logische nullen en enen

Tabel 2.5. Waarheidstabel van de serieschakeling (logische EN).

De logische operator EN wordt in de literatuur met verschillende *symbolen* aangeduid,

$$\text{EN}, \wedge, *, \cdot \quad (2.3)$$

In deze tekst kiezen we voor de halfhoge punt. De formule, die het verband tussen de proposities A en B enerzijds en S anderzijds uitdrukt, is dan:

$$S = A \text{ EN } B \leftrightarrow S = A \cdot B \quad (2.4)$$

Vaak laten we ook deze punt weg en schrijven we

$$S = AB \quad (2.4.a)$$

De logische operator OF

Figuur 2.3 toont de parallelschakeling van twee logische componenten. Tabel 2.6 is de waarheidstabel van de parallelschakeling. De parallelschakeling geleidt als één van beide takken geleidt, of als beide takken geleiden. We drukken een dergelijk verband via de bijbehorende proposities uit in de *logische operator OF*

$$S = A \text{ OF } B \quad (2.5)$$

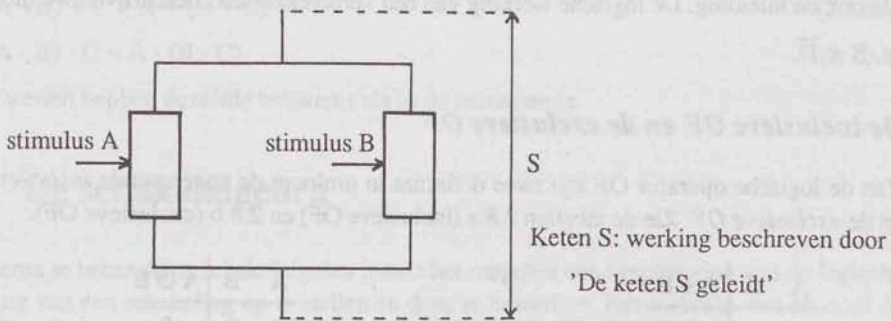
De logische operator OF wordt meestal aangeduid met een van de *symbolen*

$$\text{OF, } \vee, + \quad (2.6)$$

In deze tekst passen we de + toe. Zodat

$$S = A \text{ OF } B \leftrightarrow S = A + B \quad (2.7)$$

Ook voor tabel 2.6.a van de logische operator OF bestaat een variant in logische nullen en enen, de tabel 2.6.b.



Figuur 2.3. Parallelschakeling van logische componenten.

A	B	S
F	F	F
F	T	T
T	F	T
T	T	T

(a) in waarheidswaarden F en T

A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

(b) in logische nullen en enen

Tabel 2.6. Waarheidstabel van de parallelschakeling (logische OF).

De logische operator NIET

De *logische operator NIET* dient om de ontkenning van een uitspraak of propositie aan te geven. Schrijfwijzen zijn:

$$\text{NIET } A, \bar{A}, \neg A, A' \quad (2.8)$$

In deze tekst gebruiken we de *negatiestreek* boven een propositie of boven een in proposities uitgedrukte formule. De operator NIET is gedefinieerd in tabel 2.7.

A	$\bar{A}$
F	T
T	F

$$F \leftrightarrow 0$$

(a) in waarheidswaarden T en F

A	$\bar{A}$
0	1
1	0

(b) in logische nullen en enen

Tabel 2.7. De logische operator NIET.

Voorbeeld

De logische werking van een maakcontact kan beschreven worden met de logische formule (S = 1 geleidend, D = 1 kontakt ingedrukt)

$$S = D$$

Zie ook de inleiding. De logische werking van een verbreekkontakt beschrijven we met

$$S = \bar{D}.$$

□

De inclusieve OF en de exclusieve OF

Van de logische operator OF zijn twee definities in omloop, de zogenaamde *inclusieve OF* en de *exclusieve OF*. Zie de tabellen 2.8.a (inclusieve OF) en 2.8.b (exclusieve OF).

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

(a) inclusieve OF

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

(b) exclusieve OF

Tabel 2.8. De inclusieve en de exclusieve OF.

De inclusieve OF is de meest gebruikte definitie. De exclusieve OF drukt uit wat we bij ruzies wel eens willen zeggen met

'of hij er uit, of ik er uit'

waarmee we aangeven dat voor beide tegelijk geen plaats is. Voor speciale toepassingen komen we de exclusieve OF wel tegen, bijvoorbeeld bij het formuleren van de eigenschappen van maximum-lengte reeksen van schuifregisters (hoofdstuk 16). Ook bij het beschrijven van de eigenschappen van foutendetectorende en corrigerende codes komen we de exclusieve OF operator tegen. Als we in deze tekst spreken van de bewerking 'OF', dan bedoelen we de inclusieve OF, tenzij uitdrukkelijk anders vermeld.

Volgorde van bewerkingen

Komen in een formule verschillende logische operatoren voor, dan is de *volgorde bij interpretatie van de formule*

- voer eerst alle operaties NIET uit;
- dan alle operaties EN;
- dan alle operaties OF.

Door *haakjes* kan deze volgorde naar behoefte gewijzigd worden, evenals dat in de gewone rekenkunde gedaan wordt.

Verder geldt een aantal wetten zoals de *commutatieve wet*,

$$A + B = B + A \quad (2.9.a)$$

$$A \cdot B = B \cdot A \quad (2.9.b)$$

en de *associatieve wet*

$$(A + B) + C = A + (B + C) \quad (2.10.a)$$

$$(A \cdot B) \cdot C = A \cdot (B \cdot C) \quad (2.10.b)$$

Deze wetten hebben dezelfde betekenis als in de rekenkunde.

2.3 De schakelalgebra

De hierna te behandelen schakelalgebra maakt het mogelijk een beschrijving van de logische werking van een schakeling op te stellen en deze te bewerken, bijvoorbeeld met als doel de opgestelde beschrijving te reduceren of om deze af te beelden op een bepaald type componenten. In de digitale techniek is het gebruikelijk formules uit te drukken in logische variabelen en de operatoren EN, OF en NIET. De waarheidswaarden F en T worden genoteerd als 0 en 1 (afspraak (2.1)). Ter introductie het volgende voorbeeld.

Voorbeeld

y	z	S
0	0	0
0	1	1
1	0	1
1	1	1

$$S = y + z$$

$$S = \bar{y} \cdot z + y \cdot \bar{z} + y \cdot z$$

Tabel 2.9. Waarheidstabel met formules.

Naast de waarheidstabel 2.9 van de logische OF staan twee formules. Beide beschrijven hoe de logische variabele S afhangt van de logische variabelen y en z. De formule

$$S = y + z \quad (2.11)$$

correspondeert met de wijze waarop de logische OF geïntroduceerd is. De andere vorm van de formule

$$S = \bar{y} \cdot z + y \cdot \bar{z} + y \cdot z \quad (2.12)$$

past meer bij de wijze waarop de waarheidstabel wordt uitgelezen, namelijk regel voor regel. Omdat beide formules tot dezelfde tabel leiden, moet het mogelijk zijn om ze in elkaar om te rekenen. We zullen nagaan welke rekenregels daarbij nodig zijn.

Achter elke regel in de nu volgende afleiding is vermeld onder welke voorwaarde(n) de betreffende stap is toegestaan.

$$\begin{aligned} S &= \bar{y} \cdot z + y \cdot \bar{z} + y \cdot z \\ &= \bar{y} \cdot z + y \cdot z + y \cdot \bar{z} && y \cdot \bar{z} + y \cdot z = y \cdot z + y \cdot \bar{z} \\ &= \bar{y} \cdot z + y \cdot z + y \cdot z + y \cdot \bar{z} && y \cdot z = y \cdot z + y \cdot z \\ &= (\bar{y} + y) \cdot z + y \cdot (z + \bar{z}) && \text{op voorwaarde dat o.a. } \bar{y} \cdot z + y \cdot z = (\bar{y} + y) \cdot z \\ &= 1 \cdot z + y \cdot 1 && \bar{y} + y = 1 \text{ en } z + \bar{z} = 1 \\ &= y + z && 1 \cdot z = z \text{ en } y \cdot 1 = y \end{aligned} \quad (2.13)$$

De rekenregels mogen in beide richtingen worden toegepast. Afleiding (2.13) mag ook van onder naar boven gelezen worden.

Aan het feit dat S met een hoofdletter is geschreven en de variabelen y en z met een kleine letter moet geen speciale betekenis gehecht worden. S duidt de functie aan, of bij schakelingen het uitgangssignaal. De variabelen in kleine letters duiden de ingangswaarden of ingangssignalen aan. $\square$

Het stelsel rekenregels volgens welke bepaalde formules in andere (meestal eenvoudigere) formules kunnen worden omgerekend, wordt aangeduid met de naam *schakelalgebra*. Het volgende stelsel blijkt in de praktijk gemakkelijk hanteerbaar.

Volgorde van bewerkingen

In de schakelalgebra is de volgorde bij de interpretatie van formules (zie ook paragraaf 2.2):

- voer eerst alle negaties uit, aangeduid met $-$
- daarna alle logische bewerkingen EN, aangeduid met $\cdot$
- tenslotte de logische bewerkingen OF, aangeduid met $+$

Deze volgorde kan met haken gewijzigd worden, zoals bij de normale algebra. Alle logische formules moeten volgens deze regels geïnterpreteerd worden.

Rekenregels voor constanten

De 0 en de 1 zijn de enige constanten in de schakelalgebra. Zij representeren de logische waarden 'niet waar' en 'waar'. Voor de constanten 0 en 1 gelden de volgende rekenregels.

$$\begin{array}{lll}
 0 + 0 = 0 & 0 \cdot 0 = 0 & \\
 0 + 1 = 1 & 0 \cdot 1 = 0 & \bar{0} = 1 \\
 1 + 0 = 1 & 1 \cdot 0 = 0 & \bar{1} = 0 \\
 1 + 1 = 1 & 1 \cdot 1 = 1 &
 \end{array} \quad (2.14)$$

Deze rekenregels voor constanten volgen direct uit de definitie van de logische operatoren EN, OF en NIET. In formules waarin + en $\cdot$ voorkomen wordt rond de $\cdot$ gewoonlijk minder spatie aangehouden. De onderlinge hiërarchie van de bewerkingen komt dan beter uit.

Rekenregels voor logische variabelen

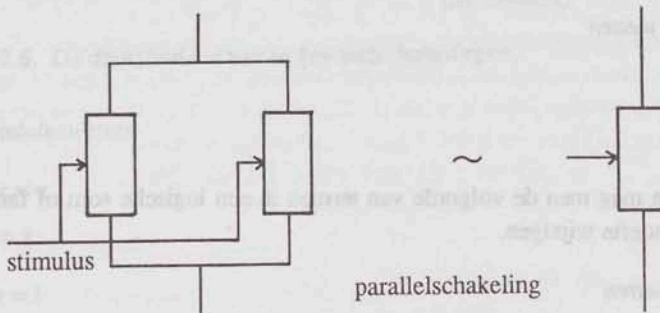
1. De gelijkheidswetten

$$z + z = z \quad (2.15.a)$$

$$z \cdot z = z \quad (2.15.b)$$

Wiskundig zeggen de gelijkheidswetten dat in een logische som of in een logisch product identieke variabelen mogen worden geschrapt. Zij mogen echter ook worden toegevoegd. Voor logische schakelingen betekenen deze wetten dat een parallelschakeling van identieke componenten of een serieschakeling ervan dezelfde logische werking heeft als één zo'n component. (We noemen componenten 'identiek' als de component van hetzelfde type zijn en dezelfde stimulus krijgen aangeboden.)

Voorbeeld

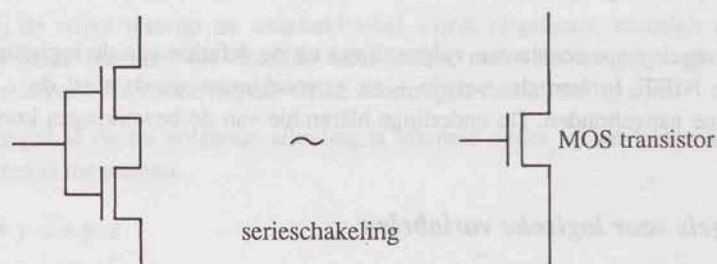


Figuur 2.4. Logisch identieke schakelingen.

De in figuur 2.4 symbolisch aangeduide parallelschakeling en de in figuur 2.5 in nMOS aangegeven serieschakeling zijn elk *logisch identiek* met de ernaast beschreven schakeling en wel op grond van de gelijkheidswetten. Elektrisch is er natuurlijk een verschil, een serieschakeling van twee identieke transistoren heeft in de geleidende toestand de dubbele weerstand en spanningsval. Zoals eerder is opgemerkt zijn dit details die op het *logische abstractieniveau* verborgen blijven. Een keuze die we in paragraaf 2.1. gemotiveerd

hebben. Voor de schakelalgebra betekenen de gelijkheidswetten dat in formules coëfficiënten anders dan 0 en 1 en machten van variabelen niet voorkomen/niet zinvol zijn.

Voorbeeld



Figuur 2.5. Logisch identieke ketens van nMOS transistoren.

2. De associatieve wetten

$$(x + y) + z = x + (y + z) \quad (2.16.a)$$

$$(x \cdot y) \cdot z = x \cdot (y \cdot z) \quad (2.16.b)$$

De associatieve wetten zeggen dat bij de interpretatie van een logische formule de aangehouden volgorde bij identieke logische bewerkingen onbelangrijk is. Dat ook de volgorde van de termen er niet toe doet is een gevolg van de commutatieve wetten. Mede daarom laat men de haakjes in de wetten (2.16) en volgende gewoonlijk weg, hoewel + en $\cdot$ formeel binaire operatoren zijn (werkend op twee operanden).

3. De commutatieve wetten

$$y + z = z + y \quad (2.17.a)$$

$$y \cdot z = z \cdot y \quad (2.17.b)$$

Volgens deze wetten mag men de volgorde van termen in een logische som of factoren in een produkt naar behoefte wijzigen.

4. De distributieve wetten

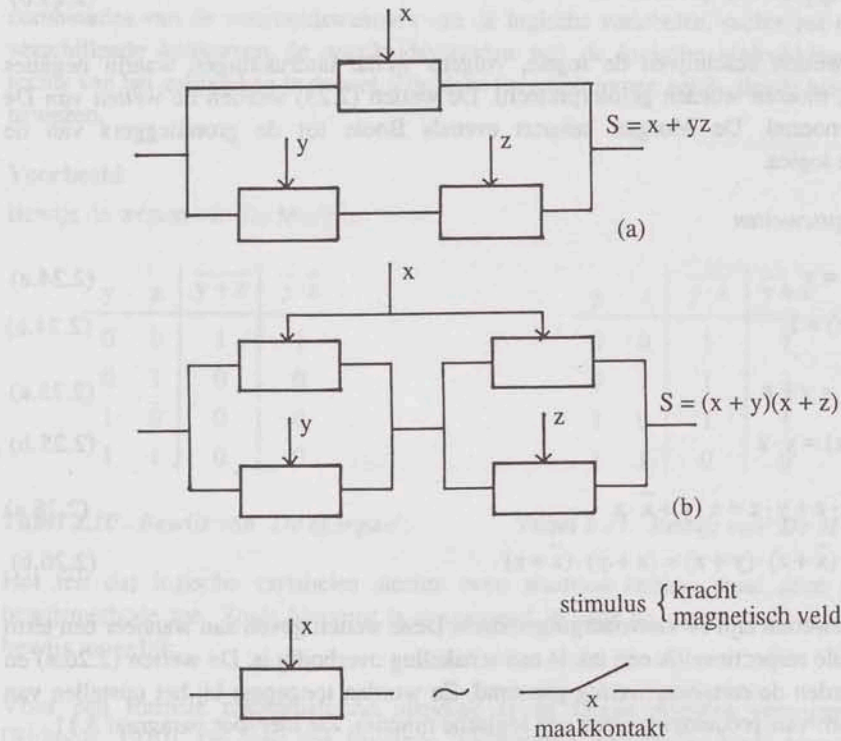
$$x + y \cdot z = (x + y) \cdot (x + z) \quad (2.18.a)$$

$$x \cdot (y + z) = x \cdot y + x \cdot z \quad (2.18.b)$$

De distributieve wetten geven aan hoe haakjes binnen respectievelijk buiten formules gebracht kunnen worden.

Het is voor de distributieve wetten interessant de met deze logische uitdrukkingen overeenkomende kontaktschakelingen te bekijken. Figuur 2.6 geeft de bij de eerste distributieve wet behorende schakelingen. De schakelingen zijn *logisch equivalent*. De

eerste schakeling bezit drie kontakten en verdient vanuit het standpunt van materiaalgebruik de voorkeur. De distributieve wetten kunnen gebruikt worden als een hulpmiddel bij het vereenvoudigen van schakelingen.



Figuur 2.6. De distributieve wet in kontaktschakelingen.

5. De moduluswetten

$$0 + z = z \quad (2.19.a)$$

$$1 \cdot z = z \quad (2.19.b)$$

$$1 + z = 1 \quad (2.20.a)$$

$$0 \cdot z = 0 \quad (2.20.b)$$

De moduluswetten beschrijven hoe formules met constanten en logische variabelen moeten worden geïnterpreteerd.

6. De negatiewetten

$$\bar{\bar{z}} + z = 1 \quad (2.21.a)$$

$$\bar{\bar{z}} \cdot z = 0 \quad (2.21.b)$$

$$\overline{(\overline{z})} = z \quad (2.22)$$

$$\overline{y + z} = \overline{y} \cdot \overline{z} \quad (2.23.a)$$

$$\overline{y \cdot z} = \overline{y} + \overline{z} \quad (2.23.b)$$

De wetten van de Morgan

De negatiewetten beschrijven de regels, volgens welke uitdrukkingen waarin negaties voorkomen, moeten worden geïnterpreteerd. De wetten (2.23) worden de *wetten van De Morgan* genoemd. De Morgan behoort evenals Boole tot de grondleggers van de symbolische logica.

7. De absorptiewetten

$$z + y \cdot z = z \quad (2.24.a)$$

$$z \cdot (y + z) = z \quad (2.24.b)$$

$$y + \overline{y} \cdot z = y + z \quad (2.25.a)$$

$$y \cdot (\overline{y} + z) = y \cdot z \quad (2.25.b)$$

$$x \cdot y + \overline{x} \cdot z + y \cdot z = x \cdot y + \overline{x} \cdot z \quad (2.26.a)$$

$$(x + y) \cdot (\overline{x} + z) \cdot (y + z) = (x + y) \cdot (\overline{x} + z) \quad (2.26.b)$$

De absorptiewetten zijn *vereenvoudigingswetten*. Deze wetten geven aan wanneer een term in een formule respectievelijk een tak in een schakeling overbodig is. De wetten (2.26.a) en (2.26.b) worden de *consensuswetten* genoemd. Ze worden toegepast bij het opstellen van minimale som-van-produktenvormen van logische functies. Zie hiervoor paragraaf 3.11.

Onder de wetten 1 t/m 7 komt een aantal *algebraïsch overbodige* wetten voor. Een wet is overbodig als de wet direct afleidbaar is uit eerder gegeven wetten. De gegeven set wetten blijkt in de praktijk echter uitstekend te voldoen. Men kan zich veel werk besparen met de absorptiewetten, hoewel zij voor de definitie van de schakelalgebra overbodig zijn.

Voorbeeld

Bewijs van wet (2.24.a)

$$\begin{aligned} z + y \cdot z &= 1 \cdot z + y \cdot z && \text{via (2.19.b)} \\ &= (1 + y) \cdot z && \text{via (2.17) en (2.18.b)} \\ &= 1 \cdot z && \text{via (2.20.a)} \\ &= z && \text{via (2.19.b)} \end{aligned}$$

Bewijs van wet (2.25.b)

$$\begin{aligned} y \cdot (\overline{y} + z) &= y \cdot \overline{y} + y \cdot z && \text{via (2.18.b)} \\ &= 0 + y \cdot z && \text{via (2.17) en (2.21.b)} \\ &= y \cdot z && \text{via (2.19.a)} \end{aligned} \quad \square$$

Het bewijzen van schakelwetten

Een formeel bewijs voor de juistheid van de boven beschreven logische wetten kan men ook leveren via een waarheidstabel. Links van de vertikale streep zet men alle mogelijke combinaties van de waarheidswaarden van de logische variabelen, rechts zet men in twee verschillende kolommen de waarheidswaarden van de logische uitdrukkingen links en rechts van het gelijkteken in de wet. Zijn deze twee kolommen gelijk, dan is hiermee de wet bewezen.

Voorbeeld

Bewijs de wetten van De Morgan.

y	z	$\overline{y+z}$	$\overline{y} \cdot \overline{z}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Tabel 2.10. Bewijs van 'De Morgan'.

y	z	$\overline{y \cdot z}$	$\overline{y} + \overline{z}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Tabel 2.11. Bewijs van 'De Morgan'.

Het feit dat logische variabelen slechts twee waarden hebben staat deze eenvoudige bewijsmethode toe. Zoals hiervoor is aangetoond is voor de wetten ook een algebraïsch bewijs mogelijk. $\square$

Voor een formele mathematische inleiding in de Boole Algebra verwijzen we naar [Birkhoff, 1970]. De door ons gevolgde 'engineering' benadering komt overeen met die van onder andere McCluskey [McCluskey, 1986]. Zie ook [Dwinger, 1961; Hohn, 1966; Whitesitt, 1961].

2.4 Uitbreidbaarheid van schakelwetten

Een propositie kan bestaan uit een

- enkelvoudige propositie;
- samengestelde propositie, bestaande uit een of meer proposities en de operatoren EN, OF en NIET (samengesteld volgens de in de logica beschreven regels).

Hetzelfde is van toepassing op logische variabelen. Elke toegestane uitdrukking in logische variabelen is weer een logische variabele. Op grond van deze conclusie en de associatieve wetten kan de interpretatie van de bewerkingen en de erop gedefinieerde rekenregels uitgebreid worden.

Voorbeeld *Uitbreiding van De Morgan:* $\overline{y+z} = \bar{y} \cdot \bar{z}$

$$\begin{aligned}\overline{x+y+z} &= \overline{(x+y)+z} \\ &= \overline{(x+y)} \cdot \bar{z} \\ &= (\bar{x} \cdot \bar{y}) \cdot \bar{z} \\ &= \bar{x} \cdot \bar{y} \cdot \bar{z}\end{aligned}\quad (2.27.a)$$

Op overeenkomstige wijze:

$$\overline{x \cdot y \cdot z} = \bar{x} + \bar{y} + \bar{z} \quad \square (2.27.b)$$

Voorbeeld *Toepassing van de wet:* $y + \bar{y} \cdot z = y + z$

Uit deze wet volgt

$$\begin{aligned}a + b + \bar{a} \cdot \bar{b} \cdot c &= a + b + (\bar{a} \cdot \bar{b}) \cdot c \\ &= (a + b) + \overline{(a + b)} \cdot c \\ &= (a + b) + c \\ &= a + b + c\end{aligned}\quad \square (2.28)$$

Voorbeeld $x \cdot (y + z) = x \cdot y + x \cdot z$ (distributieve wet)

Wat is nu

$$a \cdot (b_1 + b_2 + b_3)$$

De uitwerking hiervan geschiedt als volgt.

$$\begin{aligned}a \cdot (b_1 + b_2 + b_3) &= a \cdot ((b_1 + b_2) + b_3) \\ &= a \cdot (b_1 + b_2) + a \cdot b_3 \\ &= a \cdot b_1 + a \cdot b_2 + a \cdot b_3\end{aligned}$$

$$\begin{aligned}a \cdot (b_1 + b_2 + b_3 + b_4) &= a \cdot ((b_1 + b_2 + b_3) + b_4) \\ &= a \cdot (b_1 + b_2 + b_3) + a \cdot b_4 \\ &= a \cdot b_1 + a \cdot b_2 + a \cdot b_3 + a \cdot b_4\end{aligned}$$

enz. Tenslotte vinden we de uitwerking van

$$a \cdot (b_1 + b_2 + \dots + b_n)$$

met behulp van volledige inductie naar n:

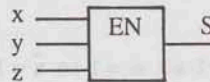
$$a \cdot (b_1 + b_2 + \dots + b_n) = a \cdot b_1 + a \cdot b_2 + \dots + a \cdot b_n \quad \square (2.29)$$

Hiermee is aangetoond hoe schakelwetten naar behoefte uitgebreid kunnen worden tot wetten in meer variabelen.

2.5 De mintermvorm van logische functies

Waarheidstabel 2.12 behoort bij een EN poort met drie ingangen. Een EN poort is een component, waarvan de uitgang S dan en slechts dan 1 is als alle ingangsvariabelen de logische waarde 1 hebben.

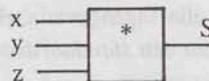
x	y	z	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



Tabel 2.12. Waarheidstabel van de logische operator EN op drie variabelen.

Tabel 2.13 beschrijft de algemene vorm van een waarheidstabel voor logische functies in drie variabelen. Bij het beschrijven (specificeren) van de logische functie van een component met drie ingangen wordt voor elke ingangscombinatie gespecificeerd wat de component doet (moet doen). De bij de i^{de} ingangscombinatie behorende uitgangswaarde noemen we de functiewaarde f_i . Aldus wordt voor elk van de acht ingangscombinaties een functiewaarde gespecificeerd. In een waarheidstabel is elke ingangscombinatie uniek en de tabel bevat ook alle mogelijke ingangscombinaties.

x	y	z	S*
0	0	0	f_0
0	0	1	f_1
0	1	0	f_2
0	1	1	f_3
1	0	0	f_4
1	0	1	f_5
1	1	0	f_6
1	1	1	f_7



*logische functie
nader te specificeren

Tabel 2.13. Waarheidstabel (algemene vorm).

Het rangnummer i van de functiewaarde f_i volgt uit de binaire waarde (formule 1.2) van de bijbehorende ingangscombinatie. Dit nummer hangt derhalve af van de volgorde van de variabelen in de kop van de tabel. Het verdient daarom aanbeveling een vaste volgorde aan te houden!

Hulpstelling 1

Een produkt van logische variabelen en/of logische functies is dan en slechts dan 1 als elk van de factoren in het produkt 1 is.

Bewijs. Volgt direct uit de rekenregels. $\square$

Hulpstelling 2

Een produkt in n logische variabelen is 1 voor slechts één combinatie van de variabelen.

Bewijs. Triviaal. $\square$

Voorbeeld

De produktterm $w\bar{x}y\bar{z}=1$ als $w=1$ en $\bar{x}=1 \rightarrow x=0$ en $y=1$ en $\bar{z}=1 \rightarrow z=0$. Voor de overige combinaties van w t/m z is dit produkt 0. $\square$

De mintermvorm

Elke regel in een waarheidstabel kan geïdentificeerd worden met een produkt m_i van alle ingangsvariabelen in de kop van de tabel. Een variabele komt dan gewoon in het produkt voor als op de betreffende regel een 1 onder de variabele genoteerd staat. Anders komt de variabele genegeerd voor, dat wil zeggen met een negatiestreep erboven. Voor het rangnummer i van de m_i 's geldt eenzelfde afspraak als voor het rangnummer i van de functiewaarden f_i hiervoor.

Voor een functie in drie variabelen x t/m z zijn de m_i 's

$$\begin{array}{ll} m_0 = \bar{x}\bar{y}\bar{z} & m_4 = x\bar{y}\bar{z} \\ m_1 = \bar{x}\bar{y}z & m_5 = x\bar{y}z \\ m_2 = \bar{x}y\bar{z} & m_6 = x y \bar{z} \\ m_3 = \bar{x}y z & m_7 = x y z \end{array} \quad (2.30)$$

Een produkt m_i waarin alle ingangsvariabelen één keer voorkomen wordt een minterm genoemd. We komen nu tot een *standaardvorm* (canonieke vorm) van een logische functie in drie variabelen, de *mintermvorm*.

$$S = m_0 \cdot f_0 + m_1 \cdot f_1 + m_2 \cdot f_2 + m_3 \cdot f_3 + m_4 \cdot f_4 + m_5 \cdot f_5 + m_6 \cdot f_6 + m_7 \cdot f_7 \quad (2.31)$$

of uitgeschreven in de variabelen

$$S = \bar{x}\bar{y}\bar{z} \cdot f_0 + \bar{x}\bar{y}z \cdot f_1 + \bar{x}y\bar{z} \cdot f_2 + \bar{x}yz \cdot f_3 + x\bar{y}\bar{z} \cdot f_4 + x\bar{y}z \cdot f_5 + xy\bar{z} \cdot f_6 + xyz \cdot f_7 \quad (2.32)$$

Op grond van het vorenstaande volgt dat bij een bepaalde ingangscombinatie één minterm tegelijk de waarde 1 heeft. De bij deze minterm behorende functiewaarde f_i bepaalt dan de *momentele waarde* van de functie S , dit is de waarde indien de betreffende ingangscombinatie wordt aangeboden.

De concepten die hebben geleid tot de mintermvorm van de logische functie in drie variabelen kunnen zonder meer uitgebreid worden tot functies in n variabelen. Een

minterm van een functie in n variabelen bevat dan n factoren, elke variabele komt één keer voor en wel gewoon of genegeerd. Samengevat zijn de eigenschappen van mintermen:

Eigenschap 1

$$m_i = 1 \text{ voor precies één ingangscombinatie} \quad \square \quad (2.33)$$

Eigenschap 2

$$m_i \cdot m_j = 0 \text{ voor } i \neq j \quad (2.34)$$

(Maximaal één minterm heeft de waarde 1.) $\square$

Eigenschap 3

$$\sum_{i=0}^{2^n-1} m_i = 1 \quad (2.35)$$

(Er is altijd één minterm die 1 is.) $\square$

Voorbeeld

Tabel 2.14 specificeert een schakeling, waarmee van drie binaire signalen x , y en z met gewichten 4, 2 en 1 bepaald wordt of de waarde van xyz_{BIN} groter dan of gelijk aan 3 is.

x	y	z	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Proposities

x : 'Signaal x heeft de waarde 2^2 ,

y : 'Signaal y heeft de waarde 2^1 ,

z : 'Signaal z heeft de waarde 2^0 ,

S : 'Het binaire gewicht van de combinatie is groter dan of gelijk aan 3'

Tabel 2.14. Waarheidstabel.

De bij tabel 2.14 behorende mintermvorm van de functie S is:

$$\begin{aligned} S &= \bar{x}\bar{y}\bar{z} \cdot 0 + \bar{x}\bar{y}z \cdot 0 + \bar{x}y\bar{z} \cdot 0 + \bar{x}yz \cdot 1 + x\bar{y}\bar{z} \cdot 1 + x\bar{y}z \cdot 1 + xy\bar{z} \cdot 1 + xyz \cdot 1 \\ &= \bar{x}yz + x\bar{y}\bar{z} + \bar{x}yz + x\bar{y}z + xyz \end{aligned} \quad (2.36)$$

(In de mintermvorm laat men termen die 0 zijn meestal weg.) $\square$

De som-van-produktenvorm

Uit tabel 2.14 volgt dat de daarin gespecificeerde functie S ook geschreven kan worden als

$$S = yz + x \quad (2.37.a)$$

Een dergelijke vorm van de functie noemen we een *som-van-produktenvorm*.

Voor een functie kan vaak meer dan één som-van-produktenvorm opgesteld worden. Voor tabel 2.14 bijvoorbeeld

$$S = yz + x\bar{y} + xy \quad (2.37.b)$$

of

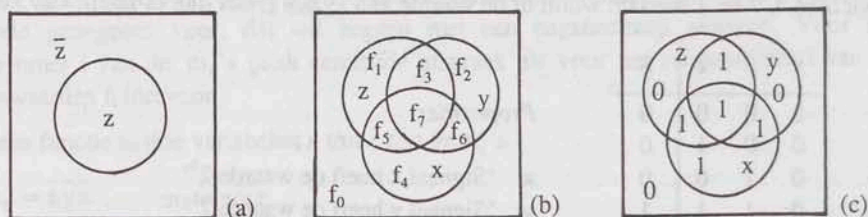
$$S = \bar{x}yz + x\bar{z} + xz \quad (2.37.c)$$

Welke hiervan de eenvoudigste is hangt mede af van de beschikbare componenten. *Het is gebruikelijk als eenvoudigste som-van-produktenvorm die vorm te nemen met het kleinste aantal produkten en de omvang van de produkten vervolgens ook zo klein mogelijk te nemen. Er kan dan geen produkterterm noch een variabele uit een term worden weggelaten zonder de formule te veranderen.*

Opmerking

De mintermvorm is ook een som-van-produktenvorm, maar meestal niet de eenvoudigste som-van-produktenvorm. $\square$

Venndiagrammen



Figuur 2.7. Venndiagrammen.

In figuur 2.7.a is een Venndiagram getekend voor één variabele. Het door de cirkel omsloten gebied correspondeert met $z=1$, ook te interpreteren als het gebied waar de propositie Z waar is. Het gebied buiten de cirkel correspondeert met $z=0$. Het hele door de rechthoek omsloten gebied wordt het *universum* genoemd. Hierin zijn alle mogelijkheden omvat, zowel $z=0$ als $z=1$.

Figuur 2.7.b toont een Venndiagram voor een willekeurige functie in drie variabelen. In het diagram zijn acht verschillende gebieden aan te wijzen, die corresponderen met de acht mintermen van een functie in drie variabelen. In elk vak is de bijbehorende functiewaarde geplaatst.

Venndiagrammen worden wel gebruikt bij de vereenvoudiging van logische functies. In figuur 2.7.c zijn de functiewaarden ingevuld die corresponderen met de in tabel 2.14 gegeven functie. We zien onmiddellijk dat het gebied waar de functie de waarde 1 heeft bestaat uit de vereniging van de gebieden x en yz . De meest eenvoudige vorm van de functie als som-van-produkten is formule 2.37.a.

Nu het Venndiagram geïntroduceerd is, zal ook de betekenis van het woord min(imale)term duidelijk zijn. In het volgende hoofdstuk wordt een meer gestileerde vorm van het Venndiagram geïntroduceerd, het Karnaughdiagram, speciaal ten behoeve van het vereenvoudigen van functies.

2.6 De maxtermvorm van logische functies

Soms wordt een andere standaardvorm van logische functies toegepast en wel de *maxtermvorm*. Dit is een *produkt-van-sommenvorm* die er voor drie variabelen uit ziet als

$$S = (M_0 + f_0) \cdot (M_1 + f_1) \cdot (M_2 + f_2) \cdot (M_3 + f_3) \cdot (M_4 + f_4) \cdot (M_5 + f_5) \cdot (M_6 + f_6) \cdot (M_7 + f_7) \quad (2.38)$$

of in de variabelen uitgedrukt

$$S = (x + y + z + f_0) \cdot (x + y + \bar{z} + f_1) \cdot (x + \bar{y} + z + f_2) \cdot (x + \bar{y} + \bar{z} + f_3) \cdot (\bar{x} + y + z + f_4) \cdot (\bar{x} + y + \bar{z} + f_5) \cdot (\bar{x} + \bar{y} + z + f_6) \cdot (\bar{x} + \bar{y} + \bar{z} + f_7) \quad (2.39)$$

Een *maxterm* M_i vervult in een produktvorm dezelfde rol als een *minterm* m_i in een somvorm van de logische functie. Een functiewaarde f_i kan de waarde van de functie S bepalen als de betreffende maxterm M_i de waarde $M_i = 0$ heeft en alle andere maxtermen de waarde 1 bezitten. Dan geldt voor $M_3 = 0$ bijvoorbeeld

$$\begin{aligned} S &= (1 + f_0) \cdot (1 + f_1) \cdot (1 + f_2) \cdot (0 + f_3) \cdot (1 + f_4) \cdot (1 + f_5) \cdot (1 + f_6) \cdot (1 + f_7) \\ &= 1 \cdot 1 \cdot 1 \cdot f_3 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \\ &= f_3 \end{aligned} \quad (2.40)$$

Een en ander berust op de hulpstelling dat een som van variabelen dan en slechts dan 0 is als alle termen (variabelen) in de som 0 zijn. Dit is slechts voor één specifieke ingangscombinatie het geval. (Vergelijk hulpstelling 1 voor produkttermen.)

Verband tussen de mintermvorm en de maxtermvorm

Zij gegeven een functie in drie variabelen. De negatie van deze functie ontstaat door alle functiewaarden te negeren, dat wil zeggen van een NIET te voorzien. Zie ook paragraaf 2.7. We kunnen de oorspronkelijke functie weer terugkrijgen door het geheel nogmaals te negeren. In formule:

$$\bar{S} = m_0 \cdot \bar{f}_0 + m_1 \cdot \bar{f}_1 + m_2 \cdot \bar{f}_2 + m_3 \cdot \bar{f}_3 + m_4 \cdot \bar{f}_4 + m_5 \cdot \bar{f}_5 + m_6 \cdot \bar{f}_6 + m_7 \cdot \bar{f}_7$$

Waaruit met behulp van de wetten van De Morgan volgt

$$\begin{aligned} S &= \bar{\bar{S}} \\ &= \overline{(m_0 \cdot \bar{f}_0 + m_1 \cdot \bar{f}_1 + m_2 \cdot \bar{f}_2 + m_3 \cdot \bar{f}_3 + m_4 \cdot \bar{f}_4 + m_5 \cdot \bar{f}_5 + m_6 \cdot \bar{f}_6 + m_7 \cdot \bar{f}_7)} \\ &= (\bar{m}_0 + f_0) \cdot (\bar{m}_1 + f_1) \cdot (\bar{m}_2 + f_2) \cdot (\bar{m}_3 + f_3) \cdot (\bar{m}_4 + f_4) \cdot (\bar{m}_5 + f_5) \cdot (\bar{m}_6 + f_6) \cdot (\bar{m}_7 + f_7) \end{aligned} \quad (2.41)$$

Stellen we dit gelijk aan de maxtermvorm,

$$S = (M_0 + f_0) \cdot (M_1 + f_1) \cdot (M_2 + f_2) \cdot (M_3 + f_3) \cdot (M_4 + f_4) \cdot (M_5 + f_5) \cdot (M_6 + f_6) \cdot (M_7 + f_7)$$

dan vinden we

$$m_i = \overline{M}_i \text{ of } \overline{m}_i = M_i \quad (2.42)$$

Ook de naam *maxtermvorm* kan nu verklaard worden. Elke maxterm (maximale term) correspondeert met een zo groot mogelijk gebied in een Venndiagram, dat nog net niet samenvalt met het universum. Een maxterm M_i is disjunct met de ermee corresponderende minterm m_i

$$m_i \cdot M_i = m_i \cdot \overline{m}_i = 0$$

Het gebied in een Venndiagram dat correspondeert met maxterm M_i is gelijk aan het gebied, dat verkregen wordt door van het universum het met minterm m_i corresponderende gebied af te trekken. De gebieden van m_i en M_i zijn *complementair*.

De mintermvorm en de maxtermvorm worden de *canonieke vormen* van een logische functie genoemd. Beide vormen worden zelden direct in deze vorm in een schakeling gerealiseerd. Daarvoor kiest men liever een vereenvoudigde uitdrukking, die minder materiaal kost. In het volgende hoofdstuk worden daartoe enkele vereenvoudigingsmethoden behandeld.

Voorbeeld

Bepaal de mintermvorm, de maxtermvorm en de meest eenvoudige somvorm en produktvorm van de functie S ,

$$S = \overline{x}\overline{y}\overline{z} + \overline{x}\overline{y}z + \overline{x}y\overline{z} + \overline{x}yz + x\overline{y}\overline{z} + x\overline{y}z + xy\overline{z} + xyz \quad (2.43)$$

Oplossing:

De mintermvorm vinden we door expansie, dat wil zeggen door introductie van de ontbrekende variabelen:

$$\begin{aligned} S &= \overline{x}\overline{y}\overline{z} + \overline{x}\overline{y}(\overline{z} + z) + x(\overline{y} + y)\overline{z} + (\overline{x} + x)yz \\ &= \overline{x}\overline{y}\overline{z} + \overline{x}\overline{y}z + x\overline{y}\overline{z} + x\overline{y}z + x\overline{y}\overline{z} + x\overline{y}z + \overline{x}yz + xyz \\ &= \overline{x}\overline{y}\overline{z} + \overline{x}\overline{y}z + x\overline{y}\overline{z} + x\overline{y}z + x\overline{y}\overline{z} + x\overline{y}z + \overline{x}yz + xyz \end{aligned} \quad (2.44)$$

De overige termen ontbreken doordat de corresponderende functiewaarden 0 zijn. De functiewaarden met waarde 1 zijn dus:

$$f_0, f_3, f_4, f_5, f_6, f_7$$

De functiewaarden f_i met de waarde $f_i = 0$ zijn:

$$f_1, f_2$$

Hieruit volgt direct de *maxtermvorm*:

$$\begin{aligned}
 S &= (\bar{m}_0 + f_0) \cdot (\bar{m}_1 + f_1) \cdot (\bar{m}_2 + f_2) \cdot (\bar{m}_3 + f_3) \\
 &\quad \cdot (\bar{m}_4 + f_4) \cdot (\bar{m}_5 + f_5) \cdot (\bar{m}_6 + f_6) \cdot (\bar{m}_7 + f_7) \\
 &= \bar{m}_1 \cdot \bar{m}_2 \\
 &= (x + y + \bar{z}) \cdot (x + \bar{y} + z)
 \end{aligned} \tag{2.45}$$

De eenvoudigste somvorm ontstaat als volgt uit de mintermvorm.

$$\begin{aligned}
 S &= \bar{x}\bar{y}\bar{z} + \bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z + x\bar{y}\bar{z} + x\bar{y}z \\
 &= (\bar{x}\bar{y}\bar{z} + x\bar{y}\bar{z}) + (\bar{x}yz + x\bar{y}z) + (x\bar{y}\bar{z} + x\bar{y}z + x\bar{y}\bar{z} + x\bar{y}z) \\
 &= (x + \bar{x})\bar{y}\bar{z} + (\bar{x} + x)yz + x(\bar{y}\bar{z} + \bar{y}z + y\bar{z} + yz) \\
 &= \bar{y}\bar{z} + yz + x
 \end{aligned} \tag{2.46}$$

De eenvoudigste produktvorm van de gegeven functie is (toevallig) gelijk aan de maxtermvorm. $\square$

Notatie

Voor de mintermvorm en de maxtermvorm van een functie in k variabelen gelden de volgende formules:

$$\text{mintermvorm: } S = \sum_{i=0}^{2^k-1} m_i \cdot f_i \tag{2.47}$$

$$\text{maxtermvorm: } S = \prod_{i=0}^{2^k-1} (M_i + f_i) \tag{2.48}$$

Zijn de functiewaarden f_i bekend, dan vallen in de mintermvorm alle produkttermen weg waarvoor de functiewaarde $f_i = 0$ is. In de maxtermvorm vallen alle factoren met $f_i = 1$ weg en worden door 1 vervangen. We kunnen de mintermvorm en de maxtermvorm daarom direct afleiden uit de waarheidstabel. Voor de in tabel 2.14 gespecificeerde functie leidt dit tot

$$S = \sum_{i=0}^{2^3-1} m_i \cdot f_i = m_3 + m_4 + m_5 + m_6 + m_7 \tag{2.49.a}$$

en

$$S = \prod_{i=0}^{2^3-1} (M_i + f_i) = M_0 \cdot M_1 \cdot M_2 \tag{2.50.a}$$

Beide formules worden in een verkorte notatie ook wel geschreven als

$$S = \sum(3,4,5,6,7) \tag{2.49.b}$$

en

$$S = \prod(0,1,2) \tag{2.50.b} \quad \square$$

2.7 Logische bewerkingen op functies

De logische operatoren EN, OF en NIET zijn gedefinieerd op proposities en op de ervan afgeleide logische variabelen. Een functie S is ook te beschouwen als een logische variabele (paragraaf 2.4). Immers, voor elke ingangscombinatie resulteert de evaluatie van de functie in de functiewaarde $S = 0$ of in de functiewaarde $S = 1$. Logische bewerkingen zijn dus ook gedefinieerd op of tussen logische functies. Voor de uitvoering van logische bewerkingen op functies leiden we hieronder een belangrijke conclusie af.

Hulpstelling 2.3 Negatiestelling

Voor de negatie $\bar{S}$ van een logische functie S in n variabelen,

$$S = \sum_{i=0}^{2^n-1} m_i \cdot f_i$$

geldt:

$$\bar{S} = \sum_{i=0}^{2^n-1} m_i \cdot \bar{f}_i \quad (2.51)$$

Bewijs.

Uitgaande van de mintermvorm volgt

$$\begin{aligned} \bar{S} &= \overline{m_0 \cdot f_0 + m_1 \cdot f_1 + m_2 \cdot f_2 + \dots + m_{2^n-1} \cdot f_{2^n-1}} \\ &= (\bar{m}_0 + \bar{f}_0)(\bar{m}_1 + \bar{f}_1)(\bar{m}_2 + \bar{f}_2)(\dots)(\bar{m}_{2^n-1} + \bar{f}_{2^n-1}) \\ &= (M_0 + \bar{f}_0)(M_1 + \bar{f}_1)(M_2 + \bar{f}_2)(\dots)(M_{2^n-1} + \bar{f}_{2^n-1}) \end{aligned} \quad (2.52)$$

Dit is de maxtermvorm (2.48) van de functie met functiewaarden $\bar{f}_i$, zodat $\bar{S}$ ook is

$$\bar{S} = m_0 \cdot \bar{f}_0 + m_1 \cdot \bar{f}_1 + m_2 \cdot \bar{f}_2 + \dots + m_{2^n-1} \cdot \bar{f}_{2^n-1}$$

zijnde de mintermvorm van de functie met functiewaarden $\bar{f}_0$ tot en met $\bar{f}_{2^n-1}$. $\square$

Voorbeeld

Zij S een logische functie in drie variabelen, waarvan de mintermvorm is

$$S = \bar{x}\bar{y}\bar{z} \cdot f_0 + \bar{x}\bar{y}z \cdot f_1 + \bar{x}y\bar{z} \cdot f_2 + \bar{x}yz \cdot f_3 + x\bar{y}\bar{z} \cdot f_4 + x\bar{y}z \cdot f_5 + xy\bar{z} \cdot f_6 + xyz \cdot f_7$$

Dan is

$$\begin{aligned} \bar{S} &= \overline{\bar{x}\bar{y}\bar{z} \cdot f_0 + \bar{x}\bar{y}z \cdot f_1 + \bar{x}y\bar{z} \cdot f_2 + \bar{x}yz \cdot f_3 + x\bar{y}\bar{z} \cdot f_4 + x\bar{y}z \cdot f_5 + xy\bar{z} \cdot f_6 + xyz \cdot f_7} \\ &= (x + y + z + \bar{f}_0) \cdot (x + y + \bar{z} + \bar{f}_1) \cdot (x + \bar{y} + z + \bar{f}_2) \cdot (\dots) \cdot (\bar{x} + \bar{y} + \bar{z} + \bar{f}_7) \end{aligned} \quad (2.53)$$

Omzetten naar de mintermvorm leidt tot

$$\bar{S} = \bar{x}\bar{y}\bar{z} \cdot \bar{f}_0 + \bar{x}\bar{y}z \cdot \bar{f}_1 + \bar{x}y\bar{z} \cdot \bar{f}_2 + \bar{x}yz \cdot \bar{f}_3 + x\bar{y}\bar{z} \cdot \bar{f}_4 + x\bar{y}z \cdot \bar{f}_5 + xy\bar{z} \cdot \bar{f}_6 + xyz \cdot \bar{f}_7$$

$$S = \sum_{i=0}^{2^n-1} m_i \cdot \bar{f}_i \quad (2.54)$$

De negatie van een functie S komt neer op het complementeren van alle functiewaarden f_i afzonderlijk ($f_i = 0/1 \rightarrow \bar{f}_i = 1/0$). De bewerking 'negatie' op de functie kan worden uitgevoerd door de functiewaarden afzonderlijk te negeren. (Eigenlijk is deze conclusie triviaal.) Zie ook [Hong, 1972]. $\square$

Hoe verloopt een en ander bij de logische operatoren OF en EN op functies?

Hulpstelling 2.4 Somstelling

Zijn

$$S_1 = \sum_{i=0}^{2^n-1} m_i \cdot f_{i1}$$

en

$$S_2 = \sum_{i=0}^{2^n-1} m_i \cdot f_{i2}$$

twee logische functies in n variabelen, dan geldt

$$S_1 + S_2 = \sum_{i=0}^{2^n-1} m_i \cdot (f_{i1} + f_{i2}) \quad (2.55)$$

Bewijs

Het bewijs volgt direct uit de mintermvorm (2.31/32) van beide functies en de commutatieve wet (2.17). $\square$

Hulpstelling 2.5 Produktstelling

Zijn

$$S_1 = \sum_{i=0}^{2^n-1} m_i \cdot f_{i1}$$

en

$$S_2 = \sum_{i=0}^{2^n-1} m_i \cdot f_{i2}$$

twee logische functies in n variabelen, dan geldt

$$S_1 \cdot S_2 = \sum_{i=0}^{2^n-1} m_i \cdot (f_{i1} \cdot f_{i2}) \quad (2.56)$$

Bewijs

Het bewijs volgt direct uit de mintermvorm van beide functies. Het produkt van twee mintermen m_i en m_j is 0,

$$m_i \cdot m_j = 0 \text{ voor } i \neq j$$

en m_i voor $i = j$. Hiermee is de rest van het bewijs triviaal. $\square$

Uit de hulpstellingen 2.3 t/m 2.5 en uit het feit dat elke samengestelde logische bewerking (bijvoorbeeld de EXOR bewerking) is uit te drukken in de logische operatoren EN, OF en NIET volgt stelling 2.6.

Stelling 2.6

Een willekeurige logische bewerking op logische functies in n variabelen kan teruggebracht worden tot een herhaalde uitvoering van deze bewerking op de afzonderlijke functiewaarden met hetzelfde onderlinge rangnummer.

Bewijs: Na het voorafgaande triviaal. $\square$

Een gevolg van deze stelling is bijvoorbeeld dat

$$S_1 \oplus S_2 = \sum_{i=0}^{2^n-1} m_i \cdot (f_{i1} \oplus f_{i2}) \quad (2.57)$$

De EXOR operator (exclusieve OF) mag op de afzonderlijke functiewaarden met hetzelfde rangnummer i worden toegepast.

In het volgende hoofdstuk, dat gaat over het representeren en reduceren van logische functies, zal van vorenstaande stellingen een dankbaar gebruik gemaakt worden.

2.8 Decompositie van functies

Grotere functies kunnen met behulp van decompositie in kleinere delen gesplitst worden. Decompositie is daarom een hulpmiddel bij het afbeelden van logische functies op hardware. Er zijn verschillende manieren bekend om een functie te splitsen. We behandelen de *Shannondecompositie*.

Stelling 2.7

Een logische functie S in n variabelen x_1 t/m x_n ,

$$S = f(x_1, x_2, \dots, x_{n-1}, x_n)$$

kan worden geschreven als

$$\begin{aligned} S &= f(x_1, x_2, \dots, x_i, \dots, x_{n-1}, x_n) \\ &= \bar{x}_i \cdot f(x_1, x_2, \dots, 0, \dots, x_{n-1}, x_n) + x_i \cdot f(x_1, x_2, \dots, 1, \dots, x_{n-1}, x_n) \end{aligned} \quad (2.58)$$

waarbij de *subfuncties* of *residuen*

$$f(x_1, x_2, \dots, 0, \dots, x_{n-1}, x_n) \quad (2.59.a)$$

$$f(x_1, x_2, \dots, 1, \dots, x_{n-1}, x_n) \quad (2.59.b)$$

uit S ontstaan door in S voor x_i de waarde 0 resp. 1 in te vullen (en voor $\bar{x}_i$ de waarde 1 resp. 0). De residuen zijn functies in $n-1$ variabelen.

Bewijs.

Voor drie variabelen verloopt het bewijs, gebaseerd op de mintermvorm, als volgt.

$$\begin{aligned}
 S &= \bar{x}_1 \bar{x}_2 \bar{x}_3 \cdot f_0 + \bar{x}_1 \bar{x}_2 x_3 \cdot f_1 + \cdots + x_1 x_2 \bar{x}_3 \cdot f_6 + x_1 x_2 x_3 \cdot f_7 \\
 &= \bar{x}_1 \cdot (\bar{x}_2 \bar{x}_3 \cdot f_0 + \bar{x}_2 x_3 \cdot f_1 + x_2 \bar{x}_3 \cdot f_2 + x_2 x_3 \cdot f_3) \\
 &\quad + x_1 \cdot (\bar{x}_2 \bar{x}_3 \cdot f_4 + \bar{x}_2 x_3 \cdot f_5 + x_2 \bar{x}_3 \cdot f_6 + x_2 x_3 \cdot f_7) \\
 &= \bar{x}_1 \cdot f(0, x_2, x_3) + x_1 \cdot f(1, x_2, x_3)
 \end{aligned} \tag{2.58.a}$$

(Op grond van de commutatieve wet mag decompositie naar elke variabele x_i worden uitgevoerd.) $\square$

Notatie

Subfuncties of residuen worden in de literatuur wel aangeduid als

$$f_{|x_i=b} \text{ met } b \in \{0, 1\} \quad \square$$

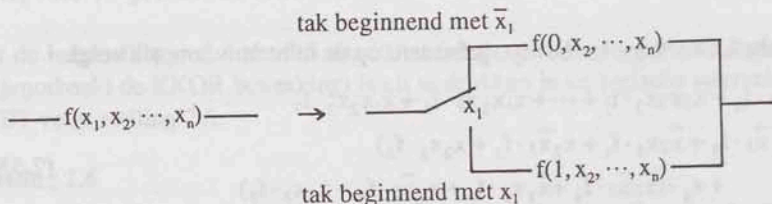
Tabel 2.15 geeft aan hoe uit de waarheidstabel van een functie in drie variabelen de waarheidstabellen van de subfuncties bij decompositie naar de variabele x volgen.

x	y	z	S		y	z	S ₁
0	0	0	f ₀		0	0	f ₀
0	0	1	f ₁		0	1	f ₁
0	1	0	f ₂	f(0, y, z)	1	0	f ₂
0	1	1	f ₃		1	1	f ₃
⋮							
1	0	0	f ₄		y	z	S ₂
1	0	1	f ₅		0	0	f ₄
1	1	0	f ₆	f(1, y, z)	0	1	f ₅
1	1	1	f ₇		1	0	f ₆
					1	1	f ₇

Tabel 2.15. Decompositie van de waarheidstabel.

Shannon [Shannon, 1938 en 1939] heeft decompositie ingevoerd om kontaktschakelingen met behulp van *wisselkontakten* te ontwerpen. Figuur 2.8 geeft het principe aan. (Een wisselkontakt is een kontakt, dat bestaat uit een beweegbaar middenkontakt en twee niet-

beweegbare vaste kontakten. Met een wisselkontakt kan gekozen worden uit één-van-twee signaalwegen.)



Figuur 2.8. Decompositie naar de variabele x_1 met een wisselkontakt.

Voorbeeld

Decompositie behoeft niet noodzakelijk op de mintermvorm van de functie gebaseerd te worden. Passen we bijvoorbeeld decompositie naar de variabele y toe op de logische functie

$$S = wx + x\bar{y}z + yz$$

dan vinden we

$$S = \bar{y} \cdot (wx + xz) + y \cdot (wx + z)$$

De term wx komt dan in beide subtermen voor. Verklaar dit. □

Herhaalde decompositie

Verschillende ontwerpmethoden voor logische schakelingen berusten op herhaalde decompositie. Voor logische functies in drie variabelen x t/m z bijvoorbeeld leidt herhaalde decompositie via

$$\begin{aligned} S &= f(x, y, z) \\ &= \bar{x} \cdot f(0, y, z) + x \cdot f(1, y, z) \\ &= \bar{x} \cdot \{\bar{y} \cdot f(0, 0, z) + y \cdot f(0, 1, z)\} + x \cdot \{\bar{y} \cdot f(1, 0, z) + y \cdot f(1, 1, z)\} \end{aligned}$$

tot

$$\begin{aligned} S &= \bar{x}\bar{y}\bar{z} \cdot f(0,0,0) + \bar{x}\bar{y}z \cdot f(0,0,1) + \bar{x}y\bar{z} \cdot f(0,1,0) + \bar{x}yz \cdot f(0,1,1) \\ &\quad + x\bar{y}\bar{z} \cdot f(1,0,0) + x\bar{y}z \cdot f(1,0,1) + xy\bar{z} \cdot f(1,1,0) + xyz \cdot f(1,1,1) \end{aligned} \quad (2.60)$$

Hierin zijn $f(0,0,0)$ t/m $f(1,1,1)$ constanten met de logische waarde 0 of 1. Zij corresponderen met de functiewaarden f_0 t/m f_7 :

$$f(0,0,0) = f_0$$

$$f(0,0,1) = f_1$$

⋮

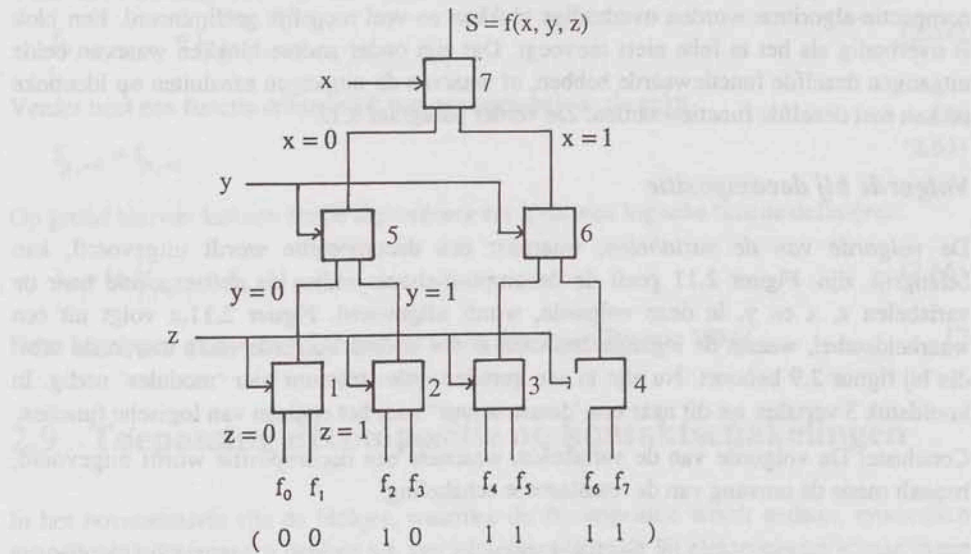
$$f(1,1,1) = f_7$$

(2.61)

Voor de indices, zie paragraaf 2.5.

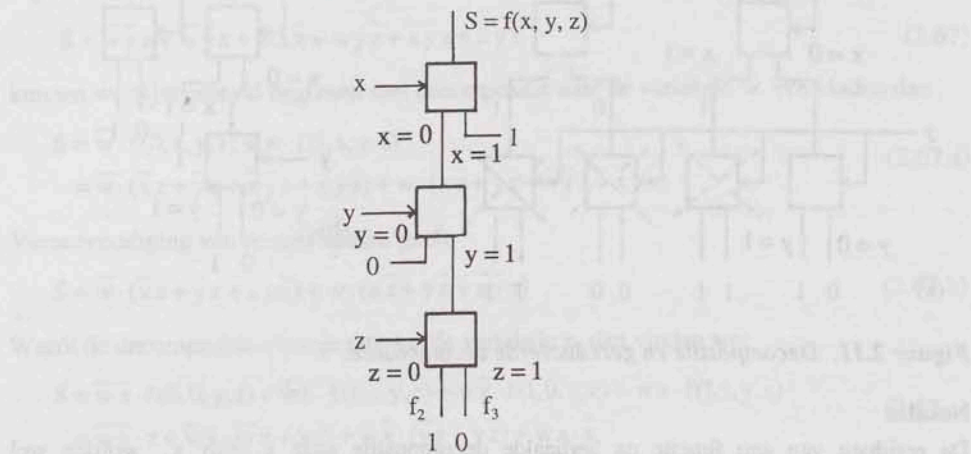
Voorbeeld

Figuur 2.9 geeft in 'black box' vorm aan waartoe voor een functie $S = f(x, y, z)$ decompositie naar de drie variabelen leidt. In drie 'beslisniveaus' wordt de bij de momentele ingangscombinatie behorende functiewaarde gevonden.



Figuur 2.9. Decompositie naar drie ingangsvariabelen.

Vullen we ook de functiewaarden in (tussen haakjes toegevoegd in de figuur), dan zie we dat de blokjes 1 en ook 3, 4 en 6 niets toevoegen en in feite overbodig zijn. Deze blokjes geven een vaste logische waarde door. Hierdoor reduceert figuur 2.9 tot figuur 2.10. $\square$



Figuur 2.10. Gereduceerde structuur na decompositie.

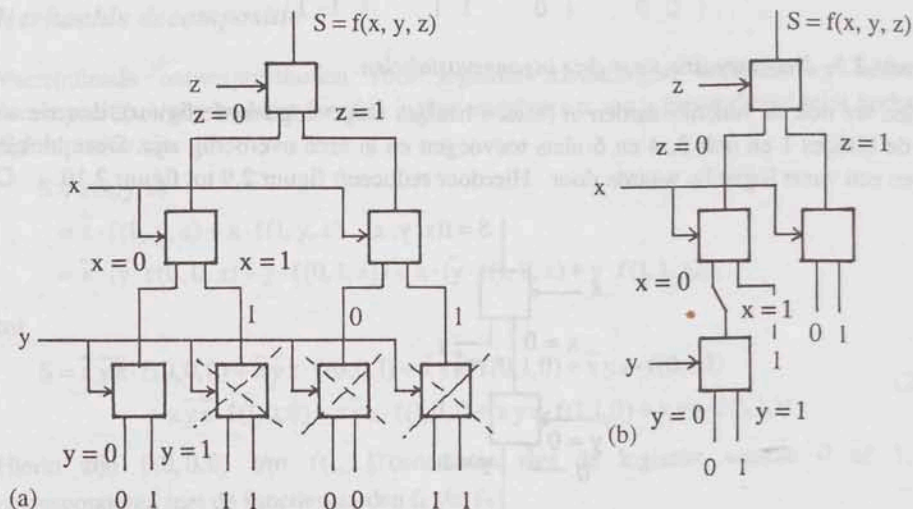
Een structuur zoals in figuur 2.9 wordt een *binary decision tree* [Knuth, 1969] genoemd (BDT), ook wel *binary decision graph*. Staan de variabelen daarbij in een vaste volgorde, dat wil zeggen op elk 'niveau' dezelfde variabele (dat hoeft namelijk niet), dan spreken we over een geordende of *ordered BDT* (OBDT).

Wanneer op een OBDT een compactie-algoritme is losgelaten, zie figuur 2.10, dan worden de diagrammen meestal aangeduid als *binary decision diagram*, ofwel BDD. Met een compactie-algoritme worden overbodige blokken zo veel mogelijk geëlimineerd. Een blok is overbodig als het in feite niets toevoegt. Dat zijn onder andere blokken waarvan beide uitgangen dezelfde functiewaarde hebben, of waarvan de uitgangen aansluiten op identieke takken met dezelfde functiewaarden. Zie verder paragraaf 3.12.

Volgorde bij decompositie

De volgorde van de variabelen, waarnaar een decompositie wordt uitgevoerd, kan belangrijk zijn. Figuur 2.11 geeft de decompositieboom indien de decompositie naar de variabelen z , x en y , in deze volgorde, wordt uitgevoerd. Figuur 2.11.a volgt uit een waarheidstabel, waarin de ingangsvariabelen in een andere volgorde staan dan in de tabel die bij figuur 2.9 behoort. Nu zijn in een gereduceerde structuur vier 'modules' nodig. In hoofdstuk 3 vertalen we dit naar een 'datastructuur' voor het opslaan van logische functies.

Conclusie: De volgorde van de variabelen, waarnaar een decompositie wordt uitgevoerd, bepaalt mede de omvang van de resulterende schakeling.



Figuur 2.11. Decompositie en gereduceerde decompositie.

Notatie

De residuen van een functie na herhaalde decompositie naar x_i t/m x_n worden wel aangeduid als

$$f_{|x_i=b_i, x_j=b_j, \dots, x_m=b_m, x_n=b_n} \quad \text{met } b \in \{0, 1\} \text{ en } x \in \{i, j, \dots, n\} \quad (2.62)$$

en recursief geïnterpreteerd als

$$f_{|x_i=b_i, x_j=b_j, \dots, x_m=b_m, x_n=b_n} = \{f_{|x_i=b_i, x_j=b_j, \dots, x_m=b_m}\}_{|x_n=b_n} \quad (2.63)$$

Voor de bewerking 'decompositie' geldt de *commutatieve wet van de decompositie*:

$$f_{|x_i=b_i, x_j=b_j} = f_{|x_j=b_j, x_i=b_i} \quad (2.64)$$

Verder heet een functie *afhankelijk van een variabele* x_i als geldt

$$f_{|x_i=0} \neq f_{|x_i=1} \quad (2.65)$$

Op grond hiervan kunnen we de *dependence set* I_f van een logische functie definiëren:

$$I_f = \{x_i \mid f_{|x_i=0} \neq f_{|x_i=1}\} \quad (2.66)$$

Deze begrippen en notatiewijze komen uit de literatuur [Bryant, 1986]. □

2.9 Toepassing decompositie op kontaktschakelingen

In het bovenstaande zijn de blokjes, waarmee de decompositie wordt gedaan, symbolisch aangeduid. Bij kontakten denken we aan *wisselkontakten* en bij elektronische schakelingen aan een *selector*. Een voorbeeld van het ontwerp van een kontaktschakeling geven we hieronder. Een voorbeeld met selectoren behandelen we in hoofdstuk 6.

Voorbeeld

Bij de realisatie van de functie S ($S = 1$ indien het netwerk geleidt)

$$S = \bar{w}xz + \bar{w}\bar{y}z + wx\bar{z} + wy\bar{z} + \bar{x}\bar{y}z + xy\bar{z} \quad (2.67)$$

kunnen we bijvoorbeeld beginnen met decompositie naar de variabele w . We vinden dan

$$\begin{aligned} S &= \bar{w} \cdot f(0, x, y, z) + w \cdot f(1, x, y, z) \\ &= \bar{w} \cdot (\bar{x}z + \bar{y}z + \bar{x}\bar{y}z + xy\bar{z}) + w \cdot (\bar{x}\bar{z} + y\bar{z} + \bar{x}\bar{y}z + xy\bar{z}) \end{aligned} \quad (2.67.a)$$

Vereenvoudiging van de subfuncties geeft

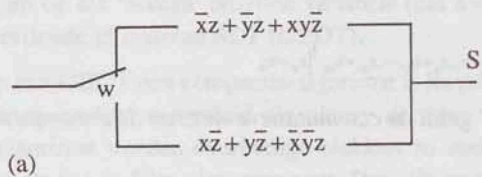
$$S = \bar{w} \cdot (\bar{x}z + \bar{y}z + xy\bar{z}) + w \cdot (\bar{x}\bar{z} + y\bar{z} + \bar{x}\bar{y}z) \quad (2.67.b)$$

Wordt de decompositie voortgezet naar de variabele x , dan vinden we:

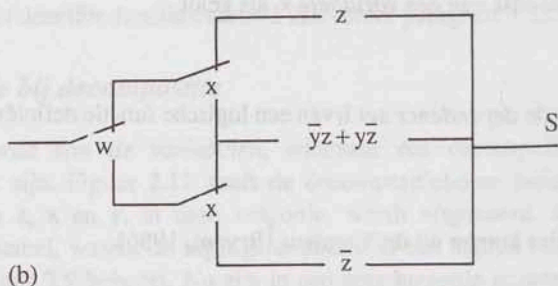
$$\begin{aligned} S &= \bar{w}\bar{x} \cdot f(0, 0, y, z) + \bar{w}x \cdot f(0, 1, y, z) + w\bar{x} \cdot f(1, 0, y, z) + wx \cdot f(1, 1, y, z) \\ &= \bar{w}\bar{x} \cdot z + \bar{w}x \cdot (\bar{y}z + y\bar{z}) + w\bar{x} \cdot (\bar{y}z + y\bar{z}) + wx \cdot \bar{z} \end{aligned} \quad (2.67.c)$$

Hierin zijn de subfuncties $f(0, 1, y, z)$ en $f(1, 0, y, z)$ gelijk. We combineren deze takken:

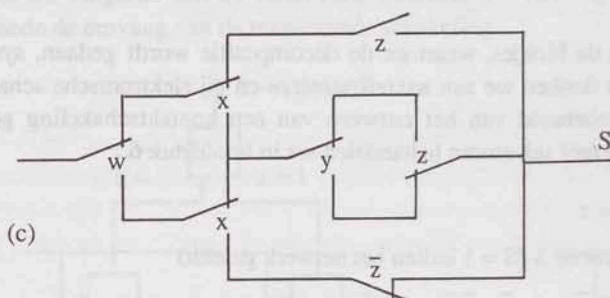
$$S = \bar{w}\bar{x} \cdot z + (\bar{w}x + w\bar{x}) \cdot (\bar{y}z + y\bar{z}) + wx \cdot \bar{z} \quad (2.67.d)$$



$$S = \bar{w} \cdot (\bar{x}z + \bar{y}z + xy\bar{z}) + w \cdot (x\bar{z} + y\bar{z} + \bar{x}y\bar{z})$$



$$S = \bar{w}\bar{x} \cdot z + \bar{w}x \cdot (\bar{y}z + y\bar{z}) + w\bar{x} \cdot (\bar{y}z + y\bar{z}) + wx \cdot \bar{z}$$



$$S = \bar{w}\bar{x} \cdot z + (\bar{w}x + w\bar{x}) \cdot (\bar{y}z + y\bar{z}) + wx \cdot \bar{z}$$

Figuur 2.12. Realisatie met decompositie naar variabelen.

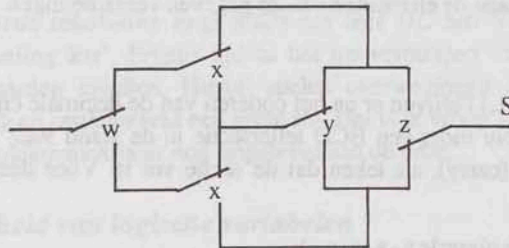
Er resteert nu nog één tak met de subfunctie

$$f = \bar{y}z + y\bar{z}$$

Deze kan gemakkelijk verder worden gedecomposeerd naar y of z. Figuur 2.12 geeft de verschillende stappen van de decompositie van deze functie en de realisatie met (wissel)kontakten weer. Het zal de lezer daarna niet veel moeite kosten uit figuur 2.12.c de schakeling van figuur 2.13 te herkennen. $\square$

Het resultaat hangt bij deze methode sterk af van de volgorde van de variabelen, waarnaar de decompositie van de functie wordt uitgevoerd. Voor functies in veel variabelen is het

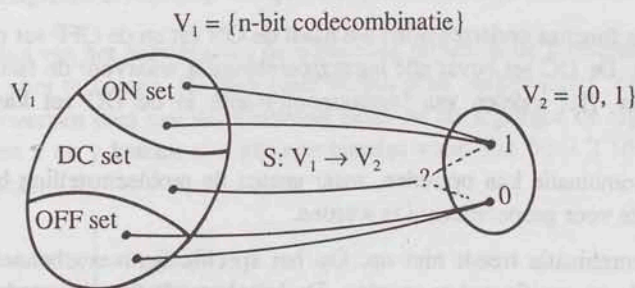
zinnig eerst te splitsen naar variabelen, waarbij zo veel mogelijk *identieke subfuncties* (residuen) optreden. De vraag is of en hoe je dat van tevoren kunt zien.



Figuur 2.13. Resultaat realisatie van de functie S .

2.10 De ON set, OFF set en DC set van logische functies

Het domein van een logische functie S in n variabelen bestaat uit de 2^n combinaties van de waarden van de n variabelen. De functie S beeldt een aantal combinaties op de functiewaarde 1 af. We noemen dit deel van het domein de *ON Set* van de logische functie. De *OFF Set* van de functie bestaat uit de codecombinaties die op 0 afgebeeld worden. Bij praktische voorbeelden is vaak voor een gedeelte van het domein geen functiewaarde gespecificeerd, bijvoorbeeld omdat de betreffende combinaties nooit optreden of omdat de waarde van de functie ervoor niet belangrijk is. De functiewaarde voor een ingangscombinatie uit dit deel van het domein is dan een don't care, dat wil zeggen vrij te kiezen. We noemen dit deel van het domein de *DC Set*. Zie figuur 2.14.



Figuur 2.14. De ON set, OFF set en DC set van een logische functie.

Een logische functie, waarvan de ON set en de OFF set samen de hele verzameling ingangscombinaties vormen, heet *volledig gespecificeerd*. Is de DC set niet-leeg, dan heet de functie *onvolledig gespecificeerd* (*incompletely specified*).

Bij het coderen van de elementen van een gegeven verzameling met behulp van binaire/logische variabelen stuit men vaak op het probleem dat de te coderen verzameling

een aantal elementen ongelijk aan 2^n heeft. Bij het coderen blijft dan een aantal combinaties over. Zo kan het voorkomen dat op een hoog ontwerpniveau een functie compleet gespecificeerd is, maar op een lager ontwerpniveau niet meer en wel na het toewijzen van de binaire codecombinaties aan de elementen van de gegeven verzamelingen.

Voorbeeld

In de *BCD code* (paragraaf 1.1) blijven er na het coderen van de decimale cijfers zes van de 16 codecombinaties over. Nu moet een BCD tellersectie in de stand $9_{\text{DEC}} = 1001_{\text{BCD}}$ een transportsignaal genereren (carry), als teken dat de sectie vol is. Voor deze carry functie geldt:

$$\text{ON set: } \{1001\} \quad (\text{volgorde } a_3 a_2 a_1 a_0) \quad (2.68.a)$$

$$\text{OFF set: } \{0000, 0001, \dots, 1000\} \quad (2.68.b)$$

$$\text{DC set: } \{1010, 1011, \dots, 1111\} \quad (2.68.c)$$

Voor de combinaties uit de DC set mogen we de waarde van de carry vrij kiezen. Noemen we de variabelen in de BCD code a_0 (gewicht 2^0) tot en met a_3 (gewicht 2^3), dan is de carry functie op basis van de ON set:

$$S_{\text{carry}} = a_3 \bar{a}_2 \bar{a}_1 a_0 \quad (2.69.a)$$

en op basis van de ON set zo eenvoudig mogelijk met een vrije keuze voor de DC set

$$S_{\text{carry}} = a_3 a_0 \quad \square \quad (2.69.b)$$

Voorbeeld

Geheel anders is de situatie als we een schakeling moeten ontwerpen, die een alarm afgeeft als een BCD tellersectie in een van de niet-optredende combinaties komt. Dan zijn alle functiewaarden gespecificeerd en is de DC set leeg. $\square$

Bij veel logische functies onderscheiden we naast de ON set en de OFF set ook een DC set (*don't care set*). De DC set bevat alle ingangscombinaties waarvoor de functiewaarde niet gespecificeerd is. Het indelen van ingangscombinaties in de DC set kan verschillende redenen hebben:

- De ingangscombinatie kan optreden, maar gezien de probleemstelling hoeft er geen functiewaarde voor gespecificeerd te worden.
- De ingangscombinatie treedt niet op. Op het specificatieniveau hoeft er dan geen functiewaarde gespecificeerd te worden. De bijbehorende functiewaarde is altijd *don't care*.
- Bij een bepaalde opeenvolging van ingangscombinaties treden overgangsverschuiven op. De functiewaarde is dan met zekere beperkingen te kiezen, nemen we bijvoorbeeld nog de oude of reeds de nieuwe waarde. Hier is dus geen sprake van een geheel vrije keuze. We noemen dit *conditionele don't cares*.

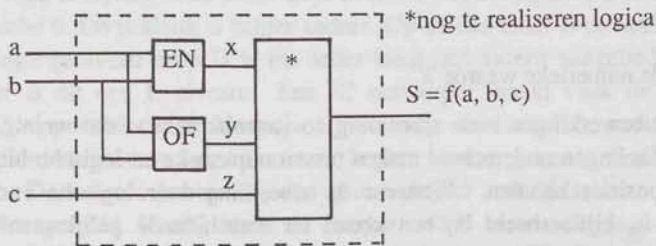
Bij dit alles speelt het momentele *ontwerpniveau*, waarop we op een bepaald moment bezig zijn, een belangrijke rol. We hebben reeds gezien dat voor een niet-gecodeerde

ingangverzameling (hoog ontwerpniveau) een complete input-output specificatie kan bestaan, terwijl voor de gecodeerde versie (lager ontwerpniveau) de specificatie incompleet is en wel vanwege overbodige codecombinaties in de BCD code.

Een gerealiseerde schakeling heeft altijd een lege DC set. Voor elke ingangscombinatie 'doet de schakeling iets'. Ergens tijdens het ontwerptraject moet de ontwerper de don't care functiewaarden invullen. Hierbij spelen overwegingen als *logische minimalisatie*, *foutafhandeling* en *testbaarheid* een grote rol. Het is te vroeg hier dieper op in te gaan. We stellen deze problematiek later nog uitgebreid aan de orde.

Afhankelijkheid van logische variabelen

Het komt vaak voor dat logische variabelen ten opzichte van elkaar een zekere afhankelijkheid vertonen. Figuur 2.15 geeft een voorbeeld van een situatie waarbij dit ontstaat. Een te ontwerpen logische schakeling heeftingangssignalen a t/m c. Hiermee worden acht ingangscombinaties gevormd. Voor elk ervan is de functiewaarde van de uitgangsfunctie S gespecificeerd. De functie $S = f(a, b, c)$ is compleet gespecificeerd.



Figuur 2.15. Logische schakeling met afhankelijke variabelen.

Bij de uitwerking van het ontwerp van de schakeling blijken in de formules de termen $a \cdot b$ en $a + b$ vaak voor te komen. De ontwerper besluit deze 'hulpfuncties' eerst te realiseren. Het nog te ontwerpen deel van de schakeling heeft nu als ingangen de signalen x, y en z. Van de signalen x en y komen niet alle combinaties voor. Zie tabel 2.16. De combinatie $xy = 10$ ontbreekt.

a	b	x	y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Tabel 2.16. Afhankelijkheid van logische variabelen.

We noemen logische variabelen die ten opzichte van elkaar een beperkt (logisch) waardebereik aannemen, *afhankelijke logische variabelen*. Dit kan het geval zijn voor *externe* of *primaire ingangssignalen*, maar ook bij *interne* of *secundaire ingangssignalen* in een schakeling. Dit verschijnsel vormt weer een bron van *don't care condities*. Bij de realisatie van combinatorische logische schakelingen komen we op het probleem van afhankelijkheid van logische variabelen uitgebreid terug. Zie de paragrafen 4.7 en verder.

2.11 De keuze van proposities en logische variabelen

Zoals we proposities hiervoor geïntroduceerd hebben definiëren zij de (interpretatie van de) logische variabelen. Hoe komen we aan geschikte proposities?

Vaak ligt de keuze voor de hand. Bij het realiseren van schakelingen voor rekenkundige bewerkingen in het binaire talstelsel beelden we numerieke bits direct af op logische bits. Formeel gaat dit via proposities van de vorm

'Bit a_i heeft de numerieke waarde 1'

of

'Bit a_i heeft de numerieke waarde 2'

Bij rekenkundige bewerkingen is de afbeelding zo vanzelfsprekend dat weinig ontwerpers van logische schakelingen onderscheid maken tussen numerieke en logische bits, laat staan dat zij hier proposities bijhalen. Wanneer de afbeelding naar logische variabelen niet vanzelfsprekend is, bijvoorbeeld bij een keuze uit verschillende gelijkwaardige alternatieven, dan zijn proposities een noodzakelijke tussenstap.

Voorbeeld

Gegeven is een netwerk, opgebouwd uit mechanische schakelaars. Geven we een geleidend netwerk aan met $S=0$ of met $S=1$? Of in geval van twee spanningsniveaus, correspondeert H met 1 en L met 0? Hierover moeten afspraken gemaakt worden. Zo'n afspraak is formeel een propositie. $\square$

De afspraak is *proposities bevestigend te formuleren*. Maar wat is 'bevestigend formuleren' bij een mechanische schakelaar, die in twee standen kan staan? In dat geval moeten de standen eerst een 'naam' krijgen, bijvoorbeeld *ruststand* en *werkstand*. Vervolgens kan de propositie zijn

'De schakelaar staat in de werkstand'

Een onhandige introductie van proposities kan veel ongemak veroorzaken. Nemen we als voorbeeld een codeschijf (paragraaf 1.6). Proposities kiezen als

'Segment i staat bovenaan'

leidt tot net zoveel proposities als er segmenten op de codeschijf zitten. Bovendien kan er van deze proposities slechts één tegelijk waar zijn, hetgeen leidt tot *afhankelijke logische*

variabelen (paragraaf 2.10). Een ervaren ontwerper doet automatisch de goede keuze. Welke is dat in dit geval?

Ondubbelzinnigheid van proposities

Proposities dienen duidelijk te zijn. Nemen we als voorbeeld het probleem hoe de toestand van een beweegbare brug aan te duiden. Een propositie als

'De brug is open'

is bepaald niet ondubbelzinnig te interpreteren. Voor welke categorie verkeer is de brug 'open', voor het rijdend verkeer of voor een schip? En wat bedoelt men met 'open'. Is de brug bij 'open' wel of niet toegankelijk? Een omschrijving als

'De brug is toegankelijk voor het rijdend verkeer'

moge dan wel langer zijn, maar is zeker duidelijker!

Daarnaast is het belangrijk zo veel mogelijk met *vaste afspraken* te werken. Een hoog spanningsniveau H bijvoorbeeld komt altijd overeen met de logische 1 en een laag niveau L met de logische 0. De praktijk is echter anders. Op de ene teller is de laadingang LD actief indien het ingangsniveau erop H is (de teller laadt dan extern aangeboden data), bij een andere teller is dit een L niveau. Een IC ontwerper maakt vaak de keuze voor een minimaal IC oppervlak (lage kostprijs bij verkoop van het IC), maar beseft niet dat daardoor op ontwerpniveau van apparatuur hogere kosten gemaakt worden vanwege H-L vergissingen. De auteurs hebben bij een groot project met studenten een tijd lang alle ontwerpfouten geturfd. Hierbij bleek de categorie H-L verwisselingen/vergissingen meer dan 50% van alle fouten te zijn! Wat is dus wijsheid in dit geval?

De stap van probleem naar logische specificatie via proposities is belangrijk. Er wordt nagedacht over wat nu precies het probleem is waarvoor de specificatie opgesteld moet worden. Hoe meer duidelijkheid in deze fase van een ontwerp geschapen wordt, des te gemakkelijker verlopen alle volgende fasen. Bij modulaire ontwerpen, waarbij verschillende ontwerpers elk een stuk voor hun rekening nemen, is een ondubbelzinnige specificatie van de interactie tussen de modules (de *interface specificatie*) een noodzaak. De afspraken aan 'weerszijden' moeten consistent zijn.

Vaak slaat men de stap via expliciet geformuleerde proposities over en tracht men direct een som-van-produktenvorm voor de logische formules te formuleren. Daarbij worden vaak en veel details over het hoofd gezien.

Voorbeeld

Uit vier objecten moeten er *precies twee* geselecteerd worden. Hoe specificeer je dit in een logische formule? Hiervoor zijn twee stappen nodig:

- Hoe specificeer je de proposities?
- Wat is dan de formule?

Proposities als

A: 'Element/object a is geselecteerd'

:

leggen de keuze van elk object vast. Wanneer we nu aan elke propositie een variabele toekennen,

A ~ variabele a

:

dan kan de formule opgesteld worden. Is de formule

$$S = ab + ac + ad + bc + bd + cd \quad (2.70)$$

dan correct of niet? ($S = 1$ bij keuze van precies twee objecten.)

(Hint: Expandeer de formule naar de mintermvorm.) □

Veel aspecten zijn in het vorenstaande nog niet genoemd. Zo specificceert men vaak alleen wat het ontwerp moet kunnen, zonder stil te staan bij wat een ontwerp *niet moet kunnen*. Wat te doen in het geval van *bedieningsfouten*? Is een *storingsafhandeling* voorzien? In de volgende hoofdstukken zullen we hier dieper op ingaan. Moge het voorlopig duidelijk zijn dat de schakelalgebra een nuttig hulpmiddel is bij het ontwerpen van logische schakelingen, maar niet meer dan dat.

2.12 Voorbeeld opstellen van formules

Het volgende voorbeeld beschrijft een type probleem dat in de digitale techniek veel voorkomt, zij het dat we dan geen boodschappen gaan doen. Het is als zodanig een aardige oefening in het opstellen/kiezen van proposities en het uitwerken van logische formules.

Voorbeeld

Iemand moet boodschappen doen en heeft op zijn lijstje staan:

- boter (t)
- suiker (u)
- medicijnen (v)
- drop (w)
- koekjes (x)
- brood (y)
- vleeswaren (z)

Tabel 2.17 geeft een overzicht van waar de diverse artikelen verkocht worden.

		t	u	v	w	x	y	z	
kruidenier	A	•	•		•	•	•	•	
apotheek	B			•	•				• verkrijgbaar
zuivelhandel	C	•			•	•		•	
bakker	D					•	•		
slager	E							•	

Tabel 2.17. Overzicht verkrijgbaarheid van boodschappen.

Hoe moet deze persoon handelen om al zijn boodschappen te doen? Om dit probleem formeel te kunnen beschrijven en om het vervolgens te kunnen optimaliseren moet overgegaan worden op een logische specificatie van het probleem. Zij nu S de propositie

S: 'Boodschappenlijstje afgewerkt'

De propositie S is slechts waar als alle boodschappen gekocht zijn. Om boter te kunnen kopen moet bijvoorbeeld een bezoek gebracht worden aan de kruidenier of aan de zuivelhandel. Tenminste één van de proposities

A: 'Kruidenier bezocht'

of

C: 'Zuivelhandel bezocht'

moet waar zijn. Zie tabel 2.17. In een formule:

$$S_{\text{boter}} = A + C \quad (2.71.a)$$

waarbij S_{boter} vastlegt dat er boter gekocht is (of beter: kan worden). Evenzo moet suiker gekocht worden. Volgens tabel 2.17 is

$$S_{\text{sui ker}} = A \quad (2.71.b)$$

Dan is

$$S_{\text{boter EN sui ker}} = (A + C) \cdot A \quad (2.71.c)$$

Alle boodschappen kunnen worden gekocht als

$$\begin{aligned} S_{\text{boter EN sui ker EN ... EN vleeswaren}} \\ = (A + C) \cdot A \cdot B \cdot (A + B + C) \cdot (A + C + D) \cdot (A + D) \cdot (A + C + E) \end{aligned} \quad (2.72)$$

waar is. Hiermee is een logische formule voor het probleem opgesteld.

Uit de schakelalgebra volgt onder andere dat

$$(A + C) \cdot A = A + A \cdot C = A$$

is. Passen we deze vereenvoudigingsregel nog een paar keer toe, dan vinden we

$$S_{\text{boter EN sui ker EN ... EN vleeswaren}} = A \cdot B \quad (2.73)$$

Hiermee is het gestelde probleem uitgewerkt/gereduceerd. In twee winkels kan het lijstje worden afgewerkt. $\square$

Opmerking

Formule (2.71) sluit niet uit dat er meer winkels bezocht worden. Zo mogen we voor

$$S_{\text{boter EN sui ker EN} \dots \text{EN vleeswaren}} = A \cdot B$$

ook schrijven

$$S_{\text{boter EN sui ker EN} \dots \text{EN vleeswaren}} = A \cdot B \cdot (\bar{C} + C) \cdot (\bar{D} + D) \cdot (\bar{E} + E) \quad (2.73.a)$$

waaruit een veelvoud van mogelijke (maar niet altijd minimale) oplossingen kan komen.

Bijvoorbeeld

$$S_{\text{boter EN sui ker EN} \dots \text{EN vleeswaren}} = A \cdot B \cdot C \cdot \bar{D} \cdot E \quad (2.73.b)$$

Om precies te zijn, er zijn acht combinaties van winkelbezoek mogelijk, die aan de gestelde eis dat het lijstje afgewerkt moet kunnen worden voldoen. Een ervan, formule (2.73), is de minimale oplossing. (Welk *criterium* wordt voor 'minimaal' gehanteerd?) $\square$

Binnen korte tijd zal de in dit voorbeeld beschreven problematiek voltooid verleden tijd zijn. De plaatselijke supermarkten verkopen dan alles. De enige reden om meer winkels te gaan bezoeken is dan de prijs van de produkten.

Voorbeeld

Wanneer we het goedkoopst boodschappen willen doen, hoe luidt dan de formulering? Hiervoor is een tussenstap nodig en wel onderzoek naar de kostprijs van elk produkt. Dit onderzoek resulteert in tabel 2.18, waarin de goedkoopste mogelijkheid met een vierkantje is aangegeven. (Bij gelijke prijs is er meer dan een leverancier aangegeven.)

		t	u	v	w	x	y	z
kruidenier	A	•	■		■	■	■	■
apotheek	B			■	■			
zuivelhandel	C	■			■	•		•
bakker	D					•	■	
slager	E							•

Tabel 2.18. Overzicht verkrijgbaarheid voor de laatste prijs.

Nu luidt de formule

$$S_{\text{boter EN sui ker EN} \dots \text{EN vleeswaren/laagste prijs}} = C \cdot A \cdot B \cdot (A + B + C) \cdot A \cdot (A + D) \cdot A \quad (2.74)$$

Na vereenvoudiging

$$S_{\text{boter EN sui ker EN} \dots \text{EN vleeswaren/laagste prijs}} = A \cdot B \cdot C \quad \square \quad (2.75)$$

Het hierboven beschreven type probleem komt in de digitale techniek veel voor. Er is dan sprake van een te bedekken verzameling objecten V_1 . Elk object uit V_1 kan door een of meer elementen van een andere verzameling V_2 'bedekt' worden. Hoeveel elementen uit V_2 zijn minimaal nodig om tot een volledige bedekking van V_1 te komen? En hoeveel zijn dat

er als een kostenfunctie meegenomen wordt? We zullen dan de hiervoor gehanteerde benadering vaak toepassen. Zie onder andere paragraaf 3.10. Het idee om het probleem op deze manier te stellen is afkomstig van [Petrick, 1956].

Literatuur

1. G. Boole, *George Boole's Collected Logical Works, Vol. 2, The Laws of Thought*, Open Court, Lasalle, 1952.
2. G. Birkhoff and T. C. Bartee, *Modern Applied Algebra*, McGraw-Hill, New York, 1970.
3. G. Boole, *An Investigation of the Laws of Thought*, London, 1854.
4. R. E. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Trans. on Computers, Vol. C-35, 1986, pp. 677-691.
5. Ph. Dwinger, *Introduction to Boolean Algebras*, Physica Verlag, Würzburg.
6. F. E. Hohn, *Applied Boolean Algebra*, Macmillan, New York, 1966.
7. S. J. Hong and D. L. Ostapko, *On Complementation of Boolean Functions*, IEEE Trans. on Computers, Vol. C-21, 1972, pp. 1022.
8. D. E. Knuth, *Fundamental Algorithms, The Art of Computer Programming*, Vol. 1, Addison-Wesley, Reading, Mass., 1969.
9. E. J. McCluskey, *Logic Design Principles*, Prentice-Hall, Englewood Cliffs, 1986.
10. S. R. Petrick, *A Direct Determination of the Irredundant Form of a Boolean Function from the Set of Prime Implicants*, Computation Laboratory of the AFCRC, Bedford, Mass. TR-56-110, April 1956.
11. C. E. Shannon, *A Symbolic Analysis of Relay and Switching Circuits*, Trans. Am. Inst. Electr. Eng., Vol. 57, 1938, pp. 713-723.
12. C. E. Shannon, *The Synthesis of Two-Terminal Switching Circuits*, Bell Syst. Tech. J., Vol. 28, 1949, pp. 59-98.
13. J. E. Whitesitt, *Boolean Algebra and Its Applications*, Addison-Wesley, Reading, Mass., 1961.

Opgaven hoofdstuk 2

2.1. Over proposities worden vier beweringen gedaan. Een ervan is niet van toepassing. Welke is dat?

- a. Proposities beschrijven de dynamische toestand van een component op het logische abstractieniveau.
- b. Proposities worden zoveel mogelijk bevestigend geformuleerd.
- c. Een propositie is waar of niet waar.
- d. Een logische variabele kan gedefinieerd worden met een propositie.

2.2. De logische werking van een transistor is de werking in

- a. de statische toestand (geen veranderingen).
- b. de dynamische toestand.
- c. een systeem met een 5 Volt voedingsspanning.
- d. een toepassing waarin twee duidelijk verschillende werkingstoestanden worden onderscheiden.

2.3. De volgorde van de operatoren, volgens welke een logische formule moet worden geëvalueerd, is:

- a. EN-OF-NIET, tenzij haakjes anders aangeven (linker operator eerst).
- b. NIET-EN-OF, tenzij haakjes anders aangeven.
- c. NIET-OF-EN, tenzij haakjes anders aangeven.
- d. EN-OF-NIET, tenzij haakjes anders aangeven.

2.4. De volgorde van de termen in een som of van de factoren in een produkt mag men naar behoefte wijzigen op grond van de

- associatieve wetten.
- commutatieve wetten.
- distributieve wetten.
- moduswetten.

2.5. Van de volgende vier gelijkheden is er één fout. Welke?

- $x + y \cdot z = (x + y) \cdot (x + z)$.
- $(x + y) \cdot (x + z) \cdot (y + z) = x \cdot y + x \cdot z + y \cdot z$.
- $\overline{x + y + z} = \overline{x} + \overline{y} + \overline{z}$.
- $x \cdot y + \overline{x} \cdot z + y \cdot z = x \cdot y + \overline{x} \cdot z$.

2.6. Van de volgende vier regels stelt er één geen schakelwet voor. Welke?

- $x + y \cdot z = (x + y) \cdot (x + z)$.
- $(x + y) + z = x + (y + z)$.
- $y \cdot (\overline{y} + z) = y \cdot z$.
- $x \cdot y + \overline{x} \cdot z + y \cdot z = x \cdot y + y \cdot z$.

2.7. Van de volgende drie gelijkheden

$$\overline{x + y + z} = \overline{x} \cdot \overline{y} \cdot \overline{z}$$

$$(x + y) \cdot (\overline{x} + z) \cdot (\overline{y} + \overline{z}) = (x + y) \cdot (\overline{x} + z)$$

$$(x + y) \cdot (x + z) \cdot (y + z) = x \cdot y + x \cdot z + y \cdot z$$

is/zijn er

- nul goed.
- één goed.
- twee goed.
- drie goed.

2.8. De mintermvorm van een logische functie is een som-van-produktenform waarvoor geldt:

- Alle variabelen of hun genegeerde komen in elk produkt één maal voor.
- Elke term heeft een minimaal aantal variabelen, vanwaar de naam min(imale)term komt.
- Het aantal termen ervan is minimaal.
- In elke produktterm is het aantal variabelen minimaal.

2.9. Een functie S in drie variabelen x, y en z wordt als volgt gespecificeerd:

$$f_0 = f_4 = f_6 = f_7 = 0$$

$$f_1 = f_2 = f_3 = f_5 = 1$$

(De indices i van de funktiewaarden f_i zijn gebaseerd op de volgende toekenning van gewichten: $x \sim 4$, $y \sim 2$ en $z \sim 1$.) De mintermvorm van deze functie is:

- $S = \overline{x}y + \overline{y}z$.
- $S = \overline{x}\overline{y}\overline{z} + x\overline{y}\overline{z} + x\overline{y}z + xyz$.
- $S = \overline{y}z + xy$.

d. $S = \bar{x}\bar{y}z + \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}z$.

2.10. In de produktterm

$$u\bar{v}\bar{w}xyz$$

mag $\bar{w}$ vervangen worden door

- a. $\bar{w} \rightarrow \bar{w} + x$
- b. $\bar{w} \rightarrow \bar{w} + \bar{x}$
- c. $\bar{w} \rightarrow \bar{w} + x + y + z$
- d. $\bar{w} \rightarrow u + \bar{v} + x + y + z$

2.11. Welke term(en) is/zijn *overbodig* in de functie

$$S = \bar{b}\bar{c} + a\bar{c} + a\bar{b} + \bar{b}\bar{c}$$

- a. $\bar{b}\bar{c}$.
- b. $a\bar{b}$ en $a\bar{c}$.
- c. $a\bar{b}$ of $a\bar{c}$.
- d. Geen enkele term is overbodig.

2.12. Gegeven is de functie

$$S = (x + y)(x + \bar{y} + \bar{z})(\bar{y} + z) + yz$$

Indien S wordt ingevuld in een waarheidstabel in de variabelen x , y en z , dan bevat de functiewaardenkolom precies

- a. twee nullen.
- b. drie nullen.
- c. vier nullen.
- d. vijf nullen.

2.13. De maxtermvorm van een logische functie is een produkt-van-sommenvorm waarvoor geldt:

- a. Het aantal factoren in het produkt is maximaal, vandaar de naam.
- b. Elke factor is een som, waarin alle variabelen van de functie hoogstens éénmaal voorkomen (gewoon of genegeerd).
- c. Elke factor is een produkt, waarin alle variabelen van de functie éénmaal voorkomen (gewoon of genegeerd).
- d. Elke factor is een som van alle variabelen, elke variabele komt in elke factor éénmaal voor (gewoon of genegeerd) en ook één van de functiewaarden. Wanneer een functiewaarde 1 is, dan mag de factor door 1 vervangen worden, welke 1 meestal weggelaten wordt.

2.14. Een functie S in drie variabelen x , y en z wordt gespecificeerd door middel van de acht functiewaarden f_i , met $0 \leq i \leq 7$. (De indices zijn gebaseerd op de toekenning van gewichten volgens $x \sim 4$, $y \sim 2$ en $z \sim 1$.) Precies drie van de functiewaarden hebben de waarde 1. Hoeveel maxtermen bevat de maxtermvorm van S ? (Hierbij worden factoren welke altijd de waarde 1 hebben niet meegeteld.)

- a. Drie.
- b. Vijf.

- c. Acht.
- d. Met deze gegevens: onbepaald.

2.15. Gegeven is de logische functie S in drie variabelen

$$S = x + yz$$

De mintermvorm van functie S bevat k_1 termen (elke term $\neq 0$) en de maxtermvorm k_2 factoren (elke factor $\neq 1$.) Dan is/zijn

- a. $k_1 = 5$ en $k_2 = 3$.
- b. $k_1 = 2$.
- c. $k_2 = 2$.
- d. Geen van de vorige antwoorden is goed.

2.16. Gegeven is een logische functie S in de mintermvorm,

$$S = \sum_{i=0}^{2^n-1} m_i \cdot f_i$$

Voor de functie $\bar{S}$ geldt:

a. $\bar{S} = \sum_{i=0}^{2^n-1} m_i \cdot \bar{f}_i$

b. $\bar{S} = \sum_{i=0}^{2^n-1} \bar{m}_i \cdot \bar{f}_i$

c. $\bar{S} = \sum_{i=0}^{2^n-1} \bar{m}_i \cdot f_i$

- d. $\bar{S}$ heeft een andere formule.

2.17. Twee logische functies S_1 en S_2 , welke zijn uitgedrukt in dezelfde variabelen, zijn gegeven in de mintermvorm. Hieruit moet de mintermvorm van de functie S

$$S = S_1 \cdot S_2 = \sum_{i=0}^{2^n-1} m_i \cdot f_i^*$$

bepaald worden. Hoeveel produkten van functiewaarden van de gegeven functies moeten er voor het bepalen van *elke* functiewaarde f_i^* van S berekend worden?

- a. Eén.
- b. $2n$.
- c. $(2^n - 1)^2$
- d. Een ander dan de genoemde aantallen.

2.18. Gegeven zijn drie logische functies F_1 , F_2 en T . Over F_1 , F_2 en T worden drie uitspraken gedaan:

1. Uit $T + F_1 = T + F_2$ volgt: $F_1 = F_2$ voor alle waarden van de variabelen.
2. Uit $F_1 = F_2$ volgt: $T + F_1 = T + F_2$ voor alle waarden van de variabelen.
3. Uit $T \cdot F_1 = T \cdot F_2$ volgt: $F_1 = F_2$ voor alle waarden van de variabelen.

Van deze uitspraken is/zijn er

- a. nul goed.
- b. één goed.
- c. twee goed.
- d. drie goed.

2.19. Gegeven zijn twee functies S en T:

$$S = ab + \bar{a}c + \bar{a}b$$

$$T = ac + \bar{a}\bar{b} + b\bar{c}$$

De functie V is de EN functie van S en T. De eenvoudigste formule voor V luidt:

- a. $V = a\bar{b}\bar{c} + \bar{a}\bar{c} + \bar{a}b\bar{c}$.
- b. $V = ab + b\bar{c} + \bar{a}b\bar{c}$.
- c. $V = \bar{a}b\bar{c} + ac + \bar{a}b\bar{c}$.
- d. $V = ab + b\bar{c} + \bar{a}c$.

2.20. Gegeven is de logische functie

$$S = f(w, x, y, z) = \bar{w}xy + yz + x\bar{y}z$$

Bij de decompositie van S naar de variabele w is de eenvoudigste som-van-produktenvorm van de subfunctie $f(0, x, y, z) = f_{|w=0}$

- a. $f_{|w=0} = yz + x\bar{y}z$.
- b. $f_{|w=0} = xy$.
- c. $f_{|w=0} = xy + yz + x\bar{y}z$.
- d. De subfunctie $f(0, x, y, z) = f_{|w=0}$ heeft een andere eenvoudigste formule.

2.21. Gegeven is de functie S,

$$S = xz + yz + \bar{w}xy$$

Stel een binary decision tree op voor een decompositie van de functie S, met als volgorde van de variabelen w, x, y en z (w eerst). Indien hierin zoveel mogelijk paden gecombineerd worden en takken die tot dezelfde uitgangswaarde(n) leiden worden weggelaten, dan resulteert een minimum aantal blokjes n.

- a. $n = 3$.
- b. $n = 4$.
- c. $n = 5$.
- d. $n = 6$.

2.22. Gegeven is de logische functie S,

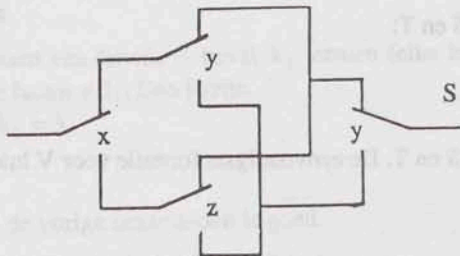
$$S = x + y\bar{z}$$

Voor functie S wordt een binary decision tree opgesteld. Hierna wordt deze BDT zoveel mogelijk gereduceerd, waarbij overbodige 'blokken' worden weggelaten. Bij het opstellen van een BDT kan de volgorde van de variabelen vrij gekozen worden. De *minst eenvoudige* gereduceerde BDT voor deze functie wordt gevonden bij de volgorde van de variabelen.

- a. x/y/z (eerst naar x, dan naar y, dan naar z).

- b. $y/x/z$.
- c. $x/z/y$.
- d. $z/y/x$.

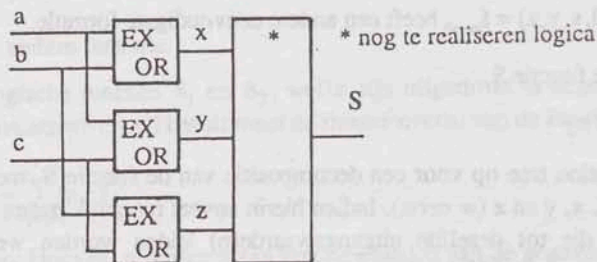
2.23.



Plaats een 1 in een waarheidstabel voor drie variabelen als het getekende kontaktennetwerk voor de betreffende ingangscombinatie geleidt ($x = 0 \rightarrow$ kontakt x staat in de getekende stand; enz.). Dan staan er in de waarheidstabel

- a. vier enen.
- b. vijf enen.
- c. zes enen.
- d. een ander aantal enen.

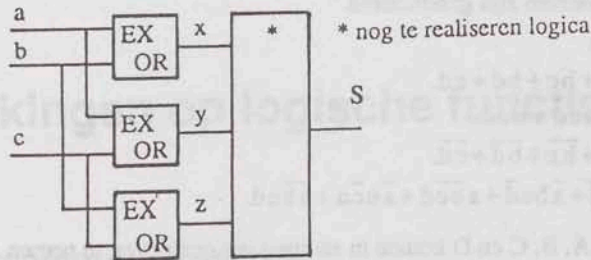
2.24. Een logische schakeling wordt in twee niveaus opgebouwd. Op het eerste niveau zit een drietal componenten, die EXOR bewerkingen uitvoeren op de ingangsvariabelen. Zie de figuur.



De schakeling op het tweede niveau met de ingangsvariabelen x , y en z heeft

- a. afhankelijke ingangsvariabelen. Slechts vier van de acht mogelijke combinaties van x , y en z komen voor.
- b. afhankelijke ingangsvariabelen. Slechts vijf van de acht mogelijke combinaties van x , y en z komen voor.
- c. afhankelijke ingangsvariabelen. Slechts zeven van de acht mogelijke combinaties van x , y en z komen voor.
- d. onafhankelijke ingangsvariabelen. Alle combinaties van x , y en z komen voor.

2.25. Een logische schakeling wordt in twee niveaus opgebouwd. Op het eerste niveau zit een drietal componenten, die EXOR bewerkingen uitvoeren op de ingangsvariabelen. Zie de figuur.



De schakeling op het tweede niveau met de ingangsvariabelen x, y en z heeft als gevolg van afhankelijkheid van zijn ingangsvariabelen een

- lege don't care (DC) set.
- DC set met twee ingangscombinaties.
- DC set met vier ingangscombinaties.
- Geen van de antwoorden is geheel juist.

2.26. De decimale cijfers worden in de BCD code binair gecodeerd met vier bits/variabelen. Een toepassing heeft een display voor twee BCD gecodeerde cijfers. Elke cijferunit wordt met een BCD code ingesteld. Er moet bewaakt worden dat het getal op het BCD display tot de BCD code behoort. Hiertoe wordt een detectieschakeling ontworpen, die geschikt is voor het totale display (twee decaden). De logische functie S ervan heeft de waarde 1 indien er op het display een verboden combinatie optreedt. De ON set van S bevat

- 6 ingangscombinaties.
- 12 ingangscombinaties.
- 100 ingangscombinaties.
- 156 ingangscombinaties.

2.27. Uit een verzameling elementen {a, b, c, d} worden er enkele geselecteerd. Propositions A t/m D leggen vast welke elementen geselecteerd zijn:

A: 'Element a is geselecteerd.'

enz. Er wordt een formule opgesteld voor het geval

'Tenminste twee van de vier elementen zijn geselecteerd.'

Welke formule is dit?

- $S = ab + ac + ad + bc + bd + cd.$
- $S = abc + abd + acd + bcd.$
- $S = \bar{a}\bar{b} + \bar{a}\bar{c} + \bar{a}\bar{d} + \bar{b}\bar{c} + \bar{b}\bar{d} + \bar{c}\bar{d}.$
- $S = ab\bar{c}\bar{d} + a\bar{b}c\bar{d} + \bar{a}b\bar{c}d + a\bar{b}\bar{c}d + \bar{a}b\bar{c}d + \bar{a}\bar{b}cd.$

2.28. Uit een verzameling elementen {a, b, c, d} worden er enkele geselecteerd. Propositions A t/m D leggen vast welke elementen geselecteerd zijn:

A: 'Element a is geselecteerd.'

enz. Er wordt een formule opgesteld voor het geval

'Precies twee elementen zijn geselecteerd.'

Welke formule is dit?

- $S = ab + ac + ad + bc + bd + cd.$
- $S = abc + abd + acd + bcd.$
- $S = \bar{a}\bar{b} + \bar{a}\bar{c} + \bar{a}\bar{d} + \bar{b}\bar{c} + \bar{b}\bar{d} + \bar{c}\bar{d}.$
- $S = abc\bar{d} + a\bar{b}c\bar{d} + \bar{a}bc\bar{d} + a\bar{b}c\bar{d} + \bar{a}bc\bar{d} + \bar{a}\bar{b}c\bar{d}.$

2.29. Vier personen A, B, C en D komen in aanmerking om zitting te nemen in het bestuur van een vereniging. Volgens de statuten moet de groepering waarvan A en B deel uitmaken de absolute meerderheid hebben in het bestuur. C en D maken geen deel uit van deze groepering. Gegeven zijn de proposities:

S: 'Het bestuur is overeenkomstig de statuten samengesteld.'

A: 'Persoon A is in het bestuur opgenomen.'

enz. De condities waaronder het bestuur statutair kan worden samengesteld, zijn geformuleerd in formule

- $S = (A + B) \cdot (\bar{C} \cdot \bar{D}) + A \cdot B \cdot (C + D).$
- $S = (A + B) \cdot (\bar{C} \cdot \bar{D}) + A \cdot B \cdot (\bar{C} \cdot \bar{D}).$
- $S = (A \oplus B) \cdot (\bar{C} + \bar{D}) + A \cdot B \cdot (C + D).$
- $S = (A \oplus B) \cdot (\bar{C} + \bar{D}) + A \cdot B \cdot (\bar{C} \cdot \bar{D}).$

2.30. Vijf personen A, B, C, D en E komen in aanmerking om zitting te nemen in het bestuur van een vereniging. A, B en C vormen groep 1; D en E vormen groep 2. Voor de samenstelling van het bestuur gelden twee eisen:

- Leden uit groep 1 moeten de absolute meerderheid hebben in het bestuur.
- Het bestuur moet uit tenminste drie leden bestaan.

Gegeven zijn de proposities:

S: 'Het bestuur is correct samengesteld.'

A: 'Persoon A is in het bestuur opgenomen.'

enz. De condities, waaronder het bestuur correct kan worden samengesteld, kunnen worden samengevat in de formule

- $S = ABC + (AB + BC + AC) \cdot (D + E).$
- $S = ABC + (AB + BC + AC) \cdot (D \oplus E).$
- $S = ABC + (A\bar{B}\bar{C} + \bar{A}BC + \bar{A}\bar{B}C) \cdot (D + E).$
- $S = ABC + (\bar{A} + \bar{B} + \bar{C}) \cdot (D \oplus E).$

Stelling 2

Indien het onderzoek van George Boole in 1854 op 'maatschappelijke relevantie' getoetst zou zijn, dan zou de hedendaagse wereld er waarschijnlijk geheel anders uitzien.

3

Bewerkingen op logische functies

3.1 Karnaughdiagrammen

In dit hoofdstuk houden we ons bezig met bewerkingen op logische functies. Deze bewerkingen kunnen tot doel hebben logische functies te vereenvoudigen, of om de functies in een zodanige vorm te brengen dat deze passen op de voor de realisatie van de functies beschikbare componenten. Het uitgangspunt is de waarheidstabel met de daarvan afgeleide mintermvorm van de functie. Hieruit wordt de (eenvoudigste) som-van-produktenvorm afgeleid, of de (eenvoudigste) produkt-van-sommenvorm. Met de regels van de schakelalgebra verloopt dit moeizaam. Daarom zijn in de loop der tijd enkele hulpmiddelen ontstaan om het bewerken van functies te ondersteunen. We behandelen hiervan het Karnaughdiagram, de Quine-McCluskey methode en het Binary Decision Diagram.

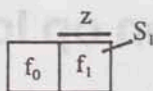
Het *Karnaughdiagram* is een *visueel hulpmiddel* om functies te vereenvoudigen. De *Quine-McCluskey* methode is een hiervan afgeleide meer formele versie, geschikt om in software te worden uitgevoerd. Een meer recente notatie van functies werkt met n -dimensionale kubussen en leidt tot een ruimtelijke voorstelling van logische functies.

BDD's (*binary decision diagrams*) tenslotte zijn *grafische voorstellingen* van logische functies. Met BDD's kan heel snel de functiewaarde geëvalueerd worden. Een hiervan afgeleide datastructuur maakt het mogelijk computerprogramma's voor het bewerken van logische functies te versnellen.

Waarheidstabellen en Karnaughdiagrammen

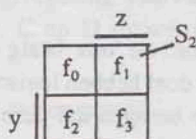
In figuur 3.1 zijn de waarheidstabellen gegeven van logische functies in één, twee, drie en vier variabelen. Met zo'n waarheidstabel correspondeert de mintermvorm van de bijbehorende logische functie (paragraaf 2.5). Naast elke waarheidstabel staat het bijbehorende Karnaughdiagram. Dit diagram specificeert dezelfde informatie als de waarheidstabel. Aan de randen van het diagram zijn *strepen* getrokken. Bij elke streep staat één van de variabelen waarin de functie is uitgedrukt. *Voor alle hokjes, die onder of boven respectievelijk links of rechts van zo'n streep liggen, wordt de bijgeschreven variabele gedacht de logische waarde 1 te hebben.* Voor hokjes welke niet 'onder' een streep vallen heeft de betreffende variabele de waarde 0. Strepen en variabelen zijn zo geplaatst dat met elk hokje één *uniek produkt van alle variabelen* correspondeert. Deze combinatie is de *minterm* die bij dit hokje behoort. In het betreffende hokje wordt de bij deze minterm behorende functiewaarde geplaatst [Veitch, 1952; Karnaugh, 1953]. Figuur 3.2 geeft aan hoe functiewaarden in een Karnaughdiagram geplaatst worden. Voor de functie S in twee variabelen

	z	S ₁
m ₀	0	f ₀
m ₁	1	f ₁



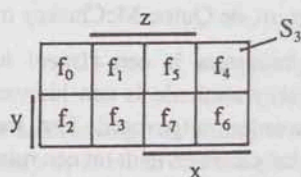
(a) diagram voor één variabele

	y	z	S ₂
m ₀	0	0	f ₀
m ₁	0	1	f ₁
m ₂	1	0	f ₂
m ₃	1	1	f ₃



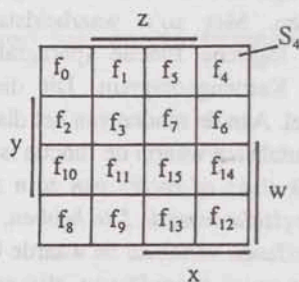
(b) diagram voor twee variabelen

	x	y	z	S ₃
m ₀	0	0	0	f ₀
m ₁	0	0	1	f ₁
m ₂	0	1	0	f ₂
m ₃	0	1	1	f ₃
m ₄	1	0	0	f ₄
m ₅	1	0	1	f ₅
m ₆	1	1	0	f ₆
m ₇	1	1	1	f ₇



(c) diagram voor drie variabelen

	w	x	y	z	S ₄
m ₀	0	0	0	0	f ₀
m ₁	0	0	0	1	f ₁
m ₁₄	1	1	1	0	f ₁₄
m ₁₅	1	1	1	1	f ₁₅

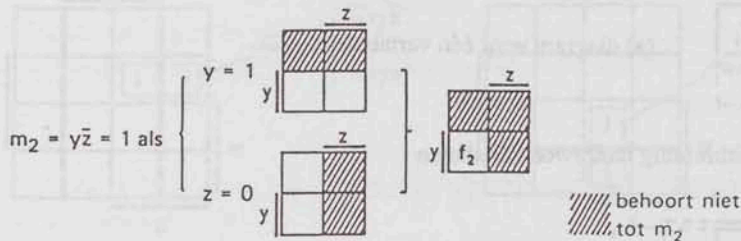


(d) diagram voor vier variabelen

Figuur 3.1. Waarheidstabellen en Karnaughdiagrammen.

$$\begin{aligned}
 S &= m_0 \cdot f_0 + m_1 \cdot f_1 + m_2 \cdot f_2 + m_3 \cdot f_3 \\
 &= \bar{y}\bar{z} \cdot f_0 + \bar{y}z \cdot f_1 + y\bar{z} \cdot f_2 + yz \cdot f_3
 \end{aligned}
 \tag{3.1}$$

is de plaatsing van de functiewaarde f_2 in deze figuur aangegeven. De plaatsing van de overige functiewaarden verloopt overeenkomstig.



Figuur 3.2. Het plaatsen van functiewaarden in een Karnaughdiagram.

De constructie van Karnaughdiagrammen

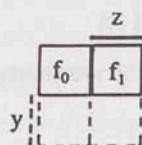
De randschriften bij Karnaughdiagrammen moeten zo geplaatst worden dat bij elk hokje een *unieke minterm* behoort. Dit kan op verschillende manieren. Wordt de opbouw via spiegelen uitgevoerd, dan blijken de diagrammen eigenschappen te hebben die gunstig zijn om functies te vereenvoudigen. De procedure voor de opbouw van Karnaughdiagrammen is als volgt.

Een Karnaughdiagram voor functies in n variabelen ontstaat uit een Karnaughdiagram voor functies in $n-1$ variabelen door het diagram voor $n-1$ variabelen over één van de randen te spiegelen. De oude randschriften worden meegespiegeld en aan het nieuwe stuk diagram wordt de volgende variabele als randschrift toegevoegd. Afwisselend wordt er naar beneden en naar rechts gespiegeld, waardoor een vierkant of rechthoekig diagram ontstaat. Op deze wijze geconstrueerd heeft een Karnaughdiagram voor n variabelen precies 2^n hokjes, met voor elk hokje één unieke combinatie van de randschriften. Zie figuur 3.3. (Merk op dat in figuur 3.3 de volgorde van de variabelen $z - y - x - w - v \dots$ is. Daarbij behoren de aangegeven functiewaarden. Bij een andere volgorde van de variabelen komen de functiewaarden op andere plaatsen te staan.)

De rangnummers i van de functiewaarden f_i zijn gebaseerd op de gewichten 2^0 (z) tot en met 2^3 (w) van de variabelen. Bij een andere volgorde van de variabelen en/of gewichten veranderen de rangnummers mee. In deze tekst hanteren we daarom een *vaste volgorde* van de variabelen in de randschriften.

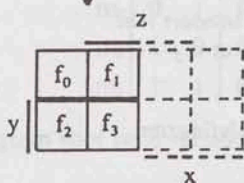
De constructie door middel van *spiegelen* brengt een voor het vereenvoudigen van logische functies prettige eigenschap in het Karnaughdiagram. Twee mintermen m_i en m_j met functiewaarden $m_i = m_j = 1$ kunnen dan en slechts dan samengenomen worden in één produktterm indien zij in één variabele verschillen. Twee van zulke mintermen heten *adjacent* (aangrenzend). Nu liggen in Karnaughdiagrammen de hokjes, welke

corresponderen met mintermen die in één variabele verschillen, symmetrisch ten opzichte van de spiegellijn die is geïntroduceerd bij uitbreiding van het diagram naar de betreffende variabele. Aan de hand van de spiegellijnen kunnen we dus nagaan in welke hokjes we moeten zoeken om adjacent of aangrenzende mintermen samen te nemen tot grotere produkttermen.



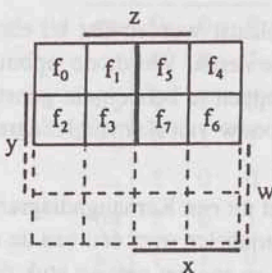
(a) diagram voor één variabele

↓ uitbreiding naar twee variabelen



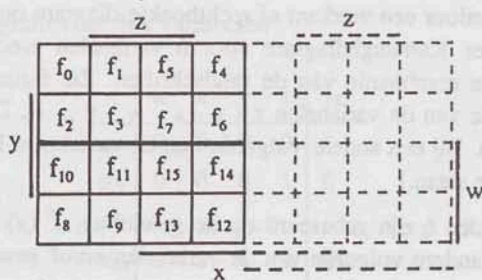
(b) diagram voor twee variabelen

↓ uitbreiding naar drie variabelen



(c) diagram voor drie variabelen

↓ uitbreiding naar vier variabelen



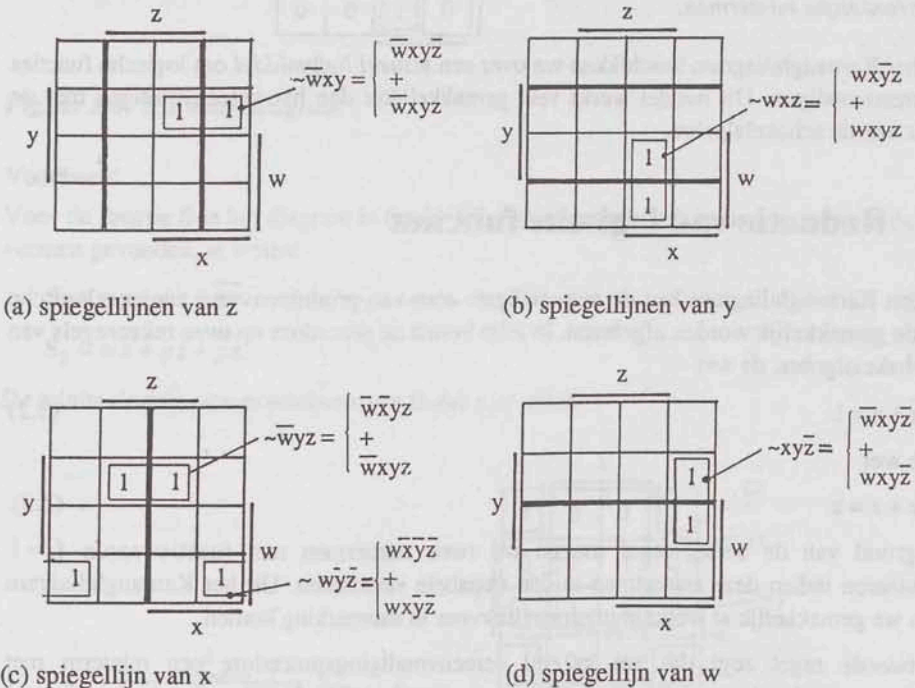
(d) diagram voor vier variabelen

↓ uitbreiding naar vijf variabelen

Figuur 3.3. Opbouw van Karnaughdiagrammen.

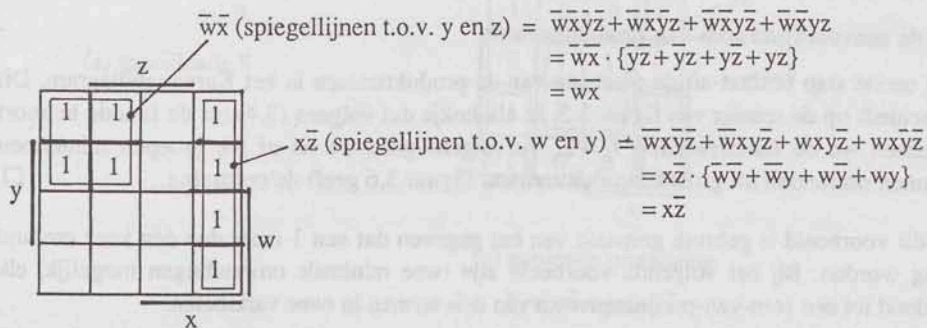
Figuur 3.4 toont de spiegellijnen in een Karnaughdiagram voor vier variabelen. We concluderen dat spiegellijnen in Karnaughdiagrammen hun oorsprong hebben in de eindpunten van de strepen bij de randschriften. Dit geldt voor elk Karnaughdiagram dat op

bovenstaande wijze geconstrueerd is. Tevens zijn in figuur 3.4 enkele mogelijkheden voor het combineren van mintermen aangegeven, waarvoor de functiewaarde $f_i = 1$ is. We zien dat in de resulterende produkttermen de 'spiegelvariabele' ontbreekt. Het door een produktterm aangeduide gebied geven we aan met een omranding van de ermee corresponderende enen.



Figuur 3.4. Spiegellijnen in Karnaughdiagrammen.

In figuur 3.5 liggen twee groepen van vier mintermen met $f_i = 1$ gespiegeld ten opzichte van twee spiegellijnen. De bijbehorende produkttermen, in twee variabelen minder, kunnen gemakkelijk herkend worden.



Figuur 3.5. Het samennemen van vier mintermen.

In het algemeen:

Groepen van 2^k mintermen, waarbij een functiewaarde $f_i = 1$ behoort, kunnen dan en slechts dan tot één produktterm samengenomen worden indien de bijbehorende hokjes symmetrisch liggen ten opzichte van de spiegellijnen van ten hoogste k verschillende variabelen. De hieruit volgende produktterm heeft k variabelen minder dan de oorspronkelijke mintermen.

Met het Karnaughdiagram beschikken we over een visueel hulpmiddel om logische functies te vereenvoudigen. Dit middel werkt veel gemakkelijker dan het vereenvoudigen met de regels van de schakelalgebra.

3.2 Reductie van logische functies

Uit een Karnaughdiagram kan de eenvoudigste som-van-produktenvorm van een logische functie gemakkelijk worden afgelezen. In feite berust de procedure op twee rekenregels van de schakelalgebra, de wet

$$\bar{z} + z = 1 \quad (3.2)$$

en de wet

$$z + z = z \quad (3.3)$$

Op grond van de eerste regel mogen we twee mintermen met functiewaarde $f_i = 1$ combineren indien deze mintermen in één variabele verschillen. Uit het Karnaughdiagram lezen we gemakkelijk af welke mintermen hiervoor in aanmerking komen.

De tweede regel zegt dat we in een vereenvoudigingsprocedure een minterm met functiewaarde $f_i = 1$ meer dan één keer mogen gebruiken. Uit de volgende voorbeelden zal het nut van deze regel blijken.

Voorbeeld

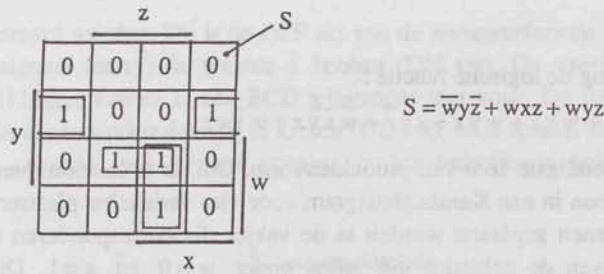
Vereenvoudig de functie

$$S = \bar{w}\bar{x}y\bar{z} + \bar{w}xy\bar{z} + w\bar{x}yz + wx\bar{y}z + wxyz \quad (3.4)$$

tot de eenvoudigste som-van-produktenvorm.

De eerste stap bestaat uit de plaatsing van de produkttermen in het Karnaughdiagram. Dit geschiedt op de manier van figuur 3.2. In elk hokje dat volgens (3.4) tot de functie behoort plaatsen we de functiewaarde $f_i = 1$. Vervolgens gaan we na of we groepen mintermen kunnen omranden tot grotere produkttermen. Figuur 3.6 geeft de oplossing. $\square$

In dit voorbeeld is gebruik gemaakt van het gegeven dat een 1 meer dan één keer omrand mag worden. Bij het volgende voorbeeld zijn twee minimale omrandingen mogelijk, elk leidend tot een som-van-produktenvorm van drie termen in twee variabelen.



Figuur 3.6. Karnaughdiagram.

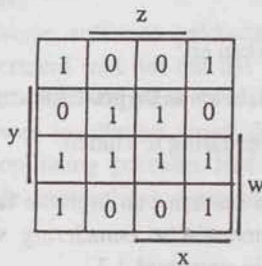
Voorbeeld

Voor de functie S in het diagram in figuur 3.7.a worden twee minimale som-van-produkten vormen gevonden, te weten:

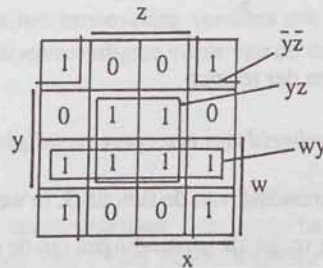
$$S_1 = wy + yz + \overline{y}\overline{z} \quad (3.5)$$

$$S_2 = w\overline{z} + yz + \overline{y}\overline{z} \quad (3.6)$$

De minimale som-van-produktenvorm is dus niet uniek. □

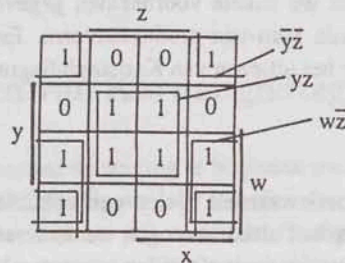


(a) specificatie S



(b) minimale omranding

$$S_1 = wy + yz + \overline{y}\overline{z}$$



(c) minimale omranding

$$S_2 = w\overline{z} + yz + \overline{y}\overline{z}$$

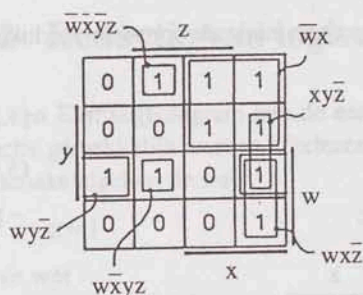
Figuur 3.7. Karnaughdiagram.

Voorbeeld

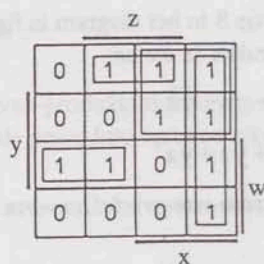
Vereenvoudig de logische functie S,

$$S = \bar{w}x + wx\bar{z} + wy\bar{z} + xy\bar{z} + \bar{w}\bar{x}\bar{y}z + w\bar{x}yz \quad (3.7)$$

tot de eenvoudigste som-van-produktenvorm. Om dit te kunnen doen moeten we eerst de produkttermen in een Karnaughdiagram voor vier variabelen plaatsen. Voor de term wx moeten er enen geplaatst worden in de vakjes die corresponderen met $\bar{w}$ en met x , de doorsnede van de gebieden die vallen onder $w=0$ en $x=1$. Dit zijn de vier vakjes rechtsboven. Op overeenkomstige wijze plaatsen we de overige functiewaarden. Zie voor het resultaat figuur 3.8.a.



(a) invullen der termen



(b) samennemen der termen

Figuur 3.8. Voorbeeld van een vereenvoudiging.

De *minimale omranding* van de functie S, te weten

- met zo weinig mogelijk omrandingen om de enen te bedekken en
- elke omranding zo groot mogelijk (= minimum aantal variabelen in de produkttermen)

staat in figuur 3.8.b. Het zal weinig moeite kosten om deze oplossing te vinden. $\square$

Hiermee hebben we enkele voorbeelden gegeven van het reduceren van logische functies tot een minimale som-van-produktenvorm. Een meer systematische benadering van de procedure voor het uitlezen van Karnaughdiagrammen staat in paragraaf 3.3.

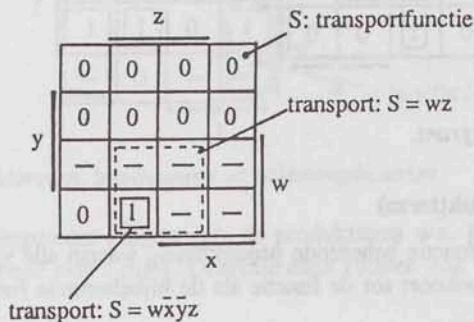
Don't cares

Don't care functiewaarden spelen een speciale rol bij het vereenvoudigen van logische functies. Al naar het uitkomt mogen we de waarde ervan als 0 of als 1 kiezen. Ook kunnen we een beslissing hierover uitstellen tot op een lager ontwerpniveau.

Voorbeeld

Figuur 3.9 specificeert de *transportfunctie* van een BCD tellersectie (paragraaf 2.10). Bij een BCD tellersectie moet in de stand 9 (1001_{BCD}) een transportsignaal (carry) gegenereerd worden naar de volgende sectie. Voor de standen 0 t/m 8 (0000_{BCD} t/m 1000_{BCD}) moet

geen transport gegenereerd worden. Dit is de OFF set van de transportfunctie. In de stand 9 moet het transportsignaal (carry) de waarde 1 hebben (ON set). De overige standen, 1010_{BCD} tot en met 1111_{BCD} , komen in een BCD tellersectie niet voor. De functiewaarde voor het transport is in deze standen dan vrij te kiezen (DC set), er is sprake van een don't care. Dit geven we aan met een liggend streepje in het betreffende hokje van het Karnaughdiagram.



Figuur 3.9. Transportfunctie van een BCD teller.

Uit figuur 3.9 lezen we gemakkelijk af dat het aanbeveling verdient om drie van de zes don't cares als 1 in te vullen en drie als 0. De eenvoudigste vorm van de transportfunctie is dan

$$S = wz \quad \square \quad (3.8)$$

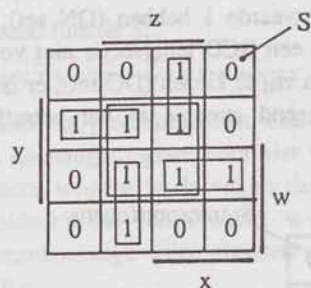
Attentie

Bij diverse software pakketten voor het vereenvoudigen van logische functies wordt geadverteerd met het feit dat het pakket ook don't care functiewaarden kan verwerken. Relatief vaak wordt dan voor alle don't cares of overal 1 ingevuld of overal 0. Voor elk van deze twee invullingen wordt de 'minimale oplossing' bepaald en uit deze twee wordt de beste oplossing gekozen. Het voorbeeld van de BCD tellersectie toont aan dat dan niet altijd de eenvoudigste formule gevonden wordt. $\square$

3.3 Het systematisch uitlezen van Karnaughdiagrammen

Bij het uitlezen van Karnaughdiagrammen bestaat de neiging te beginnen met de omranding van zo groot mogelijke gebieden. Het voorbeeld in figuur 3.10 laat zien dat, als men begint met de omranding van de binnenste vier enen (de term yz), er een overbodige term omrand wordt. De minimale som-van-produktenvorm bestaat in dit geval uit vier produkttermen, elk in drie variabelen.

Hoe moeten we het uitlezen van Karnaughdiagrammen dan aanpakken om tot een eenvoudigste som-van-produktenvorm te komen? Om deze vraag te kunnen beantwoorden moeten we eerst enkele begrippen introduceren.



Figuur 3.10. Karnaughdiagram.

Minterm (minimale produktterm)

Een minterm is een tot de functie behorende produktterm, waarin alle variabelen één keer voorkomen. Een minterm behoort tot de functie als de bijbehorende functiewaarde $f_i = 1$ is. Zie ook paragraaf 2.5.

Implicant (produktterm)

Een implicant is een tot de functie behorende produktterm in één of meer variabelen. Voor alle tot het produkt behorende mintermen staat een functiewaarde $f_i = 1$ in de ermee corresponderende hokjes van het Karnaughdiagram.

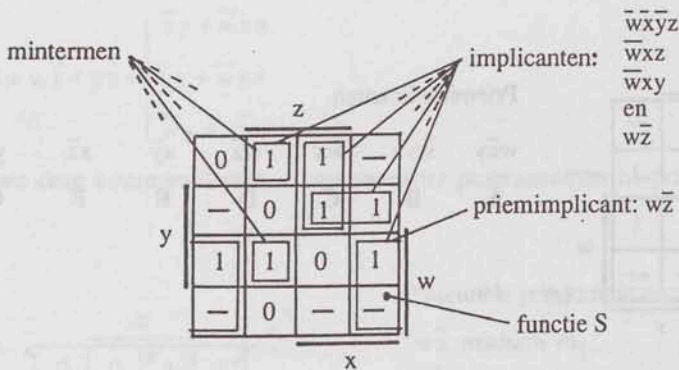
Prime implicant (produktterm behorend bij een maximale omranding)

Een priemimplicant (prime implicant) is een tot de functie behorende produktterm, die niet met andere produkttermen (implicanten) kan worden gecombineerd tot een kleinere produktterm met betrekking tot het aantal variabelen. Met een priemimplicant correspondeert een zo groot mogelijk omrand gebied in een Karnaughdiagram, dat symmetrisch ligt ten opzichte van een of meer der spiegellijnen.

Bij *onvolledig gespecificeerde functies* moeten deze begrippen uitgebreid worden tot enen en don't cares. Een minterm m_i met functiewaarde $f_i = 1$ kan gecombineerd worden met een minterm m_j met functiewaarde $f_j = \text{don't care}$ (op voorwaarde dat m_i en m_j in één variabele verschillen). Er ontstaat dan een implicant die één 1 en één don't care omvat. Het is daarom gebruikelijk bij het genereren van (priem)implicanten mintermen met don't care functiewaarde en implicanten die één of meer don't cares bevatten mee te nemen. Bij de verzameling der priemimplicanten laat men geheel uit don't cares bestaande priemimplicanten meestal weg. Zie ook paragraaf 3.9. Als toelichting een voorbeeld.

Voorbeeld

In figuur 3.11 zijn van een functie S twee *mintermen* aangegeven, de mintermen $\overline{w}xyz$ en $w\overline{x}yz$. (In totaal bestaat de ON set van de functie uit zeven mintermen.) Tevens zijn in figuur 3.11 vier *implicanten* aangegeven, de produkttermen $\overline{w}xyz$, $\overline{w}xy$, $\overline{w}xz$ en wz . (Uiteraard zijn er veel meer implicanten te omranden.)



Figuur 3.11. Mintermen, implicanten en priemimplicanten.

Ook is één priemimplicant aangegeven, de produktterm $w\bar{z}$. Deze priemimplicant bestaat uit twee enen en twee don't cares. Dezelfde enen kunnen ook door de priemimplicant yz omrand worden. $\square$

Het zal duidelijk zijn dat een minimale som-van-produktenvorm uitsluitend uit priemimplicanten bestaat. Immers, dat zijn de produkttermen die een zo groot mogelijk gebied omranden. Het heeft geen zin een implicant in de som-van-produktenvorm op te nemen als er een andere priemimplicant bestaat die deze implicant bevat. Voor het bepalen van een minimale som-van-produktenvorm moeten we dus allereerst de verzameling priemimplicanten bepalen. Daaruit wordt vervolgens een minimale deelverzameling geselecteerd, welke tenminste de hele ON set van de functie dekt. Daarnaast kunnen er enkele don't care functiewaarden onder vallen.

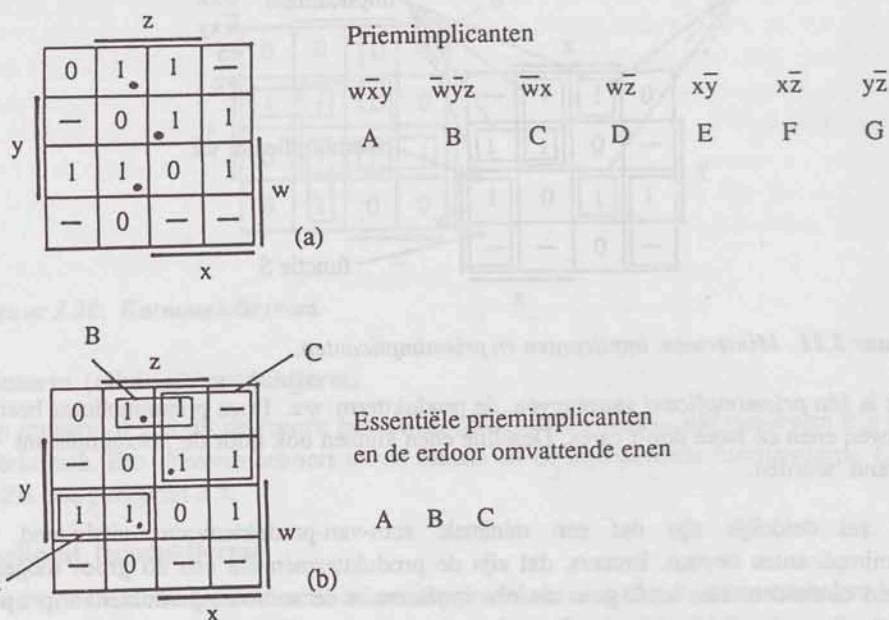
Voorbeeld

Bij de in figuur 3.12.a gespecificeerde functie behoren zeven priemimplicanten. Ga deze na! Verder geldt dat van de functiewaarden $f_i = 1$ in figuur 3.12.a drie enen door slechts één priemimplicant gedekt worden en wel de enen die behoren bij de mintermen $\bar{w}\bar{x}\bar{y}z$, $\bar{w}x\bar{y}z$ en $w\bar{x}yz$. De hierbij behorende priemimplicanten A, B en C zijn in figuur 3.12.b aangegeven. We noemen deze priemimplicanten essentiële priemimplicanten. $\square$

Essential prime implicant

Een essentiële priemimplicant (essential prime implicant) is een priemimplicant, die tenminste één minterm met functiewaarde $f_i = 1$ bevat welke door geen enkele andere priemimplicant gedekt wordt. Een essentiële priemimplicant behoort daarom noodzakelijk tot de eenvoudigste som-van-produktenvorm van de functie.

Een essentiële priemimplicant dekt meestal ook enen, waarvoor bij de dekking een keuze uit verschillende priemimplicanten bestaat. Deze enen worden automatisch gedekt door de essentiële priemimplicant. Daarnaast blijven er soms enen over die niet door essentiële priemimplicanten gedekt worden. Om deze enen te dekken moet alsnog een keuze uit de overige priemimplicanten gemaakt worden.

Voorbeeld**Figuur 3.12.** Priemimplicanten en essentiële priemimplicanten.

In figuur 3.12 blijft naast de door de essentiële priemimplicanten gedekte enen één 1 over, de minterm $wxy\bar{z}$, die door de priemimplicanten D ($w\bar{z}$), F ($x\bar{z}$) of G ($y\bar{z}$) gedekt wordt. De minimale som-van-produktenvorm voor de functie in figuur 3.12 is dus

$$S = A + B + C + \text{keuze uit } \begin{cases} D \\ F \\ G \end{cases} \quad (3.9)$$

te weten

$$S = w\bar{x}y + \bar{w}\bar{y}z + \bar{w}x + \begin{cases} w\bar{z} \\ x\bar{z} \\ y\bar{z} \end{cases} \quad (3.10)$$

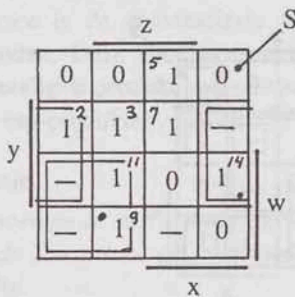
Er bestaan drie minimale som-van-produktenvormen van deze functie. □

Voorbeeld

Voor de functie in figuur 3.13 bestaan twee essentiële priemimplicanten. De enen die deze priemimplicanten essentieel maken zijn in het diagram met een • aangeduid. Voor de dekking van de overige mintermen met $f_i = 1$ moet een keuze gemaakt worden. Minimale som-van-produktenvormen van de functie S zijn:

$$S = w\bar{x} + y\bar{z} + \begin{cases} \bar{x}y + \bar{w}xz \\ \bar{w}y + \bar{w}xz \\ \bar{w}y + x\bar{y}z \end{cases} \quad (3.11)$$

Hoe we deze keuze problemen in een computer programmeren bespreken we in paragraaf 3.10. $\square$



Essentiële priemimplicanten

$w\bar{x}$ minterm m_9

$y\bar{z}$ minterm m_{14}

Dekking zoeken voor de mintermen

m_3 $\bar{x}y$ of $\bar{w}y$

m_5 $\bar{w}xz$ of $x\bar{y}z$

m_7 $\bar{w}y$ of $\bar{w}xz$

Figuur 3.13. Karnaughdiagram.

In deze paragraaf hebben we ons beziggehouden met het opstellen van *minimale som-van-produktenvormen van een logische functie*. Om een *beknopte specificatie* voor een logische schakeling op te stellen is deze vorm van de formule meestal het handigst. De formule is dan gemakkelijk interpreteerbaar, vanwege zijn beknoptheid.

Het belang van minimale som-van-produktenvormen in het realisatieproces is beperkt. Dit komt omdat de minimale schakeling vaak geen som-van-produktenstructuur heeft. Dit is zeker het geval bij multi-level realisaties van combinatorische functies. Een multi-level realisatie is een realisatie, waarbij meer dan twee operatoren in serie staan. Hoe de minimale som-van-produktenvorm moet worden omgezet in een voor de realisatie van de functie geschikte vorm is een complex probleem. We komen hier in de volgende hoofdstukken op terug.

3.4 De somvorm en de produktvorm van logische functies

Gegeven is de mintermvorm van een logische functie S in k variabelen,

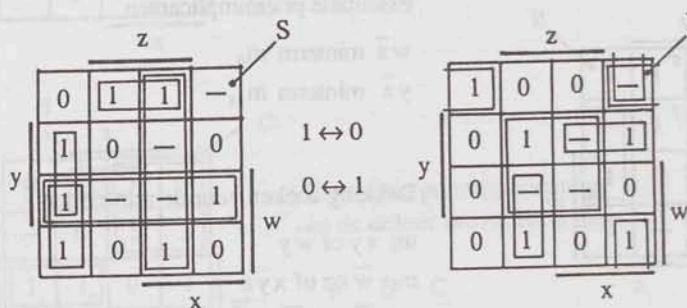
$$S = \sum_{i=0}^{2^k-1} m_i \cdot f_i \quad (3.12)$$

In paragraaf 2.7 is afgeleid dat de negatie $\bar{S}$ van S , ook wel het *complement* van S genoemd, verkregen wordt door alle functiewaarden f_i te negeren (te complementeren)

$$\bar{S} = \sum_{i=0}^{2^k-1} m_i \cdot \bar{f}_i \quad (3.13)$$

Deze bewerking kan in een Karnaughdiagram gemakkelijk uitgevoerd worden en wel door het verwisselen van alle nullen met enen en alle enen met nullen. Zie figuur 3.14. We merken uitdrukkelijk op dat eventuele don't cares don't care blijven.

Voorbeeld



Figuur 3.14. Omzetten van S in $\bar{S}$.

Voor de functie S is de eenvoudigste som-van-produktenvorm

$$S = wy + xz + w\bar{x}z + \bar{w}yz + \bar{x}y\bar{z} \quad (3.14)$$

en voor de functie $\bar{S}$

$$\bar{S} = yz + w\bar{x}z + \bar{w}xy + \bar{w}y\bar{z} + x\bar{y}\bar{z} \quad (3.14.a)$$

of

$$\bar{S} = yz + w\bar{x}z + \bar{w}x\bar{z} + \bar{w}y\bar{z} + x\bar{y}\bar{z} \quad \square (3.14.b)$$

We zien in dit voorbeeld dat de genegeerde vorm van een functie 'groter' kan zijn dan de niet-genegeerde vorm. Vaak echter is het omgekeerde het geval en is $\bar{S}$ eenvoudiger dan S (voor wat de omvang en het aantal van de termen betreft). Bij programmeerbare logica, in PLA's bijvoorbeeld, maakt men hier wel gebruik van. Beide vormen van de functie, S en $\bar{S}$, worden opgesteld en de eenvoudigste ervan wordt geprogrammeerd. In PLA's kan dit omdat deze componenten meestal zowel S als $\bar{S}$ naar buiten kunnen uitvoeren. Hebben we $\bar{S}$ geprogrammeerd en hebben we S nodig, dan complementeert de uitgangsschakeling de functie nogmaals. Zodat $(\bar{\bar{S}}) = S$ beschikbaar is. Zie ook hoofdstuk 8.

De produkt-van-sommenvorm

We hebben hiervoor gezien hoe we $\bar{S}$ in de som-van-produktenvorm uit een Karnaughdiagram van $\bar{S}$ kunnen aflezen. Meestal lezen we direct de nullen uit het diagram

van S uit, als waren het enen. Nu is $\overline{(\overline{S})} = S$. Passen we dit toe op formule (3.15), dan volgt hieruit met de *negatiewetten* van De Morgan:

$$\overline{S} = yz + w\overline{x}z + \overline{w}xy + \overline{w}\overline{y}z + x\overline{y}z \quad (3.15)$$

$$\begin{aligned} S &= \overline{\overline{S}} \\ &= \overline{yz + w\overline{x}z + \overline{w}xy + \overline{w}\overline{y}z + x\overline{y}z} \\ &= (\overline{y} + \overline{z})(\overline{w} + x + \overline{z})(w + \overline{x} + \overline{y})(w + y + z)(\overline{x} + y + z) \end{aligned} \quad (3.16)$$

Hiermee is de eenvoudigste vorm van de functie S in de *produkt-van-sommen*vorm gevonden. Deze conclusie geldt algemeen. Van elke functie S kan op deze wijze de eenvoudigste produkt-van-sommen gemaakt worden en wel uitgaande van de eenvoudigste som-van-produktenform van $\overline{S}$.

Notatie

We noemen in het vervolg de (eenvoudigste) som-van-produktenform vaak de *somvorm* van de functie en de (eenvoudigste) produkt-van-sommenform de *produktvorm* van de functie. $\square$

Recept conversie somvorm $\leftrightarrow$ produktvorm

Om een functie van de somvorm om te zetten naar de produktvorm kan het volgende recept worden toegepast.

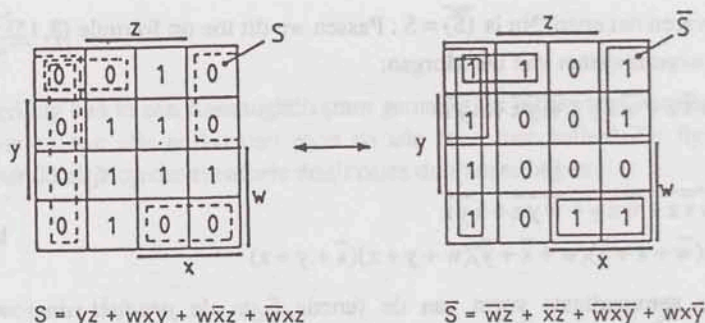
- Plaats S in een Karnaughdiagram voor k variabelen.
- Zet alle enen in nullen om en de nullen in enen. Laat de don't cares staan.
- Bepaal de (of een) *eenvoudigste* som-van-produktenform van $\overline{S}$.
- Complementeer deze eenmaal met behulp van de wetten van De Morgan. Hierna staat S in de (een) *eenvoudigste* produkt-van-sommenform.

Dezelfde procedure kan, in een aangepaste vorm, ook worden toegepast om de mintermvorm van een logische functie in k variabelen om te zetten naar de maxtermvorm. Bestaat de mintermvorm uit n produkten, dan heeft de maxtermvorm uit $2^k - n$ factoren.

Het plaatsen van een *produkt-van-sommen*vorm in een Karnaughdiagram verloopt wat lastiger dan voor de som-van-produktenform. Het alternatief voor het eerst uitwerken van de produktvorm naar de somvorm (de standaardvorm voor plaatsing) is dat we eerst de functie $\overline{S}$ bepalen. Bij een produktvorm van S resulteert dit in een somvorm voor $\overline{S}$ en deze vorm is gemakkelijk te plaatsen. Hierna is de somvorm S op de gebruikelijke wijze uit te lezen.

Voorbeeld

In figuur 3.15 is een toelichting gegeven op de procedure om de somvorm om te zetten naar de produktvorm, en andersom.



Figuur 3.15. Omzetting van de produktvorm naar de somvorm. □

Bij toepassing van de conversieprocedures op functies met don't cares worden de don't cares bij de somvorm vaak anders ingevuld dan bij de produktvorm. Bij beide vormen streeft men naar 'minimale omrandingen'. Dit betekent dat de uit de somvorm en de produktvorm van de formule afgeleide schakelingen logisch een verschillend ingangs/uitgangsgedrag kunnen vertonen! De verschillen treden uiteraard alleen op bij de ingangscombinaties waarvoor een don't care uitgangswaarde is gespecificeerd.

De boven geschetste conversieprocedure van somvorm naar produktvorm en omgekeerd wordt toegepast als schakelingen gemaakt worden met andere logische componenten dan die welke de operaties EN, OF en NIET realiseren. Zie de hoofdstukken 4 en 5.

3.5 Toepassingen van Karnaughdiagrammen 1

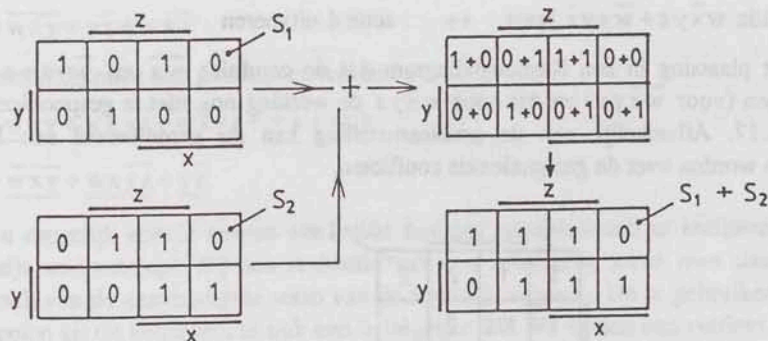
Naast de reductie van logische functies tot de minimale som-van-produktvorm bestaan er verschillende andere toepassingen van Karnaughdiagrammen. We noemen er enkele.

Logische bewerkingen op functies

In paragraaf 2.7 hebben we afgeleid dat logische bewerkingen op functies kunnen worden teruggebracht tot dezelfde logische bewerkingen op de afzonderlijke mintermen met hetzelfde rangnummer. Karnaughdiagrammen geven functies weer via de afzonderlijke mintermen en bijbehorende functiewaarden. Zij zijn daarom heel geschikt om er logische bewerkingen op functies in uit te voeren. Zie ook paragraaf 3.4, over het complementeren van functies. Een voorwaarde voor de toepasbaarheid van Karnaughdiagrammen bij de uitvoering van logische bewerkingen is dat de functies in hetzelfde type diagram genoteerd zijn, met hetzelfde aantal en dezelfde volgorde van de variabelen.

Voorbeeld

Gegeven zijn twee functies S_1 en S_2 in drie variabelen. In figuur 3.16 wordt hun som bepaald. De overige bewerkingen verlopen overeenkomstig.



Figuur 3.16. Het bepalen van $S_1 + S_2$.

□

Onderzoek van specificaties

In schakelingen hangt de uit te voeren actie op veel momenten af van de aangeboden ingangscombinatie. In specificaties wordt dit vaak opgegeven als

'conditie C_i waar' $\leftrightarrow$ 'actie A_i uitvoeren'

'conditie C_j waar' $\leftrightarrow$ 'actie A_j uitvoeren'

(3.17)

enz. Hierbij worden de condities meestal gespecificeerd als een som-van-produktenform, uitgedrukt in de ingangsvariabelen. De ervaring heeft geleerd dat hierbij veel fouten gemaakt worden, de condities overlappen elkaar voor een of meer van de ingangscombinaties of sommige ingangscombinaties behoren tot geen enkele conditie. In het laatste geval is de specificatie *incomplete*. (Bedenk dat een don't care uitgangswaarde ook expliciet als zodanig gespecificeerd moet worden.)

Wanneer condities elkaar *overlappen*, dan is de specificatie *in principe strijdig*. Immers, voor de tot de overlap van twee condities behorende ingangscombinaties kunnen geen twee verschillende acties uitgevoerd worden, zeker niet als deze elkaar uitsluiten. De specificatie is dan fout, tenzij het ingangscombinaties betreft die tot de don't care set behoren omdat zij nooit voorkomen. Het gesignaleerde conflict zal dan niet optreden. Ook komt het voor dat de bij twee overlappende condities gespecificeerde acties weliswaar verschillend zijn, maar niet persé strijdig met elkaar. Dit is bijvoorbeeld het geval als in de acties een of meer don't care specificaties voorkomen en de overige onderdelen identiek zijn. Door invullen van don't cares zijn de acties dan aan elkaar gelijk te maken. Bij het onderzoek naar overlappende condities kan met vrucht gebruik gemaakt worden van Karnaughdiagrammen.

Voorbeeld

Wanneer gespecificeerd is

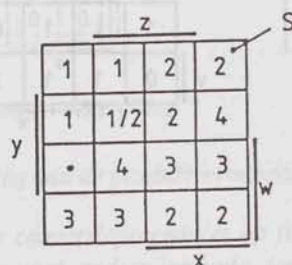
conditie $\overline{w}\overline{x}$ waar $\leftrightarrow$ actie 1 uitvoeren

conditie $\overline{w}yz + x\overline{y}$ waar $\leftrightarrow$ actie 2 uitvoeren

conditie $wxy + w\overline{x}\overline{y}$ waar $\leftrightarrow$ actie 3 uitvoeren

conditie $w\bar{x}yz + \bar{w}xy\bar{z}$ waar $\leftrightarrow$ actie 4 uitvoeren

dan leert plaatsing in een Karnaughdiagram dat de condities $\bar{w}\bar{x}$ en $\bar{w}yz + x\bar{y}$ elkaar overlappen (voor $\bar{w}\bar{x}yz$) en dat voor $w\bar{x}y\bar{z}$ de werking nog niet is gespecificeerd. Zie figuur 3.17. Afhankelijk van de probleemstelling kan nu gemotiveerd een beslissing genomen worden over de gesignaleerde conflicten.

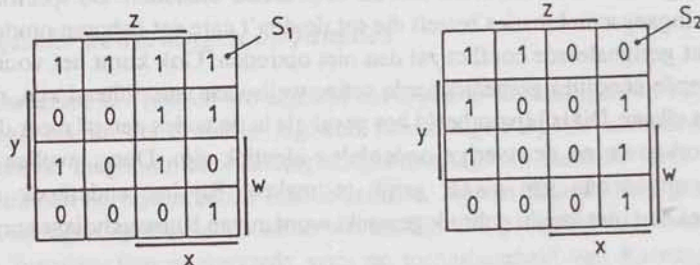


Figuur 3.17. Karnaughdiagram.

De in dit voorbeeld gesignaleerde conflicten treden niet op als bij de specificatie wordt uitgegaan van de mintermvorm van de functie(s). Meestal vindt men dit te gedetailleerd en probeert men in grotere stappen een specificatie op te stellen, met genoemde gevolgen.

Logische schakelingen met meer dan één uitgang

Het komt vaak voor dat bij het ontwerpen van een digitale schakeling op dezelfde verzameling logische variabelen meer dan één functie gedefinieerd is. Tijdens het bouwen van de schakeling bestaat dan meestal de mogelijkheid gedeelten van de schakeling voor verschillende uitgangsfuncties te benutten, hetgeen kan leiden tot een reductie van het aantal componenten.



Figuur 3.18. Specificatie van een schakeling met twee uitgangen S_1 en S_2 .

De functies S_1 en S_2 in figuur 3.18 zijn beide gedefinieerd op dezelfde ingangsverzameling $\{w, x, y, z\}$. De eenvoudigste somvorm van deze functies is:

$$S_1 = \bar{w}x + \bar{w}\bar{y} + x\bar{y}\bar{z} + xyz + w\bar{x}y\bar{z} \quad (3.18)$$

$$S_2 = \overline{w}\overline{x}\overline{y} + y\overline{z} + wx\overline{z} \quad (3.19)$$

Beide functies kunnen ook geschreven worden als

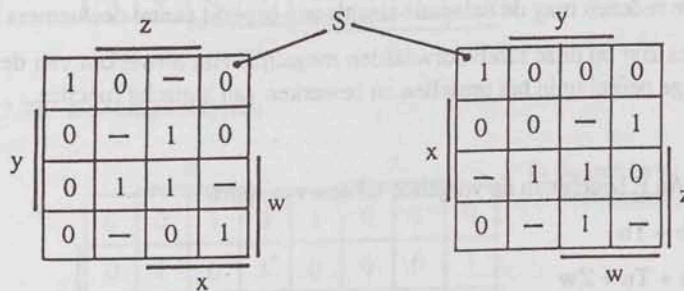
$$S_1 = \overline{w}\overline{x}\overline{y} + \overline{w}x\overline{y}z + \overline{w}x\overline{y}\overline{z} + \overline{w}x + xyz \quad (3.18.a)$$

$$S_2 = \overline{w}\overline{x}\overline{y} + \overline{w}x\overline{y}z + y\overline{z} \quad (3.19.b)$$

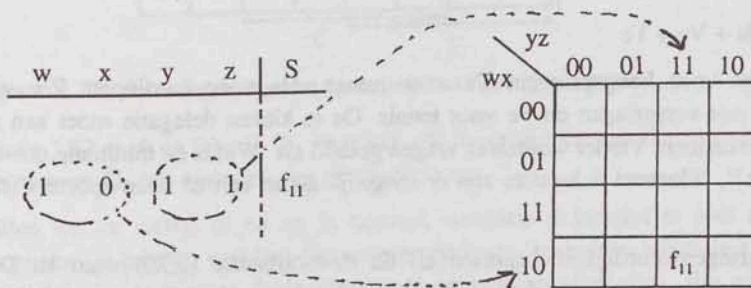
Het is nu mogelijk enkele termen van beide functies gecombineerd te realiseren. Deze termen zijn onderstreept. Bij een realisatie van een schakeling moet men daarom niet automatisch van de eenvoudigste vorm van de formules uitgaan. De te gebruiken componenten spelen bij dit keuzeproces ook een belangrijke rol. We stellen een verdere discussie hierover uit tot paragraaf 3.11.

Volgorde van de randschriften

Bij een Karnaughdiagram voor vier variabelen kunnen de variabelen in $4 \cdot 3 \cdot 2 \cdot 1 = 24$ verschillende volgorde genoteerd worden. Daarnaast kan ook de plaats van de strepen gevarieerd worden. Een vaste volgorde, zoals die gebruikt is bij de opbouw via het spiegelen, verdient een sterke voorkeur! Figuur 3.19 beschrijft twee Karnaughdiagrammen voor dezelfde functie. Ga de omzetting van de ene in de andere vorm na.



Figuur 3.19. Volgorde randschriften.



Figuur 3.20. Alternatieve vorm van het Karnaughdiagram.

In de literatuur komt de variant van figuur 3.20 veel voor. De randschriften zijn nu via een (gespiegeld binaire) code aangeduid, die correspondeert met de regels van de waarheidstabel. De functiewaarde bij een gegeven minterm kan nu snel geplaatst worden. Welke vorm men ook kiest, een vaste volgorde van de variabelen is ook hier belangrijk.

3.6 Toepassingen van Karnaughdiagrammen 2

Bij veel toepassingen moet een (deel)schakeling aan verschillende eisen voldoen. Deze eisen liggen in het beginstadium van een ontwerp vaak nog niet geheel vast. Bij het opstellen van logische functies is het dan verstandig om elk van de eisen in een afzonderlijke functie te specificeren en, nadat de deelfuncties geverifieerd zijn, de gewenste 'totaalfunctie' te vormen als produkt of som van de opgestelde deelfuncties. Het voordeel van deze aanpak is dat, als er in de specificatie van een van de delen iets verandert, het geheel weer gemakkelijk gegenereerd kan worden.

Het volgende voorbeeld beschrijft de samenstelling van een sportdelegatie voor een gezamenlijke sportdag van enkele scholen, waarbij aan de delegatie twee eisen gesteld worden:

- Voor alle genoemde sporten moet tenminste één deelnemer in de delegatie zitten.
- Om budgettaire redenen mag de delegatie slechts een beperkt aantal deelnemers hebben.

Hoeveel delegaties zijn bij deze randvoorwaarden mogelijk? Het uitwerken van deze vraag leidt tot een aardige oefening in het opstellen en bewerken van logische functies.

Voorbeeld

Vijf personen A t/m E beoefenen de volgende takken van sport.

A beoefent Ho + Tu

B beoefent Ha + Tu + Zw

C beoefent Ho + Ve + Te

D beoefent Ho + Te + Zw

E beoefent Ha + Ve + Te

waarbij Ho staat voor hoogspringen, Tu voor turnen, Ha voor hardlopen, Zw voor zwemmen, Ve voor verspringen en Te voor tennis. De te kiezen delegatie moet aan alle sporten kunnen meedoen. Verder worden er vragen gesteld als 'Wat is de minimale omvang van de delegatie?', 'Hoeveel delegaties zijn er mogelijk als er één of twee sporters meer meegaan?', enz.

De delegatie vertegenwoordigt alle sporten als de *Petricksfunctie* (3.20) waar is. Deze functie geeft aan dat er tenminste één vertegenwoordiger per sport is. Voor de wijze waarop functie (3.20) opgesteld is, zie paragraaf 2.12.

$$S = (A + C + D) \cdot (A + B) \cdot (B + E) \cdot (B + D) \cdot (C + E) \cdot (C + D + E) \quad (3.20)$$

Hierbij is gebruik gemaakt van de proposities

A: 'Sporter A gaat mee.'

enz. Plaatsen we functie (3.20) in een Karnaughdiagram voor vijf variabelen, dan geven de daarin voorkomende enen geschikte samenstellingen van de delegatie aan.

Het plaatsen van S kan via de produkttermen van de functie. Hiervoor moet de vorm (3.20) eerst uitgewerkt worden. Een produkt-van-sommenvorm kan echter veel gemakkelijker geplaatst worden door de functie $\bar{S}$ te plaatsen. Deze functie is

$$\bar{S} = \bar{A}\bar{C}\bar{D} + \bar{A}\bar{B} + \bar{B}\bar{E} + \bar{B}\bar{D} + \bar{C}\bar{E} + \bar{C}\bar{D}\bar{E} \quad (3.21)$$

Plaatsing van deze termen als nullen in het diagram leidt tot figuur 3.21. Uit figuur 3.21 volgt dat er 13 verschillende delegaties mogelijk zijn. Hiermee zijn alle mogelijkheden voor het samenstellen van een delegatie bekend.

	A				A				
	0	0	0	0	0	0	0	0	S (alle sporten vertegenwoordigd)
B	0	0	1	1*	1	1	1	0	
	0	0	1	1	1	1	1	1	
	0	0	0	0	0	1	1	0	
	C				E				
									D

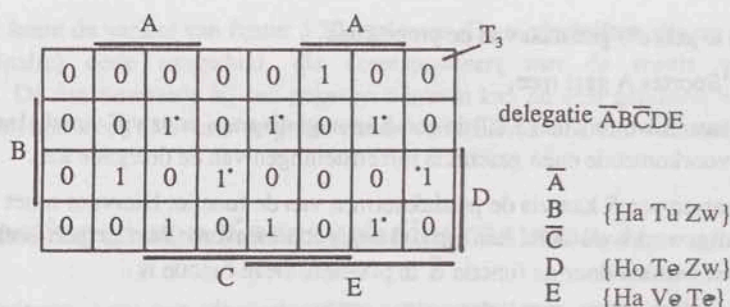
Figuur 3.21. Karnaughdiagram.

	A				A				
	0	0	1	0	1	0	1	0	T ₂ (keuze twee deelnemers)
B	0	1	0	1*	0	0	0	1	
	1	0	0	0	0	0	0	0	
	0	1	0	1	0	0	0	1	
	C				E				
									D

Figuur 3.22. Karnaughdiagram (groepen van twee deelnemers).

Figuur 3.22 geeft de functie T_2 weer van alle mogelijke keuzes van twee-uit-vijf kandidaat-deelnemers. Ga het opstellen van dit diagram na.

Willen we nu weten of en zo ja hoeveel complete delegaties er met twee deelnemers samengesteld kunnen worden, dan moeten de eisen 'complete delegatie voor alle sporten' met de eis 'er gaan twee deelnemers mee' gecombineerd worden. We nemen daarom het produkt van figuur 3.21 en 3.22 (minterm voor minterm, paragraaf 2.7). Dan blijkt er één delegatie mogelijk te zijn met twee deelnemers. Deze combinatie $\bar{A}BC\bar{D}\bar{E}$ is als 1* aangegeven.



Figuur 3.23. Karnaughdiagram (groepen van drie deelnemers).

Figuur 3.23 beschrijft alle mogelijke delegaties met drie personen. Hiervan zijn er zes geschikt. Deze zijn met een • aangegeven. □

In het bovenstaande voorbeeld is er sprake van twee verschillende eisen. Allereerst moet er voor elke tak van sport een deelnemer zijn. Deze eis is weergegeven in figuur 3.21. Daarnaast is er een andere eis, die aangeeft hoeveel deelnemers men wil afvaardigen. Deze eisen zijn voor twee respectievelijk drie deelnemers weergegeven in de figuren 3.22 en 3.23. De EN functie over beide eisen leidt tot het voldoen aan beide eisen. We verkrijgen daarmee een set oplossingen, als er tenminste oplossingen zijn. Ook andere relaties tussen op deze wijze geformuleerde voorwaarden zijn aldus te verwerken.

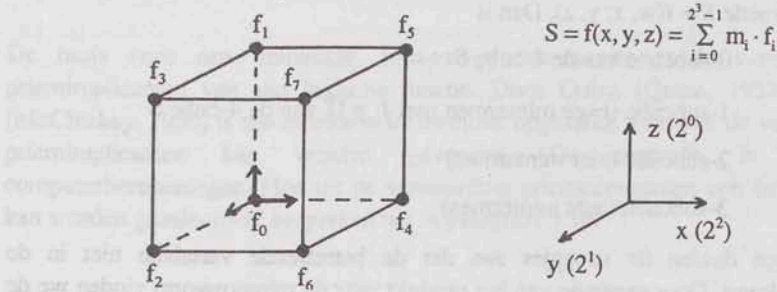
De in dit voorbeeld toegepaste technieken kunnen buiten de digitale techniek ook veelvuldig worden toegepast. Zo heeft een van de auteurs hiermee een nieuw verenigingsreglement op 'deugdelijkheid' onderzocht. Leidt elke stemmingsprocedure altijd tot een beslissing, om een van de vragen bij dit onderzoek te noemen. Tamelijk veel 'open-eind procedures' en 'incomplete procedures' kwamen aan het licht. Het verdient daarom aanbeveling de beginselen der logica en Karnaughdiagrammen op te nemen in de opleiding tot notaris.

3.7 Cube representatie van logische functies

Grafische voorstellingen van logische functies beperken zich niet tot het platte vlak, zoals in Karnaughdiagrammen. In de literatuur komen ook ruimtelijke voorstellingen van logische functies voor, met n -dimensionale kubussen (n -dimensional hypercubes). Hoewel deze ruimtelijke wijze van voorstellen bij vier en meer variabelen uiterst onpraktisch is, moeten we deze toch introduceren. Er wordt een bepaalde *terminologie* van afgeleid, die in vele artikelen voorkomt.

Een n -dimensionale kubus (cube of hypercube) is een regelmatige ruimtelijke constructie met 2^n knooppunten (vertices). Aan elk knooppunt wordt een bit string toegekend, vergelijkbaar met de randschriften bij Karnaughdiagrammen. Aldus correspondeert elk knooppunt met een minterm m_i van de functie en kan de betreffende functiewaarde f_i bij het knooppunt worden genoteerd door het knooppunt bijvoorbeeld vet te maken voor $f_i = 1$.

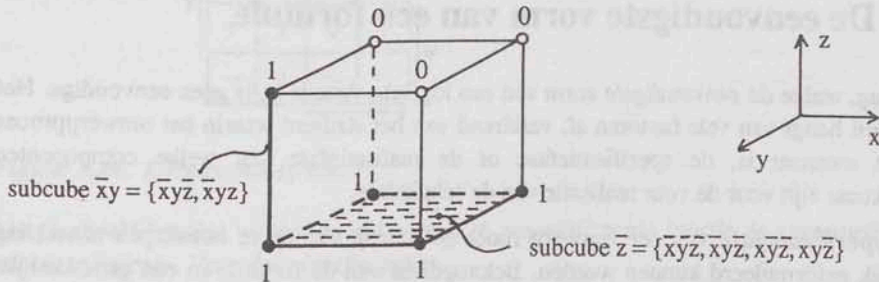
Voorbeeld



Figuur 3.24. 3-Dimensionale kubus.

□

Voorbeeld



Figuur 3.25. Cube representatie van een functie met twee subcubes.

 In figuur 3.25 is de functie S ,

$$S = \bar{x}y + \bar{z} \quad (3.22)$$

 in een driedimensionale kubus weergegeven. Uit deze *mintermvorm* is de gegeven functie als som-van-produkten gemakkelijk af te leiden. □

Als grafische voorstellingswijze zijn Karnaughdiagrammen in het voordeel. Tot functies in acht à negen variabelen zijn ze goed te hanteren. Wat van de cube representation overblijft is de terminologie, die in de literatuur bij algoritmebeschrijvingen wordt toegepast:

0-subcube ~ één knooppunt (met $f_i = 1$)
 produkt in alle variabelen (dit produkt is eerder minterm genoemd)

m-subcube ~ 2^m knooppunten (met $f_i = 1$)
 produkt van $n - m$ variabelen (produkt is eerder implicant genoemd)

enz. Bij Karnaughdiagrammen zijn de termen *implicant*, *prime implicant* en *essential prime implicant* geïntroduceerd. De hiermee overeenkomende termen zijn (sub)cube, prime (sub)cube en essential prime (sub)cube. De betekenis voor een dekking van de functie van deze aanduidingen is gelijk aan die van paragraaf 3.3.

Notatie

Zij gegeven een functie $S = f(w, x, y, z)$. Dan is

$w\bar{x}y\bar{z}$	0-subcube van de 4-cube S
$-x\bar{y}\bar{z}$	1-subcube (twee mintermen met $f_i = 1$) van de 4-cube S
$w-y-$	2-subcube (vier mintermen)
$---z$	3-subcube (acht mintermen)

In deze produkten duiden de streepjes aan dat de betreffende variabele niet in de produktterm voorkomt. Door expansie van het produkt naar de mintermvorm vinden we de omvatte mintermen. Gemakshalve laat men de aanduiding 'sub' bij de cube-aanduiding van de produkttermen meestal weg. $\square$

3.8 De eenvoudigste vorm van een formule

De vraag, welke de *eenvoudigste vorm* van een logische functie is, is geen eenvoudige. Het antwoord hangt van vele factoren af, variërend van het stadium waarin het ontwerpproces op dat moment is, de specificatiefase of de realisatiefase, tot welke componenten beschikbaar zijn voor de voor realisatie van de schakeling.

In de *specificatiefase* van een ontwerp moet de functie van de te ontwerpen schakeling duidelijk geformuleerd kunnen worden. Beknoptheid van de formule en een gemakkelijke interpretatie ervan zijn in dit stadium belangrijk. Met name bevordert dit de *verificatie* van de opgestelde specificatie ten eerste. Indien de specificatie in de vorm van een logische formule geschiedt, dan zal dit meestal in de som-van-produktenvorm zijn.

In de *realisatiefase* is meestal een *formuleconversie* nodig, tot een voor afbeelding naar de componenten geschikte vorm. Bij een realisatie met behulp van ROM's (Read Only Memories) moet de functie in de vorm van een waarheidstabel (mintermvorm) ingelezen en geprogrammeerd worden. Het vereenvoudigen van formules is dan overbodig. Bij andere componenten, PLA's (Programmable Logic Arrays), berust de realisatie op de som-van-produktenvorm. Hierbij is reductie tot een minimaal aantal produkttermen belangrijk. Van oudsher is hiervoor het *Quine-McCluskey algoritme* en ervan afgeleide versies in gebruik. Deze algoritmes veronderstellen een opbouw van de schakeling in twee niveaus.

Een geheel andere benadering bestaat uit een opbouw van de schakeling met behulp van 'poorten', operatoren van een geringe logische complexiteit, waarop de functie wordt afgebeeld. Als regel hebben 'poorten' strikte beperkingen voor het aantal ingangen, waardoor een multi-level opbouw van de schakeling nodig is. We moeten dan een logische functie via *decompositie* van de formule realiseren. Een eenvoudige vorm van decompositie is reeds in paragraaf 2.8 geïntroduceerd. Een voor multi-level decompositie geschikte datastructuur is het BDD (Binary Decision Diagram). Het Quine-McCluskey algoritme bespreken we in de paragrafen 3.9 t/m 3.11. BDD's als datastructuur bespreken we in paragraaf 3.12 t/m 3.14.

groep I met die uit groep II en van mintermen uit groep II met die uit groep III. Het blijkt dat alle mintermen uit tabel 3.1 minstens één maal bevat zijn in een der implicanten in tabel 3.2. In deze tabel zijn de implicanten geordend naar afwezigheid van variabelen. Dit vermindert de hoeveelheid werk bij verdere combinatiepogingen.

$\bar{w}\bar{x}\bar{y}z$	m_1	0001	✓	} groep I
$\bar{w}\bar{x}y\bar{z}$	m_2	0010	✓ d	
$\bar{w}x\bar{y}\bar{z}$	m_4	0100	✓ d	
$w\bar{x}\bar{y}\bar{z}$	m_8	1000	✓ d	
$\bar{w}x\bar{y}z$	m_5	0101	✓	} groep II
$\bar{w}xy\bar{z}$	m_6	0110	✓	
$w\bar{x}y\bar{z}$	m_{10}	1010	✓	
$wx\bar{y}\bar{z}$	m_{12}	1100	✓ d	
$\bar{w}xyz$	m_7	0111	✓	} groep III
$w\bar{x}yz$	m_{11}	1011	✓	
$wx\bar{y}z$	m_{13}	1101	✓ d	
$wxyz$	m_{14}	1110	✓	

Tabel 3.1. Mintermen met corresponderende ternaire notatie.

$\bar{x}y\bar{z}$	$m_2 + m_{10}$	0-10 ✓	} I
$\bar{x}\bar{y}\bar{z}$	$m_4 + m_{12}$	-100 ✓ d	
$\bar{x}\bar{y}z$	$m_5 + m_{13}$	-101 ✓	
$\bar{x}y\bar{z}$	$m_6 + m_{14}$	-110 ✓	
$\bar{w}-\bar{y}z$	$m_1 + m_5$	0-01	} II
$\bar{w}-y\bar{z}$	$m_2 + m_6$	0-10 ✓	
$w-\bar{y}\bar{z}$	$m_8 + m_{12}$	1-00 ✓ d	
$w-y\bar{z}$	$m_{10} + m_{14}$	1-10 ✓	
$\bar{w}x-\bar{z}$	$m_4 + m_6$	01-0 ✓	} III
$\bar{w}x-z$	$m_5 + m_7$	01-1 ✓	
$w\bar{x}-\bar{z}$	$m_8 + m_{10}$	10-0 ✓	
$wx-\bar{z}$	$m_{12} + m_{14}$	11-0 ✓	
$\bar{w}x\bar{y}-$	$m_4 + m_5$	010- ✓	} IV
$\bar{w}xy-$	$m_6 + m_7$	011- ✓	
$w\bar{x}y-$	$m_{10} + m_{11}$	101-	
$wx\bar{y}-$	$m_{12} + m_{13}$	110- ✓ d	

Tabel 3.2. Implicantentabel.

Vervolgens kan geprobeerd worden de implicanten uit tabel 3.2 paarsgewijs te verenigen. Het zal duidelijk zijn dat dit slechts mogelijk is binnen de groepen I, II, III en IV in tabel 3.2, dat wil zeggen dat twee implicanten slechts dan verenigd kunnen worden als de ontbrekende variabele in beide implicanten op dezelfde positie staat. Tabel 3.3 bevat het resultaat. Opgemerkt wordt nog dat elke implicant in tabel 3.3 twee maal gegenereerd wordt. (Waarom?)

$--y\bar{z}$	$m_2 + m_6 + m_{10} + m_{14}$	$--10$
$\bar{w}x--$	$m_4 + m_5 + m_6 + m_7$	$01--$
$-x\bar{y}-$	$m_4 + m_5 + m_{12} + m_{13}$	$-10-$
$-x-\bar{z}$	$m_4 + m_6 + m_{12} + m_{14}$	$-1-0$
$w--\bar{z}$	$m_8 + m_{10} + m_{12} + m_{14}$	$1--0$

Tabel 3.3. Implicantentabel.

Het blijkt dat geen der implicanten uit tabel 3.3 met een der andere verenigd kan worden. Alle implicanten uit deze tabel zijn daarom *priemimplicanten*. Het hierboven beschreven proces wordt gestopt.

Samenvatting

Bij het genereren van priemimplicanten wordt als volgt te werk gegaan.

- Ga uit van de verzameling mintermen, waarvoor de functiewaarde $f_i = 1$ is of waarvoor f_i niet gespecificeerd is. Merk deze laatste mintermen met d. Maak een tabel van alle implicanten die ontstaan door de vereniging van twee mintermen. Geef met d aan als de functiewaarde van *beide* mintermen niet gespecificeerd is.
- Verenig zoveel mogelijk de implicanten die uit twee mintermen bestaan tot implicanten die vier mintermen omvatten. Merk deze implicanten met d als alle omvatte mintermen een don't care functiewaarde bezitten.
- Herhaal dit proces tot er geen implicanten meer zijn die met andere kunnen worden gecombineerd.

Tijdens dit proces van samennemen worden alle implicanten, die tenminste één maal in een grotere implicant bevat zijn, met $\surd$ gemerkt. Alle implicanten die niet meer met $\surd$ gemerkt worden, kunnen niet met andere worden gecombineerd. Het zijn de priemimplicanten. $\square$

In bovenstaand voorbeeld worden de volgende priemimplicanten gevonden:

$$\bar{w}\bar{y}z \text{ en } w\bar{x}y \quad (\text{uit tabel 3.2})$$

en

$$\bar{w}x, w\bar{z}, x\bar{y}, x\bar{z}, \text{ en } y\bar{z} \quad (\text{uit tabel 3.3})$$

Ontstaan er priemimplicanten die uitsluitend mintermen bevatten met niet gespecificeerde functiewaarden, dan zijn deze voor de minimale dekking van de functie overbodig.

Vervolgens moet uit de verzameling priemimplicanten een (minimale) deelverzameling worden geselecteerd, waardoor de gehele functie bedekt wordt. Dit probleem bespreken we in paragraaf 3.10.

De methode voor het genereren van priemimplicanten beschrijft globaal de door Quine-McCluskey ontwikkelde methode voor het genereren van priemimplicanten. In deze uiteenzetting is niet gestreefd naar volledigheid. Zo is het bijvoorbeeld niet noodzakelijk om een gegeven functie eerst in de mintermvorm te schrijven voordat systematisch de priemimplicanten kunnen worden bepaald. Ook voor schakelingen met meer dan één uitgang zijn aangepaste versies van de methode van Quine-McCluskey ontwikkeld. Zie paragraaf 3.11.

3.10 Het bepalen van een dekking

Uit de verzameling der priemimplicanten moet een *dekking van de functie* bepaald worden. Een dekking van de functie is een *deelverzameling* van de verzameling der priemimplicanten en wel zodanig dat alle mintermen met $f_i = 1$ tenminste één maal omvat zijn. Een *minimale dekking* is een dekking met zo laag mogelijke kosten. Voorlopig is de kostenfunctie het aantal priemimplicanten. Andere kostenfuncties zijn mogelijk, bijvoorbeeld door ook het aantal variabelen per priemimplicant mee te nemen of het feit dat priemimplicanten voor meer dan één functie benodigd zijn.

Het *bepalen van een dekking* is een veel voorkomend probleem in de digitale techniek. Algemeen geformuleerd betreft het twee verzamelingen, een verzameling V van elementen

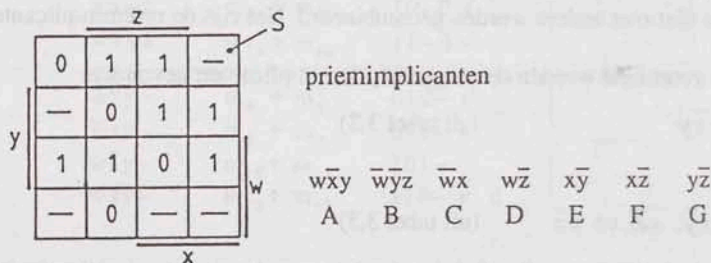
$$V = \{v_1, v_2, \dots, v_{n-1}, v_n\}$$

en een verzameling W van 'geschikte' deelverzamelingen van V ,

$$W = \{w_1, w_2, \dots, w_{k-1}, w_k\}$$

De elementen van W zijn daarbij meestal niet-disjunct. Afhankelijk van de toepassing moeten nu verschillende dekkingen van V uit W geselecteerd worden, meestal de kleinste in aantal elementen van W en/of met de laagste kosten.

Voorbeeld



Figuur 3.27. Karnaughdiagram met alle priemimplicanten.

De functie S in figuur 3.27 heeft zeven priemimplicanten, aangeduid met A t/m G. De dekkingstabel 3.4 geeft aan hoe de mintermen met $f_i = 1$ door de priemimplicanten gedekt worden.

mintermen		m_1	m_5	m_6	m_7	m_{10}	m_{11}	m_{14}
priemimplicanten		$\overline{w}\overline{x}\overline{y}z$	$\overline{w}x\overline{y}z$	$\overline{w}x\overline{y}\overline{z}$	$\overline{w}xyz$	$w\overline{x}\overline{y}\overline{z}$	$w\overline{x}yz$	$wxyz$
A	$w\overline{x}y$	•	•	•	•	×	×	•
B	$\overline{w}\overline{y}z$	×	×	•	•	•	•	•
C	$\overline{w}x$	•	×	×	×	•	•	•
D	$w\overline{z}$	•	•	•	•	×	•	×
E	$x\overline{y}$	•	×	•	•	•	•	•
F	$x\overline{z}$	•	•	×	•	•	•	×
G	$y\overline{z}$	•	•	×	•	×	•	×

Tabel 3.4. Dekkingstabel.

Uit tabel 3.1 volgt:

- minterm m_1 wordt slechts gedekt door priemimplicant B, B behoort noodzakelijk tot de selectie;
- minterm m_5 kan gedekt worden door B, door C of door E;
- minterm m_6 kan gedekt worden door C, door F of door G;
-
- minterm m_{14} kan gedekt worden door D, door F of door G.

We kunnen deze voorwaarden in een logische functie uitdrukken:

$$S = B \cdot (B + C + E) \cdot (C + F + G) \cdot C \cdot (A + D + G) \cdot A \cdot (D + F + G) \quad (3.23)$$

De logische variabelen A t/m G hebben hierin de waarde 1 (True) als de corresponderende priemimplicant tot de gekozen dekking behoort. Wanneer $S = 1$ is voor een bepaalde invulling van A t/m G, dan is er in ieder geval een dekking gevonden. Het dekkingsprobleem is hiermee elegant geformuleerd. Zie ook paragraaf 2.12 en [Petrick, 1956].

De rekenregels uit de schakelalgebra zijn van toepassing op de aldus gedefinieerde logische functie. Vooral toepassing van de wet $z \cdot (y + z) = z$ maakt dat er verschillende factoren in de formule geschrapt kunnen worden.

$$\begin{aligned} S &= B \cdot (B + C + E) \cdot (C + F + G) \cdot C \cdot (A + D + G) \cdot A \cdot (D + F + G) \\ &= B \cdot C \cdot A \cdot (D + F + G) \\ &= A \cdot B \cdot C \cdot D + A \cdot B \cdot C \cdot F + A \cdot B \cdot C \cdot G \end{aligned} \quad (3.24)$$

Deze vorm is niet verder te vereenvoudigen. Een som is 1 als tenminste één van de termen 1 is. Zodat een minimale dekking verkregen is als

$A = B = C = D = 1 \rightarrow$ dekking bevat de priemimplicanten $w\bar{x}y$, $\bar{w}\bar{y}z$, $\bar{w}x$ en $w\bar{z}$

$A = B = C = F = 1 \rightarrow$ dekking bevat de priemimplicanten $w\bar{x}y$, $\bar{w}\bar{y}z$, $\bar{w}x$ en $x\bar{z}$

$A = B = C = G = 1 \rightarrow$ dekking bevat de priemimplicanten $w\bar{x}y$, $\bar{w}\bar{y}z$, $\bar{w}x$ en $y\bar{z}$

Strikt genomen behoeven we niet voor de gehele dekkingstabel (cover table) de Petrickfunctie op te stellen. Eenvoudig is in te zien dat een beperkte Petrickfunctie ook voldoet. Immers, van een aantal priemimplicanten is direct bekend dat zij tot de dekking behoren. Dit zijn de essentiële priemimplicanten. In de tabel herkent men essentiële priemimplicanten gemakkelijk. Indien er in een kolom één kruisje staat, dan is er sprake van een essentiële priemimplicant. Een essentiële priemimplicant houdt in dat automatisch alle erin bevatte mintermen meegenomen zijn. In tabel 3.5 zijn alle gevolgen van elke essentiële priemimplicant met een cirkeltje aangegeven.

W \ V	V						
	m_1	m_5	m_6	m_7	m_{10}	m_{11}	m_{14}
A	.	.	.	.	⊗	⊗	.
B	⊗	⊗	.	.	.	.	.
C	.	⊗	⊗	⊗	.	.	.
D	.	.	.	.	×	.	×
E	.	×	.	.	.	.	.
F	.	.	×	.	.	.	×
G	.	.	×	.	×	.	×

⊗ noodzakelijk
⊗ gevolg

Tabel 3.5. Dekkingstabel met implicaties van de essentiële priemimplicanten.

Voor de nog niet gedekte mintermen moet nu een Petrickfunctie opgesteld worden. Voor tabel 3.5 is deze

$$S = A \cdot B \cdot C \cdot (D + F + G) \quad (3.25)$$

waarbij het produkt $A \cdot B \cdot C$ voor de essentiële priemimplicanten staat. Het is daarom zinvol een dekkingsprobleem in twee fasen op te lossen:

- spoor eerst alle essentiële priemimplicanten op;
- probeer daarna de overige mintermen zo gunstig mogelijk te dekken.

Een ander aspect van het dekkingsprobleem is dat sommige rijen van een dekkingstabel *dominant* zijn over andere rijen. Een voorbeeld hiervan geeft tabel 3.6. In deze tabel is de rij achter w_i dominant over de rij achter w_j , *bij gelijke kosten*. Bij het bepalen van een minimale dekking kan de rij w_j buiten beschouwing gelaten worden, tenminste wanneer de kosten van w_i kleiner dan of gelijk aan die van w_j zijn. Het verdient daarom aanbeveling na te gaan of er dominante rijen in een tabel bestaan (nadat de essentiële elementen uit W bepaald zijn). Het hangt dan van de toegevoegde kostenfunctie af of er rijen buiten beschouwing kunnen blijven.

W	V					
	v_1	v_2	v_3	v_4	v_5	v_6
w_i	x	x	•	x	x	x
w_j	•	x	•	x	•	•

Tabel 3.6. Dekkingstabel.

Voorbeeld

Gegeven is van de functie S de

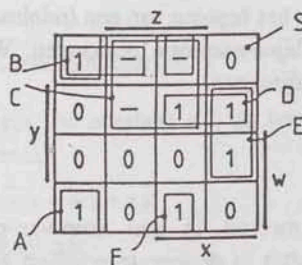
ON set: $S_1 = 0, 6, 7, 8, 13, 14$

DC set: $S_d = 1, 3, 5$

waarbij achter S_1 nummers van mintermen staan waarvoor $f_i = 1$ is en achter S_d nummers van mintermen waarvoor f_i niet gespecificeerd is. De priemimplicantentabel voor deze functie is tabel 3.7. Vergelijk ook figuur 3.28.

		m_0	m_6	m_7	m_8	m_{13}	m_{14}
$\bar{x}\bar{y}\bar{z}$	A	⊗	•	•	⊗	•	•
$\bar{w}x\bar{y}$	B	x	•	•	•	•	•
$\bar{w}z$	C	•	•	x	•	•	•
$\bar{w}xy$	D	•	x	x	•	•	•
$x\bar{y}\bar{z}$	E	•	⊗	•	•	•	⊗
$x\bar{y}z$	F	•	•	•	•	⊗	•

Tabel 3.7. Dekkingstabel.



Figuur 3.28. Karnaughdiagram.

In tabel 3.7 is rij A dominant over rij B en is rij D dominant over rij C. Tevens zijn A, E en F essentiële priemimplicanten. Bij gelijke realisatiekosten van de gegeven priemimplicanten mogen de rijen B en C dus geschrapt worden. Echter, priemimplicant C bevat één variabele minder dan priemimplicant D. Bij de nog te behandelen poortschakelingen kost realisatie van D één ingang meer dan een realisatie van C. Soms leidt dit tot een kostenverhoging. Kost in een realisatie een term in twee variabelen k_1 kosteneenheden en een term in drie of

vier variabelen k_2 kosteneenheden, met $k_2 > k_1$, dan mag de tweede term buiten beschouwing blijven, maar de eerste niet.

Voor tabel 3.7 is de Petrickfunctie:

$$\begin{aligned} S &= (A+B) \cdot (D+E) \cdot (C+D) \cdot A \cdot F \cdot E \\ &= A \cdot E \cdot F \cdot (C+D) \\ &= A \cdot E \cdot F \cdot C + A \cdot E \cdot F \cdot D \end{aligned} \quad (3.26)$$

Omdat priemimplicant C eenvoudiger is dan priemimplicant D verdient de oplossing met priemimplicant C de voorkeur. De dekking bestaat dan uit de priemimplicanten

$$\overline{x}\overline{y}\overline{z}, x\overline{y}\overline{z}, x\overline{y}z \text{ en } \overline{w}z \quad (3.27)$$

Uit dit voorbeeld blijkt dat in het behandelde dekkingsalgoritme realisatie-aspecten zoals een kostenfunctie in principe meegenomen kunnen worden. $\square$

Voor een uitgebreidere behandeling van dekkingsalgoritmes verwijzen we naar de literatuur, onder andere naar [Gimpel, 1965; Gordon, 1966]. Dekkingsproblemen met een kostenfunctie zijn besproken door [Robinson, 1967].

3.11 Uitbreidingen op het Quine-McCluskey algoritme

Een logische functie in n variabelen heeft 2^n mintermen. *Reductie-algoritmes*, welke uitgaan van de verzameling der mintermen, hebben daarom bij functies in relatief veel variabelen ook veel geheugen en/of computertijd nodig. Zo ook het eerder geïntroduceerde Quine-McCluskey algoritme. Om hiermee van een functie in 16 variabelen de priemimplicant yz te vinden moeten in 13 ronden 2^{14} mintermen stapsgewijs gecombineerd worden. Bovendien worden alle samenstellende implicanten meer dan één keer gegenereerd. Er zijn daarom op het Quine-McCluskey algoritme verschillende verfijningen bedacht om enerzijds het genereren van de priemimplicanten en anderzijds het bepalen van een (minimale) dekking te versnellen. Daarbij wordt soms volstaan met bijna-minimale dekkingen. We beschrijven hieronder beknopt de hoofdlijnen van enkele verfijningen.

Iteratieve consensus

Logische functies worden in de praktijk meestal in een som-van-produktenvorm gespecificeerd. Zo'n som-van-produktenvorm sluit in de meeste gevallen goed aan bij de gegeven probleemstelling. De opgestelde functie is dan meestal niet in de minimale som-van-produktenvorm, zowel voor wat betreft het aantal als de omvang van de produkttermen. Om de functie te vereenvoudigen behoeft dan niet noodzakelijk teruggegaan te worden naar de mintermvorm [Quine, 1955; Mott, 1960].

Meestal wordt gestart met de *triviale vereenvoudigingen*. Produkttermen kunnen bijvoorbeeld geschrapt worden als deze gelijk zijn aan andere produkttermen of erin bevat zijn. Dit op grond van de gelijkheidswet (2.15.a),

$$z + z = z$$

de absorptiewet (2.24.a)

$$z + y \cdot z = z$$

en de absorptiewet (2.25.a)

$$y + \bar{y}z = y + z$$

Ook kunnen in één variabele verschillende produkttermen samengenomen worden op grond van de negatiewet (2.21.a)

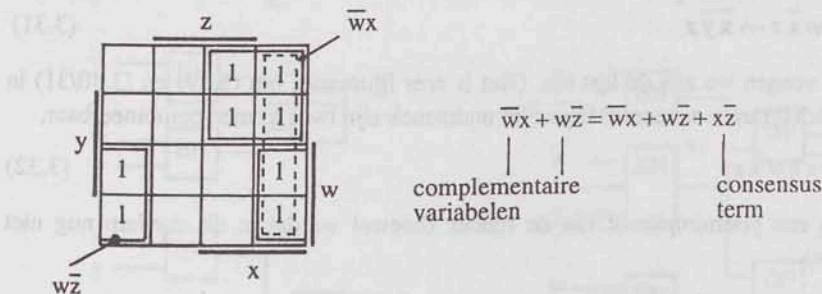
$$\bar{y}z + yz = z$$

De na deze vereenvoudigingen gevonden gereduceerde verzameling produkttermen (implicanten) is niet noodzakelijk minimaal. Op basis van het *consensustheorema* (2.26.a)

$$xy + \bar{x}z + yz = xy + \bar{x}z$$

wordt de gevonden verzameling produkttermen vervolgens uitgebreid, met als doel (nieuwe) (priem)implicanten te genereren.

Voorbeeld



Figuur 3.29. Karnaughdiagram.

Uit de 'consensus' (= overlap) van de termen $\bar{w}x$ en $w\bar{z}$ vinden we op basis van het consensustheorema

$$\bar{w}x + w\bar{z} = \bar{w}x + w\bar{z} + xz \quad (3.28)$$

de (priem)implicant xz . □

Het op *iteratieve consensus* gebaseerde algoritme voor het genereren van de verzameling der priemimplicanten werkt als volgt.

1. Stel een som-van-produktenvorm op voor de functie.
2. Verwijder termen die reeds in andere termen bevat zijn en/of neem zoveel mogelijk termen samen.

3. Bepaal de consensus van alle produkttermen onderling. Een consensusterm $\neq 0$ bestaat
 - indien twee produkten een variabele gemeen hebben, die in het ene produkt gewoon en in het andere produkt gecomplementeerd voorkomt.
 - de overige variabelen in de twee produkten niet complementair voorkomen.
4. Voeg de nieuwe consensustermen aan de lijst toe.

Herhaal de stappen 2 t/m 4 als er nieuwe termen aan de lijst toegevoegd zijn. Anders stop. De aldus gevonden verzameling produkttermen is de verzameling priemimplicanten van de functie [Mott, 1960].

Voorbeeld

De logische functie S,

$$S = \overline{w}\overline{x}\overline{y} + \overline{w}\overline{x}yz + \overline{w}xy\overline{z} + wy + w\overline{x}z + wx\overline{y} \quad (3.29)$$

bevat geen op basis van (2.24.a) of (2.26.a) overbodige termen. De consensus van de eerste term $\overline{w}\overline{x}\overline{y}$ met de overige leidt tot twee nieuwe produkttermen, te weten

$$\overline{w}\overline{x}\overline{y} \text{ en } \overline{w}\overline{x}yz \rightarrow \overline{w}\overline{x}z \quad (3.30)$$

$$\overline{w}\overline{x}\overline{y} \text{ en } \overline{w}xy\overline{z} \rightarrow \overline{x}\overline{y}\overline{z} \quad (3.31)$$

Deze termen voegen we aan de lijst toe. (Het is zeer illustratief om (3.29) en (3.30/31) in een Karnaughdiagram te tekenen!) Bij nader onderzoek zijn twee termen combineerbaar,

$$w\overline{x}z + \overline{w}\overline{x}z = \overline{x}z \quad (3.32)$$

Deze term is een priemimplicant van de functie (hoewel we dat in dit stadium nog niet kunnen weten).

Op deze wijze doorgaand vinden we de verzameling der priemimplicanten:

$$\{\overline{w}\overline{x}\overline{y}, wx, wy, wz, \overline{x}z, xy\overline{z}\} \quad \square (3.33)$$

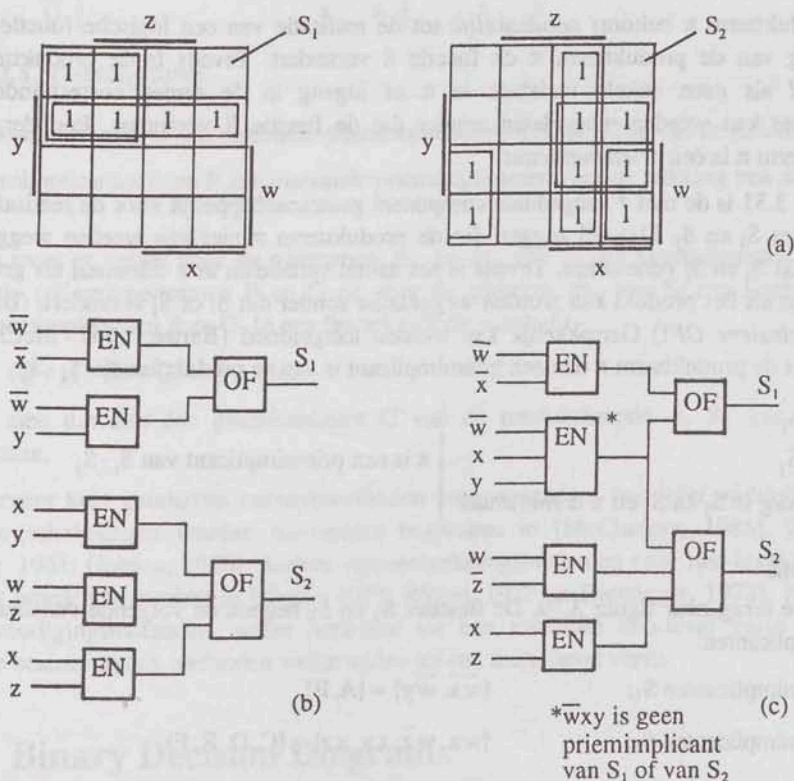
Het op combineren en iteratieve consensus gebaseerd algoritme heeft veel minder geheugen nodig dan het oorspronkelijke algoritme. Zie ook [Slagle, 1970]. Uitbreidingen van het iteratieve consensusalgoritme naar het dekkingsprobleem zijn onder meer gepubliceerd door [Biswas, 1971/1986; Mott, 1960; Tison, 1967]. Verdere verfijningen zijn mogelijk als we bedenken dat niet alle priemimplicanten tot de minimale som-van-produktenvorm behoren [Breitbart, 1979]. Daarvan kan gebruik gemaakt worden om het algoritme te versnellen [Gurunath, 1989].

Het op iteratieve consensus gebaseerde algoritme leidt tot een minimale som-van-produktenvorm. Deze kan op de in paragraaf 3.10 besproken manier uit de verzameling der priemimplicanten worden afgeleid. Deze vorm van de formule is de geschikte vorm voor programmering van de functie in vele vormen van *programmable logic*. We noemen een dergelijke vorm van de functie de *two-level vorm*.

Multiple-output networks

Meestal realiseert de combinatoriek van een logische schakeling verschillende logische functies op dezelfde (of elkaar overlappende) ingangsverzameling(en). Bij het ontwerp van de schakeling en vooral bij de optimalisatie/minimalisatie van het ontwerp staat de ontwerper voor de keuze: worden alle functies afzonderlijk ontworpen of kan/moet het ontwerp voor alle functies gecombineerd worden. Het volgende voorbeeld toont aan dat, als we uitsluitend kijken naar het benodigde aantal logische operatoren/poorten, een gecombineerd ontwerp voordelen kan bieden.

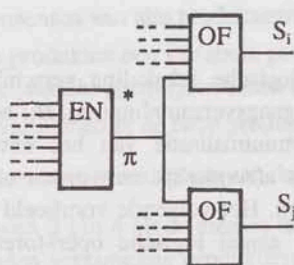
Voorbeeld



Figuur 3.30. Ontwerp multiple-output network.

In figuur 3.30.a zijn twee logische functies S_1 en S_2 gespecificeerd. De realisatie op basis van de minimale som-van-produktenvorm van beide functies, uitgevoerd met componenten die de logische bewerkingen EN en OF kunnen realiseren, staat in figuur 3.30.b. In figuur 3.30.c staat een gecombineerde realisatie. Indien de kostenfunctie het aantal componenten/het aantal ingangen per component bevat, dan is realisatie 3.30.c beter. $\square$

Bij *single-output functions* is de minimale som-van-produktenvorm gebaseerd op de priemimplicanten van de functie. Hoe zit dit bij gemeenschappelijke termen van functies?



Figuur 3.31. Gemeenschappelijke produktterm.

Een produktterm π behoort *noodzakelijk* tot de realisatie van een logische functie S als weglating van de produktterm π de functie S verandert. Tevens is de produktterm π *minimaal* als geen enkele variabele in π of ingang in de ermee corresponderende component kan worden weggelaten zonder dat de functie S verandert. Een dergelijke produktterm π is een *priemimplicant*.

In figuur 3.31 is de met * aangeduide component gemeenschappelijk voor de realisatie van de functies S_i en S_j . Dat wil zeggen dat de produktterm π niet kan worden weggelaten zonder dat S_i en S_j veranderen. Tevens is het aantal variabelen in π minimaal als geen der variabelen uit het produkt kan worden weggelaten zonder dat S_i of S_j verandert. (Bedoeld is de *inclusieve OF*!) Gemakkelijk kan worden aangetoond [Bartee, 1961; McCluskey, 1986] dat de produktterm π dan een priemimplicant is van de produktfunctie $S_i \cdot S_j$:

$$\left. \begin{array}{l} \pi \in S_i \\ \pi \in S_j \\ \pi \text{ nodig in } S_i \text{ en } S_j \text{ en } \pi \text{ is minimaal} \end{array} \right\} \pi \text{ is een priemimplicant van } S_i \cdot S_j \quad (3.34)$$

Voorbeeld

Keren we terug naar figuur 3.30. De functies S_1 en S_2 hebben de volgende verzamelingen priemimplicanten:

$$\text{priemimplicanten } S_1: \quad \{\bar{w}\bar{x}, \bar{w}y\} = \{A, B\}$$

$$\text{priemimplicanten } S_2: \quad \{wx, w\bar{z}, xy, xz\} = \{C, D, E, F\}$$

$$\text{priemimplicanten } S_1 \cdot S_2: \quad \{\bar{w}xy\} = \{G\}$$

Om beide functies te kunnen realiseren moeten we uit de drie verzamelingen priemimplicanten een keuze maken. Daartoe stellen we een multiple-output dekkingstabel op. Zie tabel 3.8.

Uit tabel 3.8 volgt:

		mintermen van S_1						mintermen van S_2									
		m_0	m_1	m_2	m_3	m_6	m_7	m_5	m_6	m_7	m_8	m_{10}	m_{12}	m_{13}	m_{14}	m_{15}	
S_1	A	x	x	x	x												
	B			x	x	x	x										
S_2	C												x	x	x	x	
	D										x	x	x		x		
	E								x	x					x	x	
	F							x		x				x		x	
$S_1 \cdot S_2$	G					x	x		x	x							

Tabel 3.8. Dekkingstabel.

- Priemimplicant A is een essentiële priemimplicant voor de dekking van de functie S_1 .
- Priemimplicanten D en F zijn essentiële priemimplicanten voor de dekking van de functie S_2 .
- Dan moet er verder voor de mintermen m_6 en m_7 van S_1 een keuze gemaakt worden uit de (priem)implicanten B en G en voor de minterm m_6 van S_2 een keuze uit de (priem)implicanten E en G. In een Petrick functie wordt dit

$$(B + G) \cdot (E + G) = G + B \cdot E$$

We zien dat met één priemimplicant G van de produktfunctie $S_1 \cdot S_2$ kan worden volstaan.

De hiervoor kort geschetste ontwerpmethoden voor zogeheten *two-level multiple-output* logische schakelingen worden uitvoeriger besproken in [McCluskey, 1986]. Zie ook [Bartee, 1961; Gordon, 1966]. Andere vereenvoudigingsmethoden voor two-level logische functies worden besproken in [Nagle, 1975; Rhyne, 1977 en Dietmeyer, 1978]. Naast de vereenvoudigingsmethoden, welke herleiden tot een minimale two-level vorm van de formule bestaan er ook methoden welke leiden tot een multi-level vorm.

3.12 Binary Decision Diagrams

De mintermvorm en de daarvan afgeleide som- en produktvorm van logische functies zijn niet voor alle toepassingen een geschikt uitgangspunt voor de realisatie van de schakeling. Vooral voor opslag van functies in computergeheugens en bij veel toepassingen van CAD (Computer Aided Design) software zijn deze vormen minder geschikt. De mintermvorm heeft als voordeel dat het een tabelvorm is. Op het door de ingangscombinatie bepaalde 'adres' vinden we direct de bijbehorende functiewaarde. Als nadeel geldt dat deze tabellen zeer groot kunnen worden (2^n adressen bij n variabelen). De som-van-produktenvorm lijkt handiger. Voor 'willekeurige' logische functies heeft het aantal produkten echter ook de neiging min of meer exponentieel toe te nemen, zij het minder snel dan bij de mintermvorm.

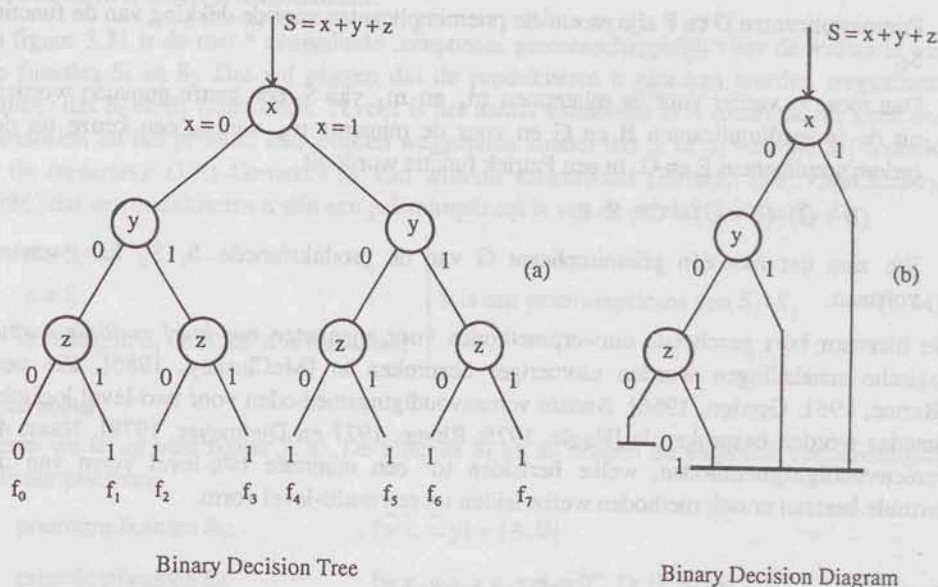
Het opzoeken van de bij een ingangscombinatie behorende functiewaarde is lastiger. Maximaal moeten alle produkten ingevuld worden. In CAD software maakt men veel gebruik van *Binary Decision Diagrams* (BDD's). Deze vorm is afgeleid van de op de *Shannondecompositie* gebaseerde *Binary Decision Trees*. Deze zijn eerder in paragraaf 2.8 geïntroduceerd.

Voorbeeld

Figuur 3.32.a beschrijft de functie

$$S = x + y + z$$

in een Binary Decision Tree. Deze graph volgt rechtstreeks uit de Shannondecompositie van S . Figuur 3.32.b is de vereenvoudigde vorm ervan, die we Binary Decision Diagram zullen noemen [Akers, 1978; Lee, 1959]. Het verschil is dat in een BDD takken, die naar een vaste functiewaarde leiden, zich niet langer voortzetten. Zulke takken verwijzen direct naar de uitgangswaarde 0 of 1 door. $\square$



Figuur 3.32. Grafische voorstellingen van de functie $S = x + y + z$.

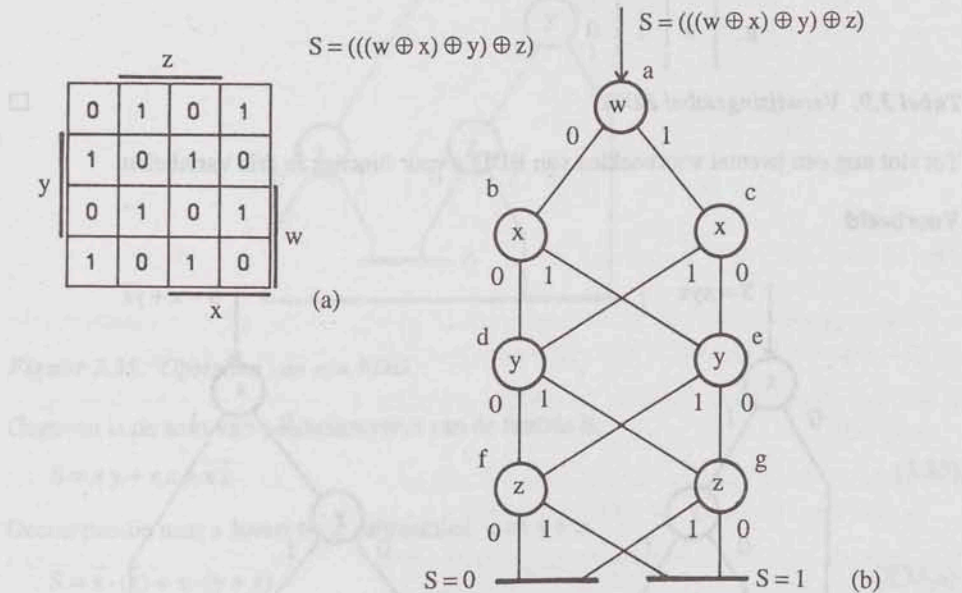
Een BDD is een gerichte graph. Te beginnen met het bovenste knooppunt wordt de functie gedecomposeerd naar de in het knooppunt ingeschreven variabele. Afhankelijk van de waarde van die variabele gaan we langs één van twee wegen het knooppunt uit en wel naar een volgend knooppunt (meestal naar links als de variabele 0 is en naar rechts als de variabele 1 is) of naar één van de uitgangen met functiewaarde 0 of 1. Elk knooppunt in een BDT heeft één ingang. In een BDD kan een knooppunt één of meer ingangen hebben.

Een BDD definieert een procedure om, bij gegeven waarden van de ingangsvariabelen, de waarde van de bijbehorende logische functie snel te bepalen. In een BDT (complete boom)

doorlopen we hierbij precies n achtereenvolgende knooppunten en in een BDD *maximaal* n (bij functies in n variabelen). Besparing op geheugenruimte en tijdbesparing bij de evaluatie van logische functies zijn de voornaamste redenen om over te gaan op BDD's.

Voorbeeld Parity functie

De parity functie is een functie, die aangeeft dat het aantal enen in de momentele ingangscombinatie ervan even of oneven is (paragraaf 1.5). De parity functie voor n variabelen heeft 2^{n-1} produkttermen met functiewaarde $f_i = 1$. In de somvorm kan deze functie niet verder vereenvoudigd worden. Zie figuur 3.33.



Figuur 3.33. Parity functie en bijbehorende BDD.

De BDD van de parity functie heeft altijd twee knooppunten per variabele. De BDD neemt lineair toe met het aantal ingangsvariabelen. Dit in tegenstelling tot de eenvoudigste som-van-produkten-vorm ervan, die exponentieel toeneemt met het aantal variabelen.

Bij opslag in een computergeheugen bestaat de BDD van de parity functie uit een tabel, die bestaat uit doorverwijzingen (pointers) naar de volgende knooppunten. Zie tabel 3.9. Om deze tabel te kunnen opstellen zijn de knooppunten in figuur 3.33.b voorzien van de letters a t/m g. Ook de omvang van deze tabel neemt lineair toe met het aantal variabelen n van de parity functie. In software zijn dergelijke tabellen snel te evalueren.

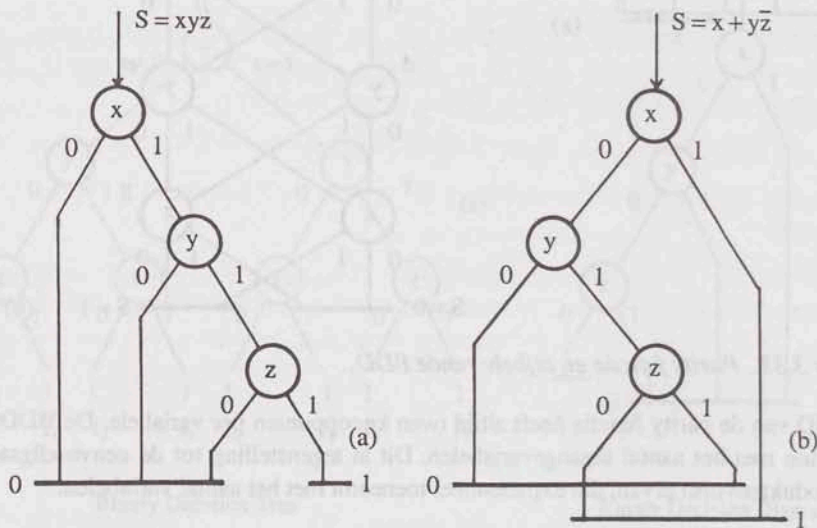
knooppunt	var	= 0	= 1
$S \rightarrow a$	w	b	c
b	x	d	e
c	x	e	d
d	y	f	g
e	y	g	f
f	z	0	1
g	z	1	0

Tabel 3.9. Verwijzingstabel BDD.

□

Tot slot nog een tweetal voorbeelden van BDD's voor functies in drie variabelen.

Voorbeeld

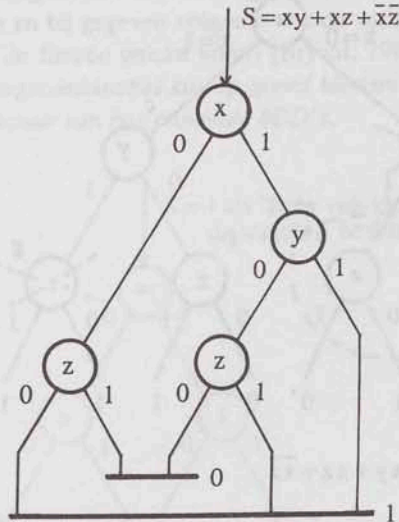


Figuur 3.34. Voorbeelden BDD's.

□

3.13 Het opstellen van BDD's

BDD's kunnen worden opgesteld door één BDT, een complete boom, te vereenvoudigen. Ook is het mogelijk direct van een (minimale) som-van-produktenvorm uit te gaan. We beginnen met de laatste vorm.

Voorbeeld**Figuur 3.35.** Opstellen van een BDD.

Gegeven is de som-van-produktenvorm van de functie S ,

$$S = xy + xz + \bar{x}\bar{z} \quad (3.35)$$

Decompositie naar x levert twee subfuncties, $\bar{z}$ en $y + z$:

$$S = \bar{x} \cdot (\bar{z}) + x \cdot (y + z) \quad (3.35.a)$$

In de linkertak komt de variabele y niet meer voor. We gaan dan direct door naar het knooppunt voor de variabele z . Voor $z = 0$ gaat dit knooppunt naar de functiewaarde 1. Voor $z = 1$ is het de functiewaarde 0.

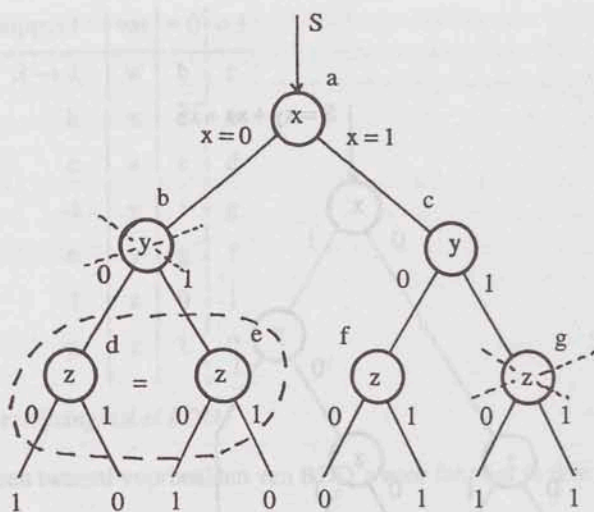
In de rechtertak moet een knooppunt naar de variabele y opgenomen worden. Bij $y = 1$ is de functiewaarde bekend. Bij $y = 0$ volgt er nog een knooppunt voor z .

Een blik op de bijbehorende complete boom (figuur 3.36) leert dat de twee linker takken van de variabele z identiek zijn (knooppunten d en e). Het knooppunt b voor de variabele y levert dan geen nieuwe informatie op en mag geschrapt worden. De takken van de knooppunten d en e worden daarbij gecombineerd.

Verder leidt knooppunt g voor de variabele z altijd tot de functiewaarde 1. Dit knooppunt is dus overbodig. Hiermee zijn we op de BDD van figuur 3.35 gekomen. $\square$

Vereenvoudigingsregels

Uitgaande van een complete BDT gelden de volgende vereenvoudigingsregels.



Figuur 3.36. BDT voor $S = xy + xz + \bar{x}z$

- Indien alle uitgaande takken van een bepaald knooppunt tot dezelfde functiewaarde leiden, dan is dit knooppunt met alle onderliggende knooppunten overbodig.
- Hebben twee takken van een knooppunt een identiek gevolg, dan is het knooppunt overbodig.
- Hebben twee of meer takken van verschillende knooppunten een identiek vervolg, dan mogen deze vervolgtakken tot één tak gecombineerd worden.

De hierboven beschreven vereenvoudigingsregels gelden uiteraard ook indien van de BDD voor een som-van-produktenvorm wordt uitgegaan.

Voorbeeld

Ga de constructie van de in figuur 3.37 beschreven BDD na. □

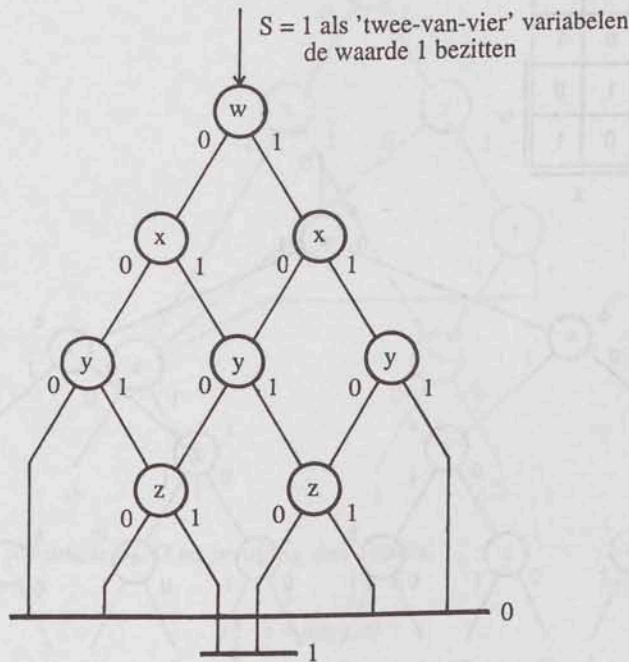
Volgorde van de variabelen

De volgorde van de variabelen, waarnaar een logische functie gedecomposeerd wordt, is van invloed op het (minimale) aantal knooppunten van de (minimale) BDD van de functie en aldus op de omvang van de tabel waarin de functie wordt opgeslagen. We hebben deze problematiek reeds toegelicht in paragraaf 2.8, voor de Shannondecompositie.

Zelfs per tak mag de volgorde voor de resterende variabelen nog worden aangepast en wel onafhankelijk van de volgorde in de andere takken. Het doel hiervan is te komen tot een graph met zo min mogelijk knooppunten. Dit resulteert weer in een minimale tabel voor opslag van de functie en een lagere gemiddelde zoektijd erin.

Het opstellen van een BDD met een minimaal aantal knooppunten voor functies in veel variabelen is een tijdrovend probleem. Zie [Friedman, 1987; Fujita, 1991]. In het algemeen

verdient het daarbij aanbeveling een vaste volgorde van de variabelen in de decompositie aan te houden. Het *vergelijken van twee functies* (gelijk of ongelijk en indien ongelijk voor welke ingangscombinaties) is dan het eenvoudigst. Een *minimale BDD* is voor een *volledig gespecificeerde functie* en bij gegeven volgorde van de variabelen namelijk een *canonieke vorm* (dit is een voor de functie unieke vorm) [Bryant, 1986]. Het *vergelijken van twee functies voor alle ingangscombinaties kan op grond hiervan worden teruggebracht tot het vergelijken van de structuur van hun minimale BDD's*.



Figuur 3.37. Binary Decision Diagram.

Don't cares in BDD's

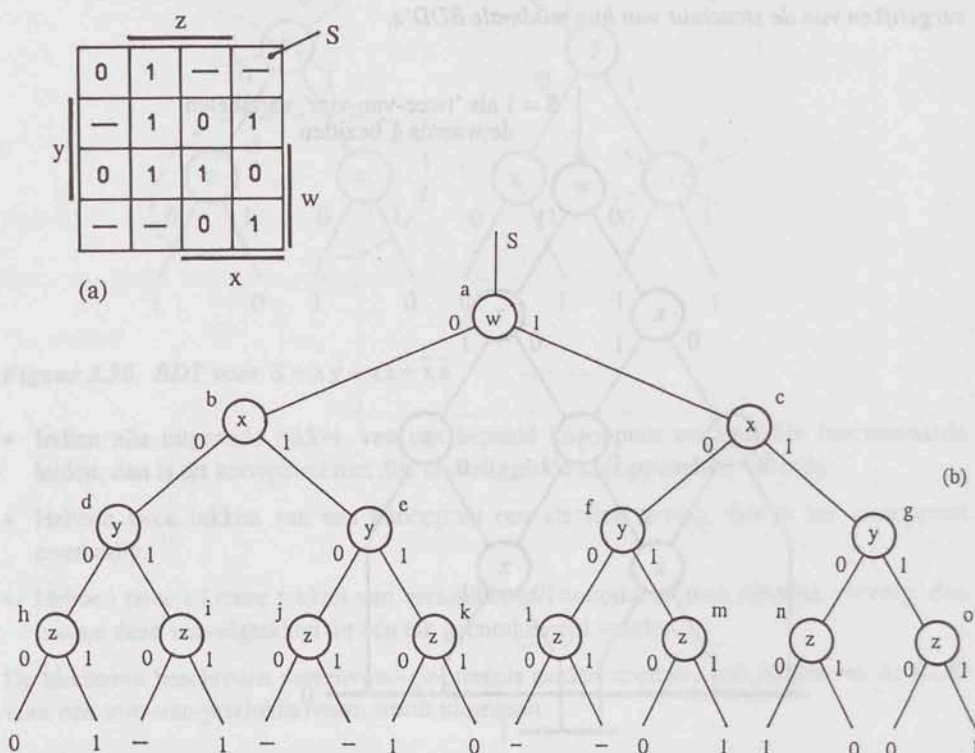
In logische functies komen vaak don't care functiewaarden voor. Deze zijn gemakkelijk in BDD's te verwerken door uit te gaan van drie functiewaarden. Zie figuur 3.38.

Het opstellen van een minimale BDD uit een complete BDT is, uiteraard, voor incompleet gespecificeerde functies lastiger dan voor een compleet gespecificeerde logische functie. De minimale oplossing is namelijk *niet meer uniek*. De zoekruimte wordt dus (veel) groter. Er zijn twee heuristische benaderingen mogelijk en wel de volgende.

- Probeer van onderaan af het aantal knooppunten te verminderen door, indien nodig, don't cares in te vullen en knooppunten aan elkaar gelijk te maken. Daarna kunnen ze samengenomen worden. Herhaal dit voor de hogere niveaus.

- Probeer, uitgaande van knooppunten hoog in de boom, de vervolgtakken door invulling van don't cares aan elkaar gelijk te maken. Dit kan uiteraard alleen als de uitgaande takken van twee knooppunten geen conflict bevatten. Combineer die takken.

In beide gevallen geldt dat invulling van een of meer don't cares combinatiemogelijkheden voor andere takken kan verhinderen.



Figuur 3.38. Functie met don't cares.

Voorbeeld

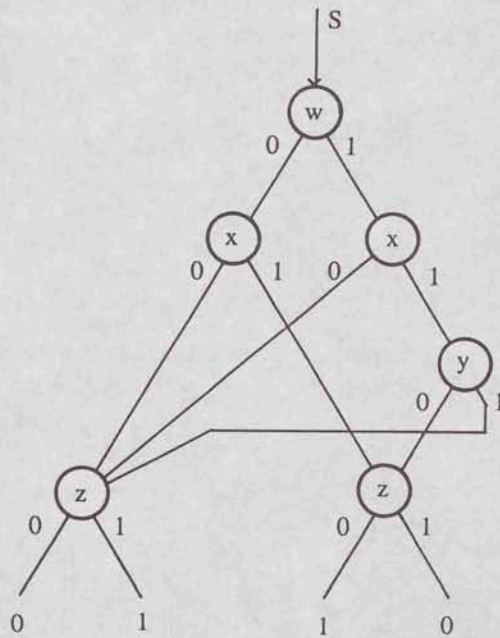
Voor de in figuur 3.38 beschreven logische functie S staat de 'minimale' BDD in figuur 3.39. (Bij deze volgorde van de variabelen!) Ga na hoe deze BDD uit figuur 3.38.b volgt. □

Er bestaan systematische methoden om 'alle' minimale BDD's voor een functie met don't cares op te stellen. Omdat het noodzakelijke mathematische gereedschap hiervoor nog ontbreekt, volstaan we met het vorenstaande.

3.14 Van BDD naar som-van-produktenvorm

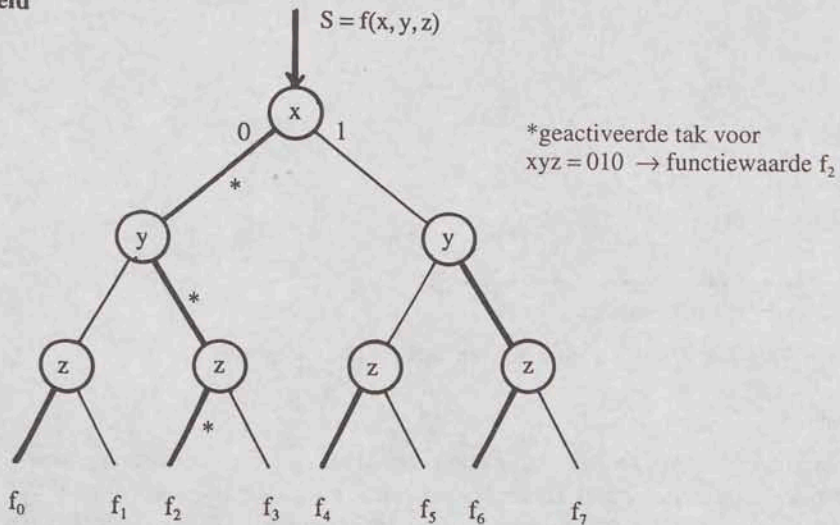
In een BDT (binary decision tree = complete boom) wordt bij één bepaalde ingangscombinatie precies één pad van ingang naar uitgang geactiveerd. Alle andere paden

zijn dan geblokkeerd. Dit verschijnsel berust op de eigenschappen van de met de boom corresponderende mintermvorm (paragraaf 2.5). Uit een BDT is de *mintermvorm* daarom eenvoudig af te lezen. Volg daartoe slechts voor alle functiewaarden $f_i = 1$ het pad, dat van de functiewaarde teruggaat naar de ingang.



Figuur 3.39. Minimale BDD na invulling don't cares.

Voorbeeld



Figuur 3.40. Geactiveerd pad in een BDT.

□

Een BDD is een gereduceerde BDT. Het zal geen verbazing wekken dat een bepaalde ingangscombinatie precies één pad van ingang naar uitgang activeert [Bryant, 1986]. Zo is uit een (minimale) BDD een (minimale) som-van-produktenvorm af te lezen, met voor elke te activeren tak van ingang naar een uitgang met functiewaarde 1 één produkt. Deze produkten overlappen elkaar niet! De som-van-produktenvorm kan dus produkten bevatten, die een of meer variabelen te veel bezitten.

Voorbeeld

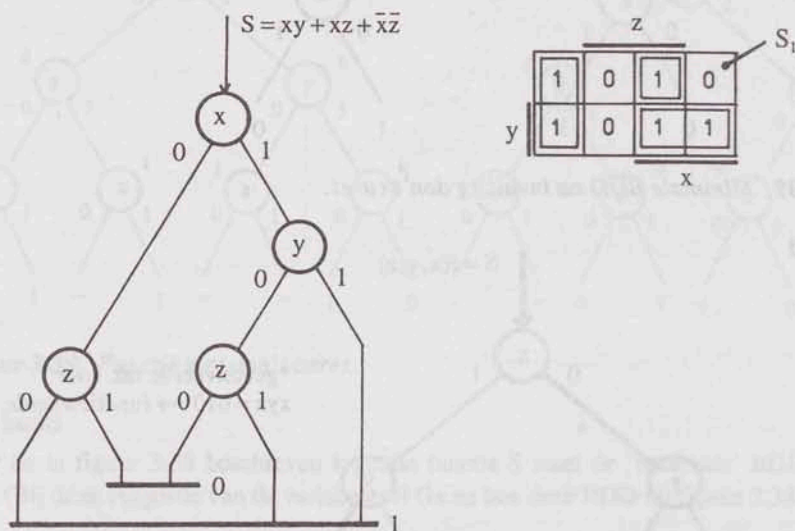
Voor de BDD in figuur 3.41.a (= figuur 3.35) vinden we:

$$S_1 = \bar{x}\bar{z} + xy + x\bar{y}z \quad (3.36)$$

Oorspronkelijk was de functie

$$S_2 = \bar{x}\bar{z} + xy + xz \quad (3.37)$$

Zetten we (3.36) in een Karnaughdiagram, dan zien we de reden van het verschil. Zie figuur 3.41.b. De verschillen tussen (3.36) en (3.37) worden verklaard doordat in een BDD paden niet gelijktijdig actief zijn. $\square$



Figuur 3.41. BDD met 'minimale' som-van-produktenvorm.

Conclusie

Uit een minimale BDD volgt een dekking van de erin beschreven logische functie met een 'minimaal' aantal *wederzijds exclusieve produktermen*. (We hebben minimaal in de vorige zin als 'minimaal' aangeduid, omdat bij een andere volgorde van de variabelen een kleinere BDD gevonden kan worden.) Voor meer informatie zie [Jacobi, 1992]. $\square$

Literatuur

1. S. B. Akers, *Binary Decision Diagrams*, IEEE Trans. on Computers, Vol. C-27, 1978, pp. 509-516.
2. T. C. Bartee, *Computer Design of Multiple-Output Logical Networks*, IRE Trans. on Electronic Computers, Vol. EC-10, 1961, pp. 21-30.
3. N. N. Biswas, *Minimization of Boolean Functions*, IEEE Trans. on Computers, Vol. C-20, 1971, pp. 925-929.
4. N. N. Biswas, *Computer-Aided Minimization Procedure for Boolean Functions*, IEEE Trans. on Computer-Aided Design, Vol. CAD-5, 1986, pp. 303-304.
5. Y. Breitbart and K. Vairavan, *The Computational Complexity of a Class of Minimization Algorithms for Switching Functions*, IEEE Trans. on Computers, Vol. C-28, 1979, pp. 941-943.
6. R. E. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Trans. on Computers, Vol. C-35, 1986, pp. 677-691.
7. D. L. Dietmeyer, *Logic Design of Digital Systems*, Allyn & Bacon, Boston, 1978.
8. S. J. Friedman and K. J. Supowit, *Finding the Optimal Variable Ordering for Binary Decision Diagrams*, Proceedings 24th ACM/IEEE Design Automation Conference, 1987, pp. 348-355.
9. M. Fujita, Y. Matsunaga, and T. Kakuda, *On Variable Ordering of Binary Decision Diagrams for the Application of Multi-level Logic Synthesis*, Proceedings IEEE ?, 1991, pp. 50-54.
10. J. F. Gimpel, *A Reduction Technique for Prime Implicant Tables*, IEEE Trans. on Electronic Computers, Vol. EC-14, 1965, pp. 535-541.
11. B. B. Gordon, R. W. House, A. P. Lechler, L. D. Nelson, and T. Rado, *Simplification of the Covering Problem for Multiple Output Logical Networks*, IEEE Trans. on Electronic Computers, Vol. EC-15, 1966, pp. 891-897.
12. B. Gurunath and N. N. Biswas, *An Algorithm for Multiple Output Minimization*, IEEE Trans. on Computer-Aided Design, Vol. 8, 1989, pp. 1007-1013.
13. R. P. Jacobi and A-M. Trullemans, *Generating Prime and Irredundant Covers for Binary Decision Diagrams*, Proceedings EDAC 1992, pp. 104-108.
14. M. Karnaugh, *The Map Method for Synthesis of Combinational Logic Circuits*, Trans. AIEE, Vol. 72, part I, 1953, pp. 593-598.
15. C. Lee, *Representation of Switching Circuits by Binary Decision Diagrams*, Bell Syst. Tech. Journal, Vol. 38, 1959, pp. 985-999.
16. E. J. McCluskey, *Minimization of Boolean Functions*, Bell Syst. Tech. J., Vol. 35, 1956, pp. 1417-1444.
17. E. J. McCluskey, *Logic Design Principles*, Prentice-Hall, Englewood Cliffs, N.J., 1986.
18. T. H. Mott, *Determination of the Irredundant Normal Forms of a Truth Function by Iterated Consensus of the Prime Implicants*, IEEE Trans. on Electronic Computers, Vol. EC-9, 1960, pp. 245-252.
19. H. T. Nagle, B. D. Carroll, and J. D. Irwin, *An Introduction to Computer Logic*, Prentice-Hall, Englewood Cliffs, N.J., 1975.
20. S. R. Petrick, *A Direct Determination of the Irredundant Form of a Boolean Function from the Set of Prime Implicants*, Computation Laboratory of the AFCRC, Bedford, Mass., TR-5-110, April 1956.
21. W. V. Quine, *The Problem of Simplifying Truth Functions*, Am. Math. Monthly, Vol. 59, 1952, pp. 521-531.
22. W. V. Quine, *A Way to Simplify Truth Functions*, Am. Math. Monthly, Vol. 62, 1955, pp. 627-631.
23. V. T. Rhyne, P. S. Noe, M. H. McKinney, and U. W. Pooch, *A New Technique for the Fast Minimization of Switching Circuits*, IEEE Trans. on Computers, Vol. C-26, 1977, pp. 757-764.
24. S. U. Robinson and R. W. House, *Gimpel's Reduction Technique Extended to the Covering Problems with Costs*, IEEE Trans. on Electronic Computers, Vol. EC-16, 1967, pp. 509-514.
25. J. R. Slagle, C-L. Chang, and R. C. T. Lee, *A New Algorithm for Generating Prime Implicants*, IEEE Trans. on Computers, Vol. C-19, 1970, pp. 304-310.
26. P. Tison, *Generalization of Consensus Theory and Application to the Minimization of Boolean Functions*, IEEE Trans. on Electronic Computers, Vol. EC-16, 1967, pp. 446-456.
27. S. Unger, *The Essence of Logic Circuits*, Prentice-Hall, Englewood Cliffs, N.J., 1989.
28. E. W. Veitch, *A Chart Method for Simplifying Truth Functions*, Proc. ACM, Pittsburgh, May 1952, pp. 127-133.

Opgaven hoofdstuk 3

3.1. Twee hokjes in een Karnaughdiagram heten aangrenzend (adjacent) als de ermee corresponderende mintermen in slechts één variabele verschillen. In een Karnaughdiagram voor n variabelen heeft elk hokje

- afhankelijk van de positie in het diagram 4, 3 of 2 aangrenzende hokjes.
- $n - 1$ aangrenzende hokjes.
- n aangrenzende hokjes.
- $n + 1$ aangrenzende hokjes.

3.2. Wanneer de logische functie S ,

$$S = \bar{w}x\bar{y} + \bar{w}xz + w\bar{x}y + w\bar{x}z + wyz + xyz$$

in een Karnaughdiagram voor vier variabelen geplaatst wordt, dan staan hierin

- zes enen.
- zeven enen.
- acht enen.
- meer dan acht enen.

3.3. Van de in het Karnaughdiagram $\downarrow$ gespecificeerde logische functie S bevat de eenvoudigste som-van-produktenvorm

- vier termen met in totaal negen variabelen/letters.
- vijf termen met in totaal elf variabelen.
- zes termen met in totaal dertien variabelen.
- zes termen met in totaal veertien variabelen.

$\uparrow$

	z				
	1	1	1	1	S
y	1	1	1	0	
	1	0	1	1	
	1	1	1	0	w
					x

	z				
	-	-	1	0	S
y	1	1	1	0	
	-	0	1	1	
	0	1	-	0	w
					x

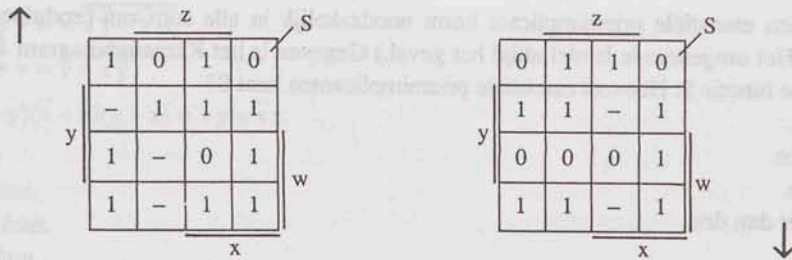
$\downarrow$

3.4. Van de in het Karnaughdiagram $\uparrow$ gespecificeerde logische functie S bevat de eenvoudigste som-van-produktenvorm

- drie termen met in totaal zes variabelen/letters.
- vier termen met in totaal negen variabelen.
- vier termen met in totaal tien variabelen.
- vier termen met in totaal elf variabelen.

3.5. Van de in het Karnaughdiagram $\downarrow$ gespecificeerde logische functie S bevat de eenvoudigste som-van-produktenvorm

- vier termen met in totaal negen variabelen/letters.
- vier termen met in totaal tien variabelen.
- vijf termen met in totaal elf variabelen.
- vijf termen met in totaal twaalf of meer variabelen.

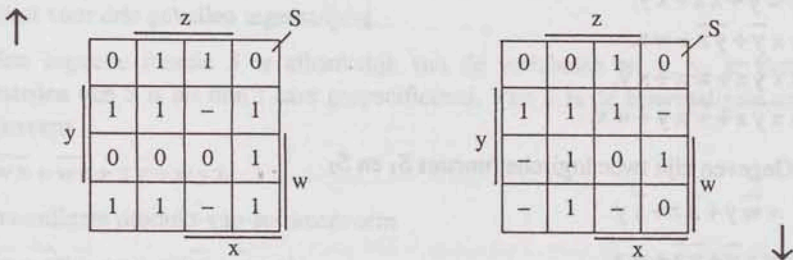


3.6. Van de in het Karnaughdiagram $\uparrow$ gespecificeerde logische functie S bestaat/bestaan van de eenvoudigste som-van-produktenvorm

- één vorm.
- twee vormen met een gelijk aantal variabelen/letters.
- drie vormen met een gelijk aantal variabelen.
- vier vormen met een gelijk aantal variabelen.

3.7. Een priemterm of priemimplicant van een functie S is een tot S behorende produktterm, die niet verder met andere produkttermen kan worden gecombineerd tot een produktterm in minder variabelen. In het proces van samennemen mogen don't cares benut worden. De verzameling van priemimplicanten vormt de basis van waaruit de minimale som-van-produktenvorm kan worden bepaald. Het Karnaughdiagram $\downarrow$ specificeert een logische functie S , met don't cares. De verzameling van alle priemimplicanten voor de gespecificeerde functie bevat

- vier priemimplicanten.
- vijf priemimplicanten.
- zes priemimplicanten.
- zeven priemimplicanten.

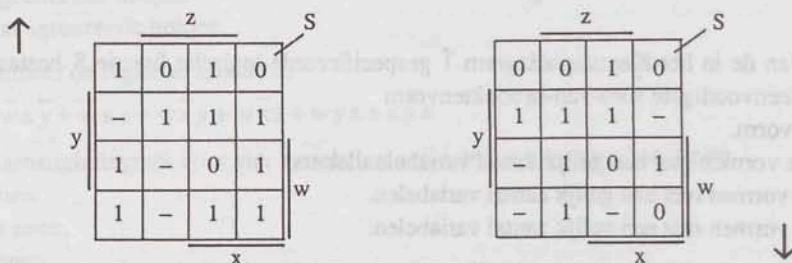


3.8. Van de in het Karnaughdiagram $\uparrow$ gespecificeerde logische functie S wordt de eenvoudigste som-van-produktenvorm bepaald. Van de eenvoudigste som-van-produktenvorm bestaan in het algemeen verschillende vormen. Zo ook bij de functie S . Hoeveel van de priemimplicanten komen in alle minimale som-van-produktenvormen voor?

- Geen.
- Een.
- Twee.
- Drie.

3.9. Een essentiële priemimplicant komt noodzakelijk in alle som-van-produktenvormen voor. (Het omgekeerde is niet altijd het geval.) Gegeven is het Karnaughdiagram ↓ van een logische functie S. Hoeveel essentiële priemimplicanten kent S?

- Een.
- Twee.
- Drie.
- Meer dan drie.



3.10. Van de in het Karnaughdiagram ↑ gespecificeerde logische functie S bevat de eenvoudigste produkt-van-sommenvorm

- drie factoren met in totaal acht variabelen/letters.
- drie factoren met in totaal negen variabelen.
- vier factoren met in totaal negen variabelen.
- zeven factoren met in totaal 28 variabelen.

3.11. De negatie $\bar{S}$ van de logische functie S,

$$S = \bar{w}y + \bar{x}y + \bar{x}z$$

in de eenvoudigste som-van-produktenvorm is

- $\bar{S} = \bar{w}y + \bar{x}z + \bar{x}y.$
- $\bar{S} = x\bar{y} + y\bar{z} + wx.$
- $\bar{S} = \bar{x}\bar{y}\bar{z} + wx + x\bar{y}.$
- $\bar{S} = x\bar{y}\bar{z} + \bar{w}x\bar{y} + wx.$

3.12. Gegeven zijn twee logische functies S_1 en S_2

$$S_1 = \bar{w}y + \bar{x}z + \bar{x}y$$

$$S_2 = x\bar{y} + y\bar{z} + wx$$

Bepaal de logische functie $S = S_1 \oplus S_2$ met behulp van Karnaughdiagrammen. In het Karnaughdiagram van S staan dan

- zes enen.
- acht enen.
- negen enen.
- een ander aantal dan de genoemde aantallen enen.

3.13. Van de volgende vier gelijkheden

$$xy + xz + yz = \overline{\overline{xy} + \overline{xz} + \overline{yz}}$$

$$xy + vw = \overline{\overline{xy} + \overline{vw}}$$

$$xy + v = \overline{\overline{v} + \overline{xy}}$$

$$(x + y)(\overline{x} + z)(y + z) = xy + \overline{x}z$$

is/zijn er

- een fout.
- twee fout.
- drie fout.
- vier fout.

3.14. De werking van een apparaat hangt af van de momentele waarde van vier signalen w t/m z. Hiervoor is gespecificeerd:

$$\overline{w}\overline{x} = 1 \quad \rightarrow \quad \text{actie 1 instellen;}$$

$$yz = 1 \quad \rightarrow \quad \text{actie 2 instellen;}$$

$$wx\overline{z} = 1 \quad \rightarrow \quad \text{actie 3 instellen;}$$

$$w\overline{y} = 1 \quad \rightarrow \quad \text{actie 4 instellen;}$$

$$\overline{w}x\overline{y} + \overline{w}x\overline{z} = 1 \quad \rightarrow \quad \text{actie 5 instellen.}$$

Onderzoek deze specificatie met behulp van een Karnaughdiagram. Daaruit volgt:

- Voor alle combinaties van w t/m z is de werking gespecificeerd. De specificatie bevat geen tegenstrijdigheden.
- Voor één combinatie van w t/m z is de werking niet gespecificeerd. De specificatie is daarnaast voor twee gevallen tegenstrijdig.
- Voor één combinatie van w t/m z is de werking niet gespecificeerd. De specificatie is daarnaast voor drie gevallen tegenstrijdig.
- Voor twee combinaties van w t/m z is de werking niet gespecificeerd. De specificatie is daarnaast voor drie gevallen tegenstrijdig.

3.15. Een logische functie S is afhankelijk van de variabelen w, x, y, z. Een aantal functiewaarden van S is als don't care gespecificeerd. Van S is de eenvoudigste som-van-produktenvorm

$$S = \overline{w}\overline{x} + \overline{w}\overline{z} + x\overline{y} + w y z$$

en de eenvoudigste produkt-van-sommenvorm

$$S = (\overline{x} + \overline{z})(\overline{w} + x + y)(\overline{w} + \overline{y} + z)$$

Hoeveel don't cares bevat deze specificatie minimaal?

- Geen don't cares.
- Een of meer don't cares.
- Twee of meer don't cares.
- Drie of meer don't cares.

3.16. De logische variabelen x t/m z kunnen in alle combinaties van waarden voorkomen. De logische functies S, T en U zijn uitgedrukt in x, y en z. S en U zijn gespecificeerd in Karnaughdiagrammen ↓.

		z			
		0	1	0	1
y		1	0	1	0
		x			

		z			
y					
		x			

		z			
		-	-	1	1
y		1	0	-	0
		x			

Bepaal het Karnaughdiagram van T zodanig dat voor U geldt:

$$U = S + T$$

Hoeveel don't cares bevat dan het Karnaughdiagram van T maximaal?

- Drie don't cares.
- Vier don't cares.
- Vijf don't cares.
- Een ander aantal don't cares.

3.17. Bij een examen wordt gevraagd een gegeven logische functie in een Karnaughdiagram voor vier variabelen te plaatsen. De docent vergeet bij het getekende diagram de variabelen te vermelden. Om het gemaakte werk snel te kunnen nakijken tekent de docent het betreffende Karnaughdiagram met de functie erin van tevoren. Hoeveel diagrammen moet de docent tekenen om het werk van elke student te kunnen nakijken?

- Vier.
- Acht.
- 16.
- Meer dan 16.

3.18. Over de in het Karnaughdiagram gespecificeerde logische functie S

		z			
		0	-	1	0
y		-	0	1	1
		-	0	1	1
		0	1	-	-
		x			

worden vier uitspraken gedaan. Een ervan is niet juist. Welke?

- Voor de eenvoudigste vormen van de functies $S_1 = S$ en $S_2 = S$ geldt: $\bar{S}_2 = S$.
- De eenvoudigste produkt-van-sommenvorm bevat in totaal zes letters/variabelen (die van 'S = ' niet meegerekend).
- De eenvoudigste produkt-van-sommenvorm en de eenvoudigste som-van-produkten-vorm bevatten evenveel variabelen/letters.
- Alle don't cares mogen vrij als 0 of als 1 gekozen worden. Een verschillende keuze voor elk van de don't cares is daarbij toegestaan.

3.19. Bij gelegenheid van een sportuitwisseling moet een delegatie van drie personen samengesteld worden. Deze personen doen aan de volgende sporten:

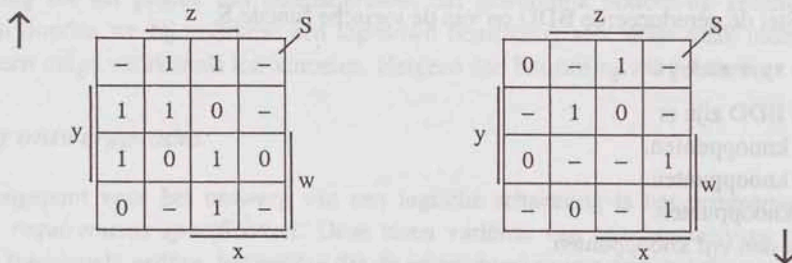
- A doet aan turnen en hockey.
 B doet aan zwemmen, turnen en hardlopen
 C doet aan hockey, schaatsen en tennis.
 D doet aan hockey, zwemmen en tennis.
 E doet aan hardlopen, schaatsen en tennis.

Op hoeveel manieren kan een delegatie van precies drie personen samengesteld worden uit de groep A t/m E, zodanig dat alle sporten tenminste één maal vertegenwoordigd zijn?

- a. Drie.
 b. Vijf.
 c. Zes.
 d. Twaalf.

3.20. Bepaal van de in het Karnaughdiagram $\downarrow$ gespecificeerde logische functie S (met de tabelmethode van Q-M) de verzameling van alle priemimplicanten. Bij functies in vier variabelen worden dan in (maximaal) vier rondes de priemimplicanten gevonden. Voor de gespecificeerde functie S worden in de achtereenvolgende rondes de volgende aantallen priemimplicanten gevonden (dubbele niet meetellen):

- a. 0 / 2 / 5 / 0.
 b. 0 / 12 / 5 / 0.
 c. 7 / 12 / 5 / 0.
 d. Andere aantallen dan genoemd.



3.21. Stel van de in het Karnaughdiagram $\uparrow$ gespecificeerde logische functie S de complete dekkingstabel op. In deze tabel staan x kruisjes en vinden we y minimale som-van-produktenvormen. (Het criterium is: minimaal aantal produkten.) Dan geldt:

- a. $x = 9$ en $y = 6$.
 b. $x = 9$ en $y = 12$.
 c. $x = 11$ en $y = 6$.
 d. $x = 11$ en $y = 12$.

3.22. Bepaal van de in het Karnaughdiagram $\downarrow$ gespecificeerde logische functie S de dekkingstabel voor de vier mintermen met functiewaarde $f_i = 1$. Dekkingstabellen bevatten vaak groepen van (onderling) dominante rijen. Hoeveel van deze groepen zijn er?

- a. Geen.
 b. Een.
 c. Twee.
 d. Drie.

		z		
		0	1	
y	0	0	1	S
	1	1	0	
	2	0	0	
	3	0	1	
		x		

3.23. Stel de gereduceerde BDD op van een logische functie S in vier variabelen, die aangeeft dat er precies twee-uit-vier elementen geselecteerd zijn. Over deze op te stellen BDD worden drie beweringen gedaan.

- De minimale BDD bevat acht knooppunten.
- De minimale BDD bestaat uit drie niveaus.
- De volgorde van de variabelen heeft invloed op de structuur van de minimale BDD.

Hiervan zijn er

- a. geen juist.
- b. één juist.
- c. twee juist.
- d. drie juist.

3.24. Stel de gereduceerde BDD op van de logische functie S ,

$$S = xy + xz + yz$$

In deze BDD zijn er

- a. drie knooppunten.
- b. vier knooppunten.
- c. vijf knooppunten.
- d. meer dan vijf knooppunten.

3.25. Voor BDD's geldt:

- In een gereduceerde BDD kan er meer dan één tak van ingang naar uitgang actief zijn.
- Het aantal knooppunten in een gereduceerde BDD is onafhankelijk van de volgorde van de variabelen.
- BDD's in gereduceerde vorm zijn uniek, ook bij don't cares.

Van deze beweringen is/zijn er

- a. geen juist.
- b. één juist.
- c. twee juist.
- d. drie juist.

Stelling 3

Karnaughdiagrammen behoren niet alleen in het lesprogramma van Digitale Techniek voor te komen. Ook notarissen en anderen kunnen hun voordeel doen met enige kennis ervan.

4

Inleiding tot het ontwerpen van combinatorische schakelingen

4.1 Het ontwerpproces

In de vorige hoofdstukken hebben we (mathematisch) gereedschap ontwikkeld voor het ontwerpen van logische schakelingen. De hoofdstukken 4 t/m 8 behandelen het ontwerpen van *combinatorische schakelingen*, dat zijn schakelingen zonder geheugenwerking. De op een zeker moment aangeboden ingangscombinatie legt bij combinatorische schakelingen eenduidig de uitgangswaarde(n) vast. Schakelingen met geheugenwerking, sequentiële schakelingen, bespreken we in de hoofdstukken 9 en verder.

Dit hoofdstuk is een inleiding tot het ontwerpen van combinatorische schakelingen. Bij ontwerpen vanuit een bepaalde voorkennis wordt bij voorkeur top-down gewerkt. Dit in tegenstelling tot het proces van kennisopbouw, dat gewoonlijk bottom-up geschiedt. In deze tekst houden we bij voorkeur een top-down benadering aan, maar vaak moeten we daartoe eerst enige voorkennis introduceren. Hetgeen dan bottom-up zal gebeuren.

Indeling ontwerpproces

Het uitgangspunt voor het ontwerp van een logische schakeling is het *programma van eisen*, de *requirements specification*. Deze eisen variëren van een beschrijving van het gewenste functionele gedrag, het gedrag dat de opdrachtgever van de schakeling verwacht, tot en met een overzicht van wat de opdrachtgever voor het te maken produkt (maximaal) beschikbaar wil stellen in termen van kostprijs, ontwerptijd (time-to-market), IC oppervlak, tijd voor het uitvoeren van bewerkingen (performance), enz. In het daarop volgende *top-down ontwerpproces* onderscheidt men gewoonlijk een drietal *hoofd fasen* [Blaauw, 1976; Dinklo, 1994]:

- de architectuur,
- de implementatie,
- de realisatie.

De *architectuur* legt vast wat de te ontwerpen schakeling moet kunnen. Het resultaat is een al dan niet formele beschrijving van het *uitwendige functionele gedrag* van de schakeling en wel vanuit het *standpunt van de gebruiker*. Het probleem wordt daarbij vaak afgebeeld op een blokschema. De functioneel gedefinieerde blokken moeten samen het gevraagde gedrag kunnen realiseren. In de digitale techniek is het eindprodukt van de architectuur meer en meer een formele beschrijving in de specificatietaal VHDL [Navabi, 1993].

Het uitwerken van de architectuur in een voorstel voor het logisch ontwerp van de schakeling duiden we aan met *implementatie*. Het in eerste instantie nog zeer globaal gespecificeerde ontwerp wordt in een aantal stappen nader uitgewerkt en afgebeeld op een logisch ontwerp. Zo'n *logisch ontwerp* bestaat uit logische operatoren, geheugenelementen en verbindingen. Tijdens het uitwerken van het logisch ontwerp wordt geprobeerd zo veel mogelijk *structuur* in de probleemstelling te herkennen en/of aan te brengen, om deze structuur vervolgens te benutten bij de afbeelding op de hardware.

De laatste fase van het ontwerpproces duiden we aan met *realisatie*. Hierbij wordt het logische ontwerp in detail uitgewerkt en omgezet in de definitieve schakeling. Dit kan een echte proefschakeling zijn, of een computersimulatie van het ontwerp. Na keuze van de technologie (bij IC's) kunnen we de fysische eigenschappen bepalen, waaronder het benodigde oppervlak en het vermogen, de bewerkingssnelheid en parameterwaarden voor de propagatietijden van signalen. Vervolgens komt de hamvraag: Is dit het eindproduct zoals bedoeld in de requirements specification?

Deze indeling van het ontwerpproces in architectuur, implementatie en realisatie dient om structuur en volgorde aan te brengen in het ontwerpproces. Het heeft weinig zin om te discussiëren over allerlei technische details als er nog geen concreet ontwerp is. Vandaar dat in de *ontwerpactiviteiten* ook een duidelijke *volgorde* gewenst is.

De scheiding tussen de ontwerp-niveaus is uiteraard vaag. De niveaus gaan geleidelijk in elkaar over. Het hangt ook van de individuele ontwerper af waar deze de scheiding wenst te leggen. De ontwerper is daar in principe vrij in. Anders is het als verschillende ontwerpteams zich met het ontwerp bezighouden. Dan is er *vooraf* een duidelijke scheiding nodig, met afspraken over wie wat doet en hoe het tussenresultaat moet worden aangeboden. Hetzelfde geldt als van software tools gebruik gemaakt wordt.

Het ontwerptraject van een opteller

We lichten het vorenstaande toe aan de hand van een globaal overzicht van de activiteiten in het ontwerptraject van een optelschakeling. We kiezen dit voorbeeld omdat enerzijds de functie van de opteller eenvoudig te omschrijven is en anderzijds omdat de opeenvolgende stappen van het ontwerpproces gemakkelijk in detail kunnen worden toegelicht.

Bij de *architectuur van de opteller* geven we aan wat de opteller moet kunnen. Zo is in de architectuur gespecificeerd:

- het formaat van de op te tellen getallen (integers, reals, etc.) en het aantal tegelijk op te tellen getallen;
- de codering van de operanden (binair/two's complement) en de gewenste codering van het resultaat;
- wat er dient te gebeuren bij capaciteitsoverschrijding.

Kortom, de architectuur specificeert welke (functionele) eigenschappen de opdrachtgever van de opteller verwacht. (Merk op dat bijvoorbeeld de interne codering van de operanden en tussenresultaten niet in de architectuur vastgelegd is.)

Nadat de definitieve architectuur vastgesteld is volgt de *implementatie*. Dit deel van het ontwerpproces begint met de keuze van het optelalgoritme en eindigt met een logisch ontwerp van de schakeling. In termen van het optellerontwerp:

- het opstellen van het optelalgoritme (wiskundigen deden dit reeds) en de keuze van de interne codering van de operanden/tussenresultaten;
- het herkennen van de repeterende structuur van de bewerkingen in het optelalgoritme;
- het afbeelden van de (repeterende) deelbewerking op een regelmatige cellulaire structuur;
- het logisch ontwerp van de cel (full adder), waaruit de opteller kan worden samengesteld;
- het aanbrengen van voorzieningen voor capaciteitoverschrijding, etc.

Een goed voorbeeld van een *implementatieconflict* is besproken in de paragrafen 1.8 en 1.9. De daarin besproken teken-en-modulus representatie is lastig te implementeren. Het two's complement is veel geschikter.

Indien de beschikbare tijd voor de optelling erg kritisch is, dan zal tijdens de implementatie een schatting gemaakt moeten worden van de definitieve opteltijd. Indien nodig kan er dan in een vroeg stadium voor een meer parallel werkend algoritme gekozen worden. Zie hoofdstuk 7. De ontwerper ontkomt er vaak niet aan om van tijd tot tijd een blik vooruit te werpen naar de technische mogelijkheden.

De *realisatie* omvat de volgende stappen:

- de keuze van de technologie en aanpassingen van het ontwerp aan de gekozen technologie;
- het opstellen van een voorlopige kostenschatting in termen van oppervlak (IC's), dissipatie en snelheid;
- de generatie van een layout;
- de bepaling, vanuit de layout door middel van extractie, van de parameters van het ontwerp met betrekking tot snelheid, oppervlak en dissipatie van de cellen;
- het hieruit bepalen van de eigenschappen van de n-bit opteller.

In dit laatste deel van het ontwerptraject spelen simulaties een belangrijke rol. Het is niet nodig om een IC daadwerkelijk te fabriceren om de eigenschappen ervan te kunnen schatten.

Door alle fasen van het ontwerpproces heen speelt het probleem van de *ontwerpverificatie*: is het tot op dat moment verkregen (voor)ontwerp nog correct? Op architectuurniveau omvat het verificatieproces onder andere het controleren van de randvoorwaarden van het algoritme. Bij optellen moet daarbij nagaan worden welk bereik voor het resultaat gereserveerd moet worden om geen capaciteitoverschrijding te krijgen. Op het implementatieniveau moeten we de ontworpen logische structuur verifiëren. Is de architectuur correct afgebeeld op de logica? Ook moeten de afspraken over de interface

tussen de verschillende modules geverifieerd worden. Op het realisatieniveau moet onder andere gecontroleerd worden of logische variabelen correct zijn afgebeeld op spanningsniveaus, dat signalen tijdig beschikbaar zijn, enz. Ook moet onderzocht worden of de ontworpen schakeling voldoende *robuust* is, dat wil zeggen bestand tegen allerlei ongewenste invloeden van buiten af. Hieronder vallen afwijkingen van de veronderstelde omgevingscondities met betrekking tot temperatuur, voedingsspanningen en elektromagnetische velden. Een goed ontwerp omvat voldoende marges om een en ander op te vangen. In de praktijk bepaalt de zorgvuldigheid waarmee de ontwerpverificatie wordt uitgevoerd in belangrijke mate de kwaliteit van het eindproduct.

Tenslotte kan het optellerontwerp inclusief zijn eigenschappen en de bijbehorende verzameling van testpatronen opgenomen worden in een *cellenbibliotheek*. Zo'n opgeslagen ontwerp is daarna direct vanuit een architectuurspecificatie oproepbaar.

Scope van deze tekst

Deze tekst beperkt zich tot de hoofdlijnen van het logische deel van het ontwerptraject, het deel dat we hiervoor als implementatie hebben aangeduid. Tevens zullen we een brug slaan naar het fysische deel van het IC ontwerp- en realisatieproces door in een logisch ontwerp mogelijke knelpunten te lokaliseren en zullen we aangeven wanneer specifieke eigenschappen van de componenten verwacht worden. En dit is vaker dan waarmee veel ontwerpers rekening houden!

In de volgende paragraaf behandelen we het logisch ontwerp van een full adder, vanuit een waarheidstabel. We doorlopen daarmee het middelste deel van het implementatietraject. De architectuur en de structurele eigenschappen van optellers komen in hoofdstuk 7 uitgebreider aan bod; de technologie-aanpassingen in hoofdstuk 5 en in hoofdstuk 6.

In de rest van dit hoofdstuk lichten we delen van het ontwerptraject voor combinatorische schakelingen toe en introduceren we het bijbehorende gereedschap. We eindigen met het aangeven van de relatie tussen ontwerpen en testen.

In hoofdstuk 5 bespreken we enkele logische families. We gaan in op de vraag, hoe we de fysische eigenschappen van de componenten moeten modelleren om op logisch niveau te kunnen ontwerpen en we gaan dieper in op de randvoorwaarden daarbij. Veel nadruk ligt op de opzet van een hanteerbaar tijdmodel.

Hoofdstuk 6 bespreekt een aantal standaardoplossingen voor veel voorkomende problemen. We maken onderscheid tussen een *theoretisch logisch ontwerp* en een *realiseerbaar logisch ontwerp*. Dit laatste inclusief allerlei aanpassingen aan de technologie.

Er is reeds gezegd dat het zoeken naar *structuur in de probleemstelling* belangrijk is. Hoofdstuk 7 gaat hier dieper op in. We bespreken daarin een aantal ontwerpvoorbeelden, onder meer van optellers, waarbij de structuur ervan uitgebuit wordt om de optelling te versnellen. Ook werken we andere voorbeelden uit.

Hoofdstuk 8 tenslotte bespreekt wat achtergronden van moderne ontwerptechnieken. Hiermee is het overzicht van het ontwerptraject voor de combinatoriek compleet.

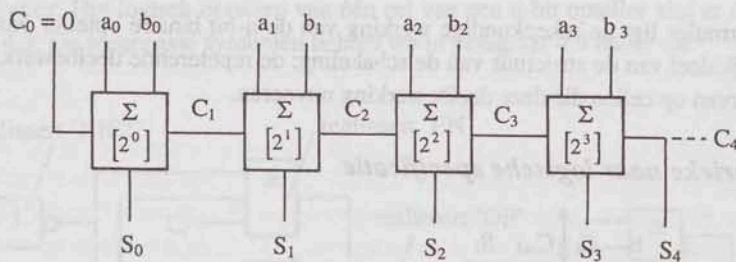
4.2 Het logisch ontwerp van een n-bit full adder

We volgen nu een gedeelte van het ontwerptraject van een opteller, die bedoeld is om twee niet-negatieve binair gecodeerde gehele getallen op te tellen. Zie paragraaf 1.7 voor de toegepaste begrippen en termen.

Uitgaande van de *potlood-en-papiermethode* voor het optellen van twee niet-negatieve binair gecodeerde gehele getallen constateren we dat optellen bestaat uit het *repetierend* toepassen van een *deelbewerking*. De deelbewerking bestaat uit de optelling van twee bits met hetzelfde gewicht, waarbij het inkomende carrybit van de vorige kolom moet worden opgeteld. De inkomende carry heeft hetzelfde gewicht als de databits. Het blokschema van een hierop gebaseerde opteller staat in figuur 4.1.a. De tabel specificeert de *numerieke werking* van één cel. Het *gewicht van de bits* wordt bij optellen impliciet verwerkt, via de positie van de betreffende cel in de rij. Zie ook de notatie-afspraken in paragraaf 1.7. Een wiskundige formulering van de werking van de cel staat in de formules (4.1) en (4.1.a):

$$a_i 2^i + b_i 2^i + C_i 2^i = C_{i+1} 2^{i+1} + S_i 2^i \quad \text{met } a_i, b_i, C_i, C_{i+1}, S_i \in \{0, 1\} \quad (4.1)$$

$$a_i + b_i + C_i = 2C_{i+1} + S_i \quad (4.1.a)$$



(a) 4-Bit full adder

$a_i (2^i)$	$b_i (2^i)$	$C_i (2^i)$	$C_{i+1} (2^{i+1})$	$S_i (2^i)$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(b) full adder tabel

Figuur 4.1. Blokschema van een 4-bit opteller.

Een n-bit opteller heeft een *beperkte capaciteit*. Bij 4-bit niet-negatieve gehele getallen ligt de numerieke som van twee getallen A en B tussen

$$0 \leq S = A + B \leq 30 \quad (4.2)$$

en het bereik van de 4-bit opteller, indien uitgedrukt in de sombits, tussen

$$0 \leq S_3 S_2 S_1 S_0_{\text{BIN}} \leq 15 \quad (4.3)$$

Er treedt *capaciteitsoverschrijding* op als de uitgaande carry $C_4 = 1$ is, tenzij we het resultaat met één bit uitbreiden tot $S_4 S_3 S_2 S_1 S_0_{\text{BIN}}$. Evenzo volstaat bij de optelling van k n -bit binaire getallen een opteller die (tenminste)

$$n + \lceil {}^2\log k \rceil \quad (4.4)$$

bits breed is. Anders is een voorziening nodig waarmee capaciteitsoverschrijding kan worden gedetecteerd.

Opmerking

Houden we rekening met het feit dat n -bit binaire getallen een bereik hebben tussen 0 en $2^n - 1$, dan vinden we voor de bovengrens formule (4.4.a):

$$\lceil {}^2\log(k \cdot (2^n - 1) + 1) \rceil \quad (4.4.a)$$

In de praktijk wordt met formule (4.4) gerekend. □

Met deze formules ligt de rekenkundige werking van de n -bit binaire opteller vast en ook een belangrijk deel van de structuur van de schakeling: de repeterende deelbewerking en de afbeelding ervan op cellen die deze deelbewerking uitvoeren.

Van numerieke naar logische specificatie

a_i	b_i	C_i	C_{i+1}	S_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

a_i, b_i, C_i, C_{i+1} en $S_i \in \{0, 1\}$

Tabel 4.1. Waarheidstabel van één optelsectie (full adder).

De eerste stap van het logisch ontwerp is de overgang van *numerieke nullen en enen* naar *logische nullen en enen*. Het ligt voor de hand de afbeelding zo te kiezen dat

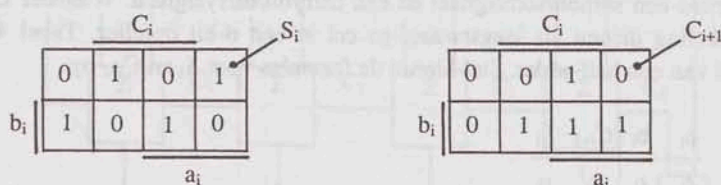
numerieke 0 $\leftrightarrow$ logische 0

numerieke 1 $\leftrightarrow$ logische 1

Aldus ontstaat uit de numerieke specificatie in figuur 4.1 de waarheidstabel 4.1. Uit tabel 4.1 volgen de Karnaughdiagrammen van figuur 4.2 en de formules (4.5) en (4.6) voor het sombit en het carrybit:

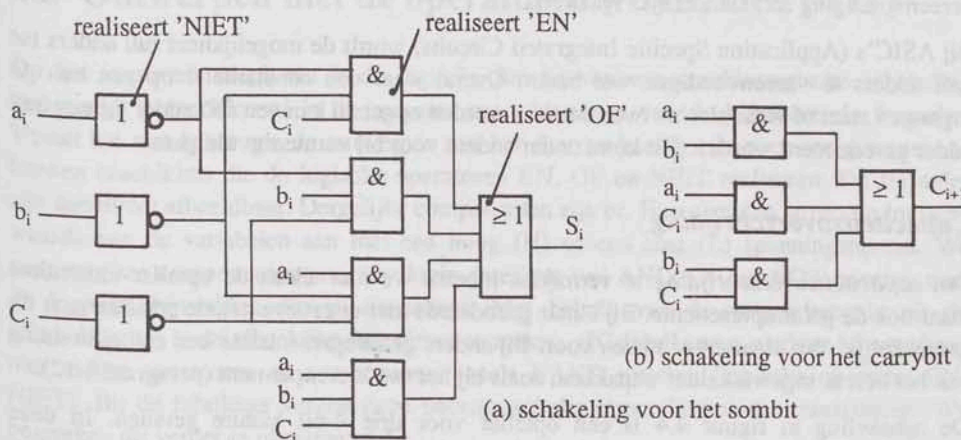
$$S_i = \bar{a}_i \bar{b}_i C_i + \bar{a}_i b_i \bar{C}_i + a_i \bar{b}_i \bar{C}_i + a_i b_i C_i \quad (4.5)$$

$$C_{i+1} = a_i b_i + a_i C_i + b_i C_i \quad (4.6)$$



Figuur 4.2. Karnaughdiagrammen van S_i en C_{i+1} .

De formules (4.5) en (4.6) zijn uitgedrukt in de operatoren EN, OF en NIET. We nemen voorlopig aan dat er componenten bestaan die deze bewerkingen op hun ingangssignalen kunnen uitvoeren. Elk ingangssignaal stelt daarbij de waarde van de betreffende logische variabele voor. Het logisch ontwerp van één cel van een n-bit opteller ziet er dan uit zoals in figuur 4.3. De toegepaste symbolen lichten we in paragraaf 4.9 nader toe.



Figuur 4.3. Logisch ontwerp van een optelsectie.

Nu we een eerste logisch ontwerp hebben rijst de vraag of dit het ontwerp is wat we willen. Zijn er nog alternatieve ontwerpen te bedenken en zo ja, wat zijn de voor- en nadelen van elk ontwerp? Is er bijvoorbeeld een realisatie met 'minder componenten' mogelijk? Middelen om deze vragen te beantwoorden bespreken we in de volgende paragrafen van dit hoofdstuk.

Half adder en full adder

Een schakeling die drie bits/signalen a_i , b_i en C_i met hetzelfde gewicht kan optellen en daarbij een sombit/somsignaal S_i en een carrybit/carrysignaal C_{i+1} afgeeft (volgens de betrekkingen (4.5) en (4.6)) noemen we een *full adder*.

Een *half adder* kan slechts twee bits/signalen met hetzelfde gewicht optellen. Daarbij ontstaan eveneens een sombit/somsignaal en een carrybit/carrysignaal. Wanneer $C_0 = 0$ is kan zo'n schakeling dienen als laagstwaardige cel in een n -bit opteller. Tabel 4.2 is de waarheidstabel van een half adder. Stel hieruit de formules voor S_i en C_{i+1} op.

a_i	b_i	C_{i+1}	S_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabel 4.2. Waarheidstabel van een half adder.

Bij standaard 4-bit optellers wordt deze vereenvoudigingsmogelijkheid van full adder tot half adder niet benut. Bij het cascaderen van n 4-bit optellers tot een $4n$ -bit opteller is deze vereenvoudiging zelfs hinderlijk. Waarom?

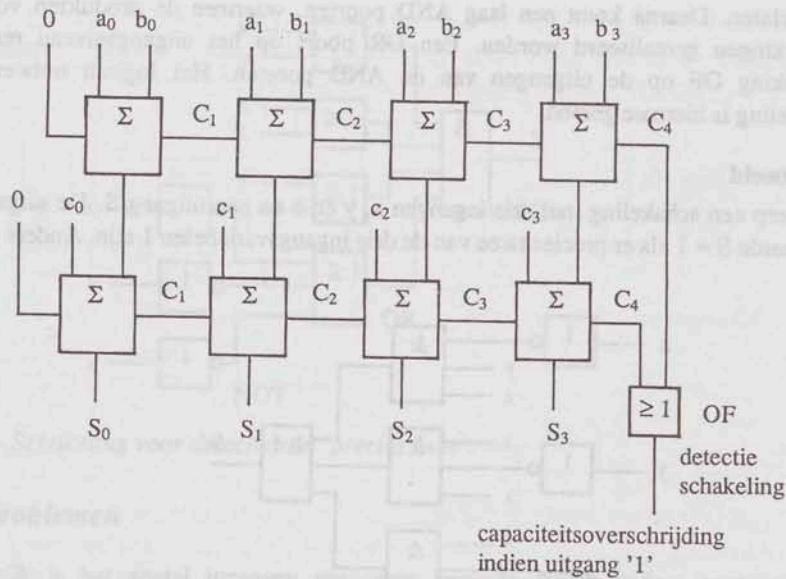
Bij ASIC's (Application Specific Integrated Circuits) wordt de mogelijkheid full adders tot half adders te vereenvoudigen wel benut. Overal waar een constante 0 op een van de ingangen staat of waar slechts twee databits worden opgeteld kan een full adder tot een half adder gereduceerd worden. Dit komt onder andere voor bij vermenigvuldigers.

Capaciteitsoverschrijding

Om *capaciteitsoverschrijding* te vermijden moeten we niet alleen de opteller uitbreiden, maar ook de getalrepresentatie. Bij binair gecodeerde niet-negatieve gehele getallen gaat dit gemakkelijk. We plaatsen er nullen voor. Bij andere getalrepresentaties kan een uitbreiding van het bereik ingewikkelder uitpakken, zoals bij het two's complement (paragraaf 1.12).

De schakeling in figuur 4.4 is een opteller voor drie 4-bit binaire getallen. In deze schakeling kan capaciteitsoverschrijding optreden, omdat de som slechts vier bits breed is. Ga na dat de toegepaste schakeling voor de detectie van capaciteitsoverschrijding werkt. Is deze detectieschakeling ook toepasbaar bij een opteller die in het two's complement werkt?

Hiermee hebben we een gedeelte van het ontwerptraject van een binaire optelschakeling doorlopen. In de volgende paragrafen lichten we telkens een onderdeel van het ontwerptraject voor combinatorische schakelingen toe. Later zullen we ook alternatieven voor het full adder ontwerp bespreken.



Figuur 4.4. Opteller met detectie van capaciteitsoverschrijding.

4.3 Ontwerpen met de operatoren {AND, OR, NOT}

Op het implementatieniveau is de logische functie van een combinatorische schakeling gewoonlijk gespecificeerd in de *som-van-produktenvorm* van de bijbehorende formules. Vanuit het standpunt van de ontwerper zou het daarom handig zijn over componenten te kunnen beschikken die de logische operatoren EN, OF en NIET realiseren. De formules zijn dan direct afbeeldbaar. Dergelijke componenten zijn er. Hun signalen geven de logische waarde van de variabelen aan met een hoog (H) of een laag (L) spanningsniveau. We noemen deze componenten *poorten* (logic gates) en wel AND, OR en NOT poorten naar de logische bewerking die zij verrichten. Met behulp van de volgorderegels uit de schakelalgebra is de afbeelding dan direct te maken. (IC fabrikanten werken bij voorkeur met andere operatoren en componenten, zoals NAND poorten (logische operatie: EN-NIET). Bij de fabricage vergen deze poorten minder oppervlak/minder transistoren. We bespreken dit verder in paragraaf 4.4.)

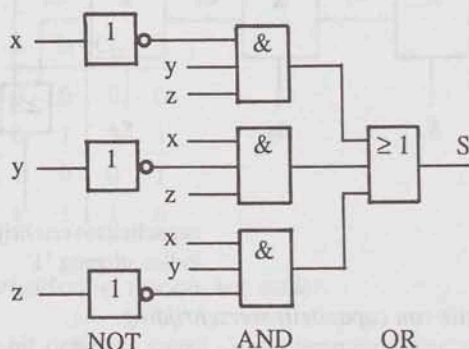
Two/three-level NOT-AND-OR schakelingen

Wanneer de logische functie van een te ontwerpen schakeling in een (minimale) som-van-produktenvorm gespecificeerd is, dan is deze formule direct afbeeldbaar met behulp van de volgorderegels van de schakelalgebra. De hoogste prioriteit heeft de bewerking NIET. Het logisch ontwerp van de schakeling begint met één niveau NOT poorten, waarmee de signaalwaarden gemaakt worden die behoren bij de complementaire variabelen. Zijn deze beschikbaar, hetgeen vaak het geval is, dan kan deze laag NOT poorten worden

weggelaten. Daarna komt een laag AND poorten, waarmee de produkten voor de EN bewerkingen gerealiseerd worden. Een OR poort op het uitgangsniveau realiseert de bewerking OF op de uitgangen van de AND poorten. Het logisch ontwerp van de schakeling is hiermee gereed.

Voorbeeld

Ontwerp een schakeling met drie ingangen x, y en z en een uitgang S. De uitgang S heeft de waarde $S = 1$ als er precies twee van de drie ingangsvariabelen 1 zijn. Anders is $S = 0$.



Figuur 4.5. Schakeling voor detectie van 'precisies twee'.

De logische functie van deze schakeling is

$$S = \bar{x}yz + x\bar{y}z + xy\bar{z} \quad (4.7)$$

Figuur 4.5 beschrijft de schakeling. □

Two/three-level NOT-OR-AND schakelingen

Bij de produkt-van-sommenvorm van de logische formule wordt met haakjes de prioriteitsvolgorde van de logische bewerkingen veranderd. De afbeelding geschiedt in deze nieuwe prioriteitsvolgorde.

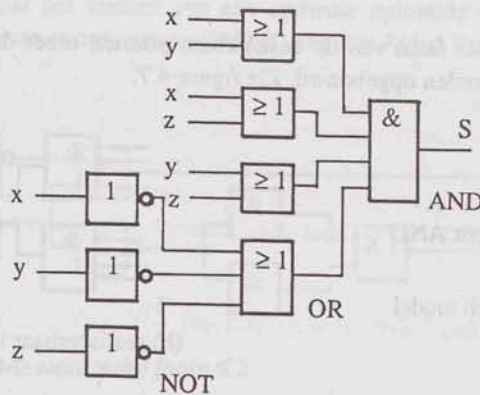
Voorbeeld

De produkt-van-sommenvorm van formule (4.7) is formule (4.8):

$$S = (x + y)(x + z)(y + z)(\bar{x} + \bar{y} + \bar{z}) \quad (4.8)$$

Figuur 4.6 beschrijft de schakeling. □

Vergelijken we de logische schakeling van figuur 4.5 met die in figuur 4.6, dan is de eerste kleiner (minder poorten/ingangen). Of we in de praktijk na realisatie in hardware ook nog voor de opzet in figuur 4.5 zullen kiezen hangt mede van de omvang en de andere eigenschappen van de componenten af.



Figuur 4.6. Schakeling voor detectie van 'precies twee'.

Passingsproblemen

In de praktijk is het aantal ingangen per poort beperkt, onder andere vanwege de elektrische eigenschappen van de transistoren. Bij standaard IC's zijn de beperkingen nog groter. Tabel 4.3 geeft aan welke uitvoeringsvormen van {AND, OR, NOT, EXOR} verkrijgbaar zijn in een bepaalde logische familie van IC's.

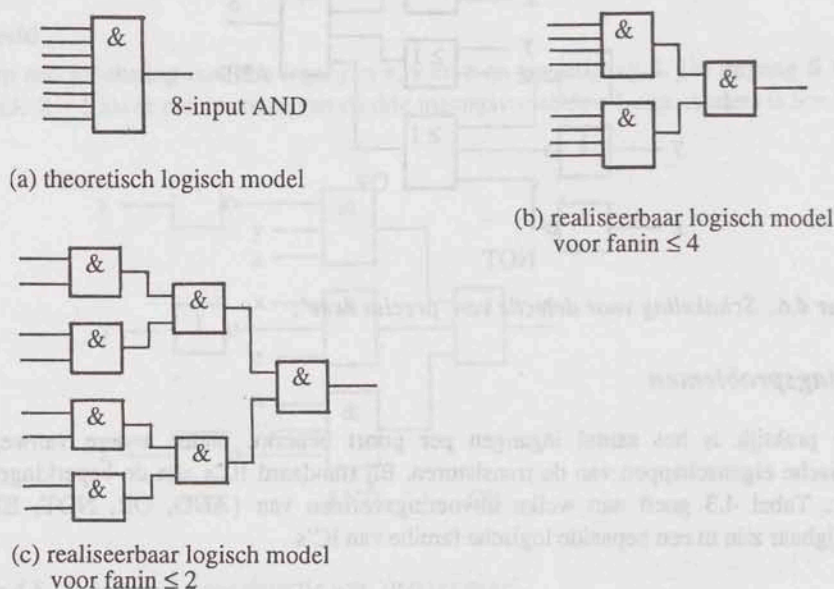
aantal ingangen	poorten per package
AND 2	4
3	3
4	2
OR 2	4
3	3
EXOR 2	4
NOT 1	6

Tabel 4.3. AND/OR/EXOR/NOT poorten per package (Philips High-Speed CMOS, 1994)

Het aantal ingangen op een poort noemen we de *fanin* van die poort. Dit begrip niet te verwarren met het begrip ingangsimpedantie/ingangsstroom van een poortingang, dat ook wel met fanin wordt aangeduid. Wanneer de beschikbare fanin (aantal ingangen) op een component kleiner is dan in het logisch model van de schakeling is toegepast, dan is een conversie van de formule noodzakelijk om de formule passend te maken op de componenten. Dit passend maken is een onderdeel van wat men wel aanduidt als 'technologie-mapping', een onderdeel van het afbeelden van een ontwerp waar het om veel meer gaat dan om het aantal poortingangen.

Voorbeeld

Afhankelijk van de maximale fanin van de beschikbare poorten moet de functie AND in één, twee of drie niveaus worden opgebouwd. Zie figuur 4.7.



Figuur 4.7. Voorbeelden van fanin aanpassingen. □

Het fanin probleem is oplosbaar door i niveaus AND en j niveaus OR poorten met beperkte fanin achter elkaar te schakelen totdat er 'poorten' met de gewenste fanin verkregen zijn. We noemen dit de $AND^i OR^j$ of $OR^i AND^j$ oplossing. Bij AND's en OR's werkt dit recept altijd, maar het resulteert niet altijd in een oplossing met een minimaal aantal poorten.

Het fanin probleem is ook op te lossen met een *decompositie van de formule* met haakjes, totdat er een vorm ontstaat die direct afbeeldbaar is op de beschikbare poorten.

Voorbeeld

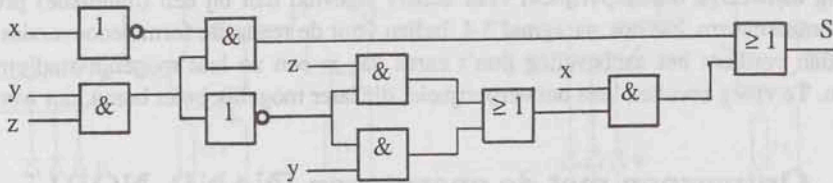
Realiseer formule (4.7) in AND/OR poorten met twee ingangen. Decompositie van de formule leidt tot

$$\begin{aligned}
 S &= \bar{x}yz + x\bar{y}z + xy\bar{z} \\
 &= \bar{x} \cdot yz + x \cdot (\bar{y}z + y\bar{z}) \\
 &= \bar{x} \cdot yz + x \cdot (\bar{y}z \cdot z + \bar{y}z \cdot y) \quad (4.9)
 \end{aligned}$$

Figuur 4.8 beschrijft het logisch ontwerp van de schakeling. □

Er zijn voor figuur 4.8 varianten te bedenken, waaronder oplossingen in vier of vijf niveaus maar met meer componenten. De afbeelding van een formule op poorten is niet uniek. Het

zal duidelijk zijn dat het zoeken van een optimale oplossing voor grotere schakelingen slechts met behulp van computers en CAD (Computer Aided Design) software kan worden uitgevoerd.



Figuur 4.8. Logisch ontwerp bij fanin ≤ 2 .

Open ingangen

Naast het probleem van te weinig ingangen bestaat ook het probleem van te veel ingangen. Soms is een ingang op een AND of OR poort gewoon over. In sommige IC's trekken niet aangesloten ingangen automatisch naar het 'neutrale' niveau, te weten '1' voor een AND of '0' voor een OR. Omdat

$$1 \cdot yz = yz$$

en

$$0 + y + z = y + z$$

lijkt het probleem van open ingangen opgelost. Toch mag men open ingangen nooit laten zweven omdat metingen aantonen dat open ingangen storingsgevoelig zijn. Om dit probleem op te lossen heeft men de keuze uit twee alternatieven:

- leid een vaste 1 of 0 af van de voedingsspanning en zet deze op de open ingang;
- sluit een ingangsvariabele twee of meer keer aan.

Logisch is er geen verschil tussen beide oplossingen, maar elektrisch wel! In het laatste geval moet het ingangssignaal meer dan één ingang aansturen. Het signaal ziet een hogere belasting, hetgeen gevolgen heeft voor de schakeltijden. Zie hoofdstuk 5.

Conclusie.

Formules, uitgedrukt in de bewerkingen EN, OF en NIET, kunnen gemakkelijk op hardware worden afgebeeld. Vaak zijn daarbij extra conversies van de formules nodig, omdat de formules niet direct op de beschikbare componenten passen. Men onderscheidt daarom verschillende ontwerp niveaus, variërend van een *theoretisch logisch model* (logic model) van de schakeling tot een *direct afbeeldbaar logisch model* (detailed logic model). Hierna komen de ontwerp niveaus waarop andere details ingevuld worden, bijvoorbeeld de afbeelding op standaard IC's en de plaatsing ervan op het printed circuit board. Ook moet het elektrisch gedrag van de schakeling geverifieerd worden. Bij het realiseren van de schakeling in een IC heeft men vaak wat meer ontwerpvrijheid dan bij toepassing van standaardcomponenten. □

Don't care signaalwaarden

Veel probleemstellingen in de digitale techniek leiden niet tot een complete en unieke logische specificatie. Bij het opstellen van een (minimale) som-van-produktenvorm wordt de nog aanwezige ontwerprijheid vaak anders ingevuld dan bij een (minimale) produkt-van-sommenvorm. Zie ook paragraaf 3.4. Indien voor de realisatie formuleconversies nodig zijn, dan verdient het aanbeveling don't cares pas in een zo laat mogelijk stadium in te vullen. Te vroeg invullen kost ontwerprijheid, die later mogelijk beter benut kan worden.

4.4 Ontwerpen met de operatoren {NAND, NOR}

De keuze voor AND, OR en NOT poorten berust op ontwerpgermak. Op het realisatieniveau sluiten deze poorten vrijwel naadloos aan bij de in de schakelalgebra toegepaste operatoren EN, OF en NIET. Vanuit de IC technologie komt de vraag of ontwerpen ook in *andere types poorten* gerealiseerd kunnen worden. Het ontwerpproces wordt dan mogelijk wat gecompliceerder, er komt een extra afbeeldingsstap van de operatoren EN t/m NIET naar de andere operatoren bij, maar omdat van IC's vaak grote aantallen gemaakt worden vallen de extra ontwerpkosten tegen de besparingen weg.

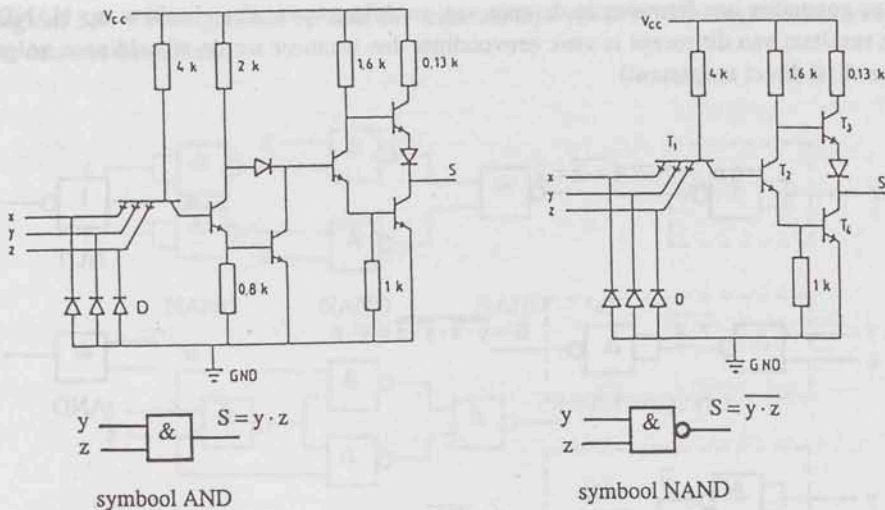
Figuur 4.9.a beschrijft de uitvoeringsschema's van twee poorten uit de bipolaire TTL technologie (TTL = Transistor-Transistor Logic). Het linkschema stelt een AND poort voor en het rechtschema een *NAND poort*. Deze laatste poort verricht een combinatie van de elementaire logische operatoren EN en NIET. Vandaar de naam NAND (NOT-AND). Een blik op de twee schema's leert dat het schema van de NAND eenvoudiger is dan dat van de AND. De vraag, of ook met de *operator NAND* een logisch ontwerp gerealiseerd kan worden, is dus een heel reële vraag. Figuur 4.9.b beschrijft dezelfde poorten in CMOS logica. Hiervoor geldt dezelfde conclusie.

In vrijwel elke andere technologie is de NAND poort eenvoudiger dan de AND poort. Dit geldt ook voor de schakelingen voor de NOR poort (NOT-OR) en de OR poort. De onderbouwing van deze uitspraken stellen we uit tot hoofdstuk 5.

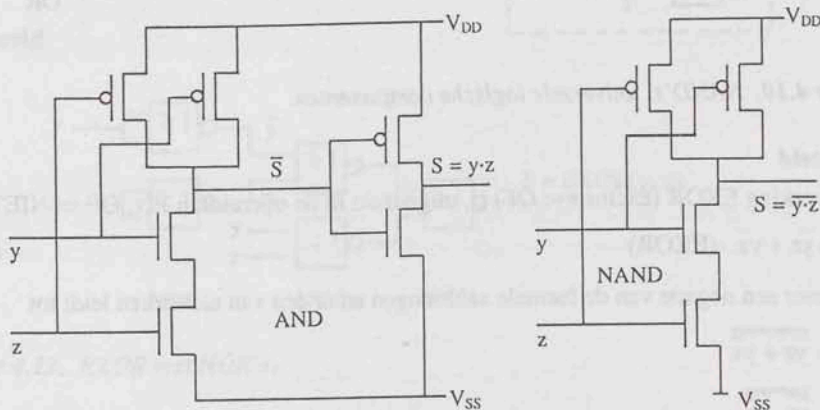
NAND en NOR, universeel toepasbare componenten

Met NAND's (NOR's) kan iedere combinatorische schakeling gerealiseerd worden. Figuur 4.10 laat zien hoe met 2-input NAND's de poorten AND, OR en NOT gerealiseerd kunnen worden. Hetzelfde geldt voor NOR poorten. We laten de verificatie hiervan aan de lezer over. Men noemt NAND's en NOR's daarom wel *universele logische componenten*.

Het *symbool* voor de NAND poort bestaat uit het symbool voor de AND poort, met aan de uitgang een cirkeltje. Dit cirkeltje heet *negatie-indicator* en geeft de negatie aan de uitgang weer. Het symbool voor de NOR poort bestaat uit het symbool voor de OR poort, gevolgd door de negatie-indicator. Zie ook paragraaf 4.8 en 4.9.



(a) AND en NAND poort in TTL logica



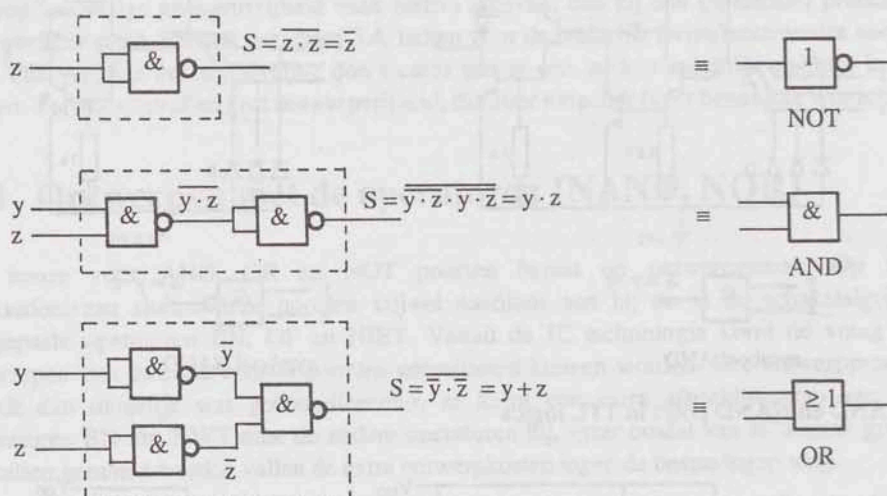
(b) AND en NAND poort in CMOS logica

Figuur 4.9. Poorten in TTL logica en in CMOS logica.

Two/three-level NAND-NAND schakelingen

Een logische formule kan eenvoudig op NAND's worden afgebeeld door uit te gaan van de som-van-produktenform van de formule. Twee keer de formule van een negatiestreep voorzien verandert het resultaat van de formule niet. Wordt vervolgens één negatie uitgewerkt, dan herkent men direct een realisatie in NAND poorten. Zijn aan de ingangen van de poorten ook genegeerde ingangsvariabelen nodig, dan kunnen deze op het ingangsniveau worden gemaakt uit de variabelen zelf. We hebben hiermee een eenvoudig

recept gevonden om formules in de som-van-produktenvorm af te beelden op NAND's. (Het resultaat van dit recept is vaak eenvoudiger dan wanneer we de afbeeldingen volgens figuur 4.10 direct toepassen!)



Figuur 4.10. NAND's, universele logische componenten.

Voorbeeld

De bewerking EXOR (exclusieve OF) is, uitgedrukt in de operatoren EN, OF en NIET:

$$S = \bar{y}z + y\bar{z} \quad (\text{EXOR}) \quad (4.10)$$

Twee keer een negatie van de formule aanbrengen en er één van uitwerken leidt tot

$$\begin{aligned} S &= \overline{\overline{y\bar{z} + y\bar{z}}} \\ &= \overline{y\bar{z} \cdot y\bar{z}} \end{aligned} \quad (4.11)$$

In NAND's uitgedrukt wordt dit:

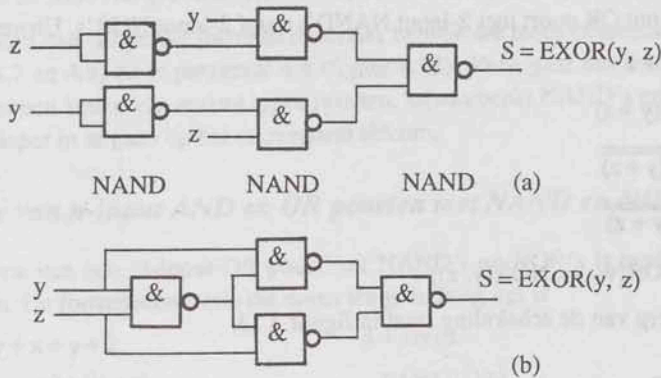
$$\begin{aligned} S &= \text{NAND}(\text{NAND}(\bar{y}, z), \text{NAND}(y, \bar{z})) \\ &= \text{NAND}(\text{NAND}(\text{NAND}(y, y), z), (\text{NAND}(y, \text{NAND}(z, z)))) \end{aligned} \quad (4.11.a)$$

Het logisch ontwerp van de schakeling staat in figuur 4.11. In figuur 4.11.b is een vereenvoudiging toegepast. Waarom mag deze vereenvoudiging? $\square$

Two/three-level NOR-NOR schakelingen

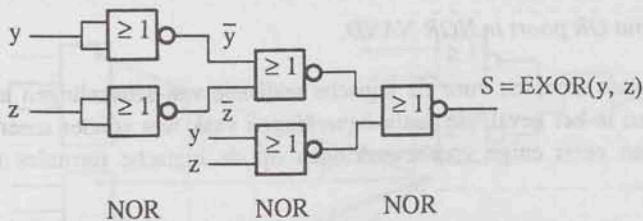
Het recept voor NOR poorten gaat uit van de *produkt-van-sommenvorm* van de formule. Hoe deze afgeleid kan worden uit de som-van-produktenvorm hebben we besproken in

paragraaf 3.4. Verder gaat het op dezelfde wijze als bij NAND's: twee negaties aanbrengen en er één van uitwerken.



Figuur 4.11. EXOR met NAND's.

Voorbeeld



Figuur 4.12. EXOR met NOR's.

De formule voor de EXOR, in de produktvorm, is:

$$\begin{aligned}
 S &= (\bar{y} + \bar{z}) \cdot (y + z) \\
 &= \overline{(\bar{y} + \bar{z}) \cdot (y + z)} \\
 &= \overline{(\bar{y} + \bar{z})} + \overline{(y + z)} \\
 &= \text{NOR}(\text{NOR}(\bar{y}, \bar{z}), \text{NOR}(y, z)) \\
 &= \text{NOR}(\text{NOR}(\text{NOR}(y, y), \text{NOR}(z, z)), \text{NOR}(y, z)) \quad (4.12)
 \end{aligned}$$

Het logisch ontwerp van de schakeling staat in figuur 4.12. □

Vergelijk de *two-level structuur* van een AND-OR realisatie met die van een NAND-NAND realisatie van dezelfde formule. Wat valt op? Hoe zit dat bij NOR's?

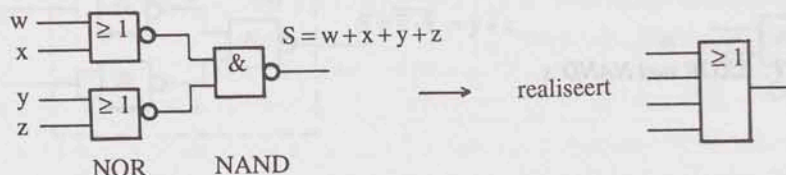
Het volgende voorbeeld laat zien dat het zinvol kan zijn NAND's en NOR's door elkaar heen toe te passen.

Voorbeeld

Realiseer een 4-input OR poort met 2-input NAND's en/of 2-input NOR's. Uitwerking:

$$\begin{aligned}
 S &= w + x + y + z \\
 &= (w + x) + (y + z) \\
 &= \overline{\overline{(w + x) + (y + z)}} \\
 &= \overline{\overline{(w + x)} \cdot \overline{\overline{(y + z)}}} \\
 &= \text{NAND}(\text{NOR}(w, x), \text{NOR}(y, z))
 \end{aligned} \tag{4.13}$$

Het logisch ontwerp van de schakeling staat in figuur 4.13. □



Figuur 4.13. 4-Input OR poort in NOR-NAND.

De hiervoor besproken recepten voor de logische realisatie van schakelingen in NAND's en/of NOR's kunnen in het geval van *fanin beperkingen* vaak niet zonder meer toegepast worden. Er zijn dan eerst enige voorbereidingen op de logische formules nodig. Zie paragraaf 4.5.

In de jaren 1960-1970 is er veel literatuur verschenen op het gebied van ontwerpen met NAND's en/of NOR's, vooral in de Transactions van de IEEE on (Electronic) Computers [Ellis, 1965; Hellerman, 1963; Lee, 1972; McCluskey, 1963]. Een aparte richting hierbij werd aangegeven in de literatuur over de TANT netwerken. Een *TANT netwerk* is een in drie niveaus met NAND poorten opgebouwd netwerk, waarbij uitsluitend niet-genegeerde externe ingangsvariabelen gebruikt worden. Is in een term een genegeerde variabele nodig, dan wordt deze ter plaatse gegenereerd. Hiervoor kan een negator dienen, maar soms ook een NAND waarvan de uitgang voor verschillende doelen gebruikt wordt. Zie als voorbeeld figuur 4.11.b [Gimpel, 1967; Vink, 1978].

4.5 Ontwerpen met fanin beperkingen

Fysische beperkingen (spanningsval over geleidende transistoren) liggen ten grondslag aan het streven het aantal ingangen van poorten zo veel mogelijk te reduceren. Ook heeft een verhoging van het aantal ingangen een verlaging van de maximale schakelsnelheid tot

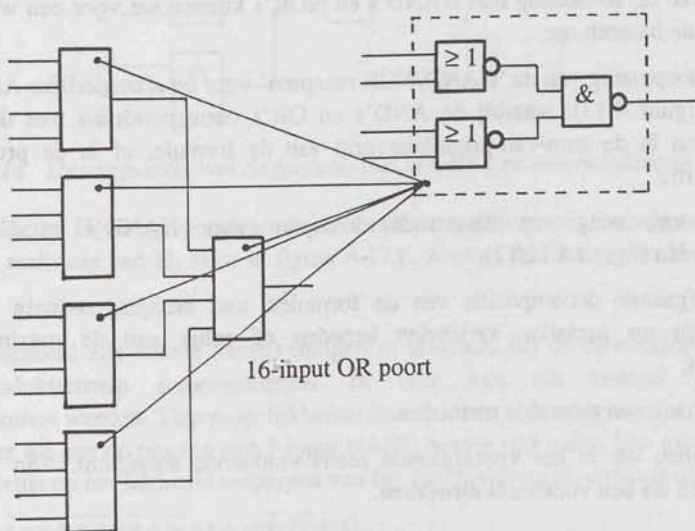
gevolg. In logische formules daarentegen komen vaak produkten en sommen voor met meer dan het toegestane aantal ingangen per poort. Is dit het geval, dan moet de formule passend gemaakt worden op de componenten. Er is dan een conversie nodig van de formule in de som-van-produktenvorm naar een vorm die (beter) afbeeldbaar is.

Voorbeelden van dergelijke formuleconversies hebben we reeds besproken in paragraaf 4.3 (figuren 4.7 en 4.8) en in paragraaf 4.4 (figuur 4.13). Vaak gaat een formuleconversie ook samen met een keuze van andere types poorten, bijvoorbeeld NAND's en/of NOR's. Reden om wat dieper in te gaan op het conversieprobleem.

Opbouw van n-input AND en OR poorten met NAND en NOR poorten

De opbouw van een '4-input OR poort' uit NAND's en NOR's is reeds in paragraaf 4.4 besproken. De formuleconversie die eraan ten grondslag ligt is

$$\begin{aligned}
 S &= w + x + y + z \\
 &= (w + x) + (y + z) \\
 &= \overline{\overline{(w + x)} \cdot \overline{(y + z)}} \\
 &= \overline{(w + x) \cdot (y + z)} \\
 &= \text{NAND}(\text{NOR}(w, x), \text{NOR}(y, z))
 \end{aligned}
 \qquad
 \begin{aligned}
 S &= wxyz \\
 &= (wx) \cdot (yz) \\
 &= \overline{\overline{(wx)} \cdot \overline{(yz)}} \\
 &= \overline{wx + yz} \\
 &= \text{NOR}(\text{NAND}(w, x), \text{NAND}(y, z))
 \end{aligned}
 \tag{4.14}$$

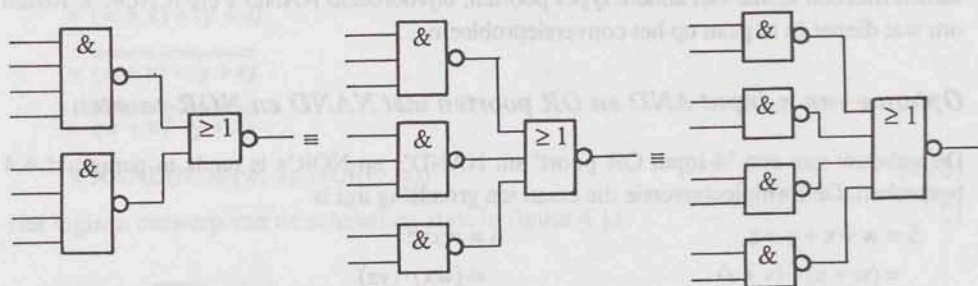


Figuur 4.14. Modulair samengestelde 16-input OR poort.

Het concept is gemakkelijk uitbreidbaar tot n-input OR poorten en AND poorten. Zo kan een 16-input OR poort samengesteld worden uit vijf keer de module van figuur 4.13. Ga de opbouw van de 16-input OR in figuur 4.14 na. Op overeenkomstige wijze kunnen n-input AND poorten worden samengesteld uit 2-input NAND/NOR combinaties.

Wanneer de fanin beperkt is tot 3 à 4, dan zijn ook andere combinaties mogelijk. Figuur 4.15 beschrijft uitvoeringen van een 8-input AND poort uit NAND en NOR poorten met maximale fanin ≤ 4 .

Uit figuur 4.15 zien we dat er meer oplossingen mogelijk zijn naarmate er meer ingangen per NAND of NOR poort zijn toegestaan. Ook zijn er meer oplossingen als het aantal lagen van achter elkaar geschakelde poorten in het definitieve ontwerp groter dan twee mag zijn.



Figuur 4.15. Opbouwmogelijkheden 8-input AND poort.

Multilevel decompositie met NAND en NOR poorten

De realisatie van de schakeling met NAND's en NOR's kunnen we voor een willekeurige logische formule baseren op:

- herhaalde toepassing van de 'NAND/NOR recepten' voor de afzonderlijke AND en OR poorten (figuur 4.13), waarbij de AND's en OR's corresponderen met de logische bewerkingen in de som-van-produktenvorm van de formule, of in de produkt-van-sommenvorm;
- herhaalde toepassing van de standaardrecepten voor NAND's en NOR's, op formuleniveau (figuur 4.11/12);
- een voorafgaande decompositie van de formules, met haakjes, zodanig dat alleen bewerkingen op aantallen variabelen beneden of gelijk aan de maximale fanin voorkomen;
- een combinatie van deze drie methoden.

De eerste hebben we in het voorafgaande reeds voldoende toegelicht. Van de andere methodes zullen we een voorbeeld uitwerken.

Voorbeeld Herhaalde toepassing van het standaardrecept voor NAND's

Realiseer de functie

$$S = w\bar{x}yz + w\bar{y}z + \bar{w}\bar{x}yz + \bar{w}xz + wx\bar{z} \quad (4.15)$$

in NAND's met hoogstens vier ingangen. Voor toepassing van het standaardrecept bevat de *niet-redundante* formule (4.15) één produktterm te veel. Wanneer we twee termen

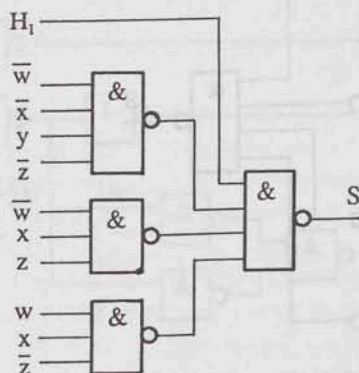
tussen haakjes zetten, dan bevat de formule vier 'termen' en is het standaardrecept toepasbaar.

$$\begin{aligned}
 S &= (\overline{w}\overline{x}yz + \overline{w}x\overline{y}z + \overline{w}xz + w\overline{x}\overline{z}) \\
 &= \overline{(\overline{w}\overline{x}yz + \overline{w}x\overline{y}z + \overline{w}xz + w\overline{x}\overline{z})} \\
 &= \overline{(\overline{w}\overline{x}yz + \overline{w}x\overline{y}z) \cdot \overline{w}xz \cdot w\overline{x}\overline{z}} \\
 &= \text{NAND}(H_1, \text{NAND}(\overline{w}, x, y, z), \text{NAND}(\overline{w}, x, z), \text{NAND}(w, x, \overline{z}))
 \end{aligned} \tag{4.16}$$

waarbij de hulpfunctie H_1 ,

$$H_1 = \overline{(\overline{w}\overline{x}yz + \overline{w}x\overline{y}z)} \tag{4.16.a}$$

apart gerealiseerd moet worden. Zie figuur 4.16.



Figuur 4.16. Decompositie van de formule met NAND's en een hulpfunctie.

De hulpfunctie H_1 is gespecificeerd in het Karnaughdiagram van figuur 4.17.a. Een mogelijke realisatie van H_1 staat in figuur 4.17.b. Andere realisaties van H_1 bespreken we nog. $\square$

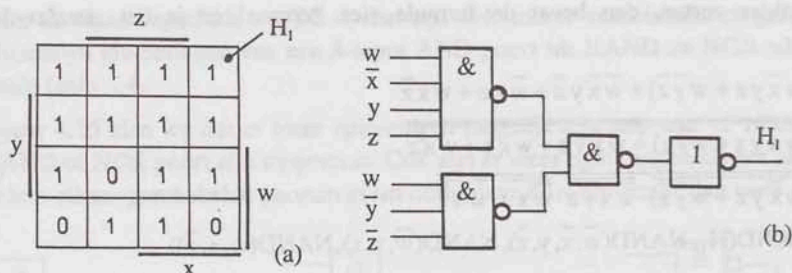
Bij dit voorbeeld zijn enkele kanttekeningen te plaatsen. Bij de uitwerking zijn de eerste twee produkttermen samengenomen. In feite kan elk tweetal produkttermen samengenomen worden. Tien mogelijkheden dus voor een oplossing van het fanin probleem en wanneer we aan de uitgang een 3-input NAND nemen nog meer. Een aardige oplossing, die niet berust op het herhaald toepassen van het standaardrecept, ontstaat uit

$$\begin{aligned}
 S &= \overline{w}\overline{x}yz + \overline{w}x\overline{y}z + \overline{w}xz + w\overline{x}\overline{z} \cdot (\overline{y} + x) \\
 &= \overline{w}\overline{x}yz + \overline{w}x\overline{y}z + \overline{w}xz + w\overline{x}\overline{z} \cdot H_2
 \end{aligned} \tag{4.17}$$

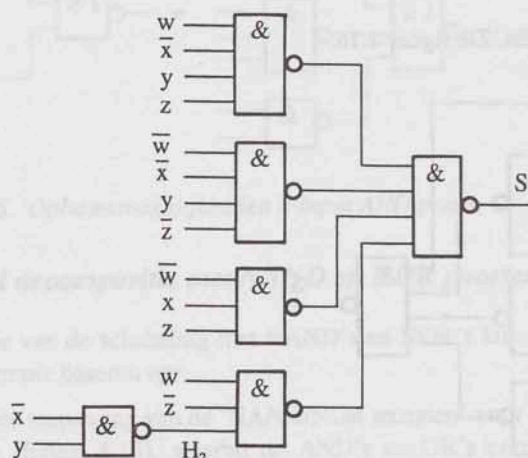
met hulpfunctie H_2 ,

$$H_2 = \overline{y} + x$$

De schakeling staat in figuur 4.18.



Figuur 4.17. Realisatie van hulpfunctie H_1 .



Figuur 4.18. Decompositie van de formule met NAND's en hulpfunctie H_2 .

De derde afbeeldingsmethode, decompositie van de formule met haakjes totdat een afbeeldbare formule wordt verkregen, lichten we toe in het volgende voorbeeld. Dit voorbeeld is eerder besproken voor een multilevel AND/OR realisatie (figuur 4.8). We werken dit voorbeeld nu uit naar 2-input NAND's.

Voorbeeld Multilevel decompositie naar 2-input NAND's

Bij de afbeelding van formule (4.7)

$$S = \bar{x}yz + x\bar{y}z + xy\bar{z}$$

naar 2-input NAND's mogen geen sommen en/of produkten in de formule voorkomen in meer dan twee variabelen. We schrijven formule (4.7) dan bijvoorbeeld in de vorm

$$\begin{aligned}
 S &= \bar{x}yz + x\bar{y}z + xy\bar{z} \\
 &= \bar{x} \cdot yz + x \cdot (\bar{y}z + y\bar{z}) \\
 &= \bar{x} \cdot yz + x \cdot (\bar{y}z \cdot z + \bar{y}z \cdot y) \\
 &= \bar{x} \cdot H_1 + x \cdot H_2
 \end{aligned} \tag{4.18}$$

met

$$H_1 = yz \quad (4.18.a)$$

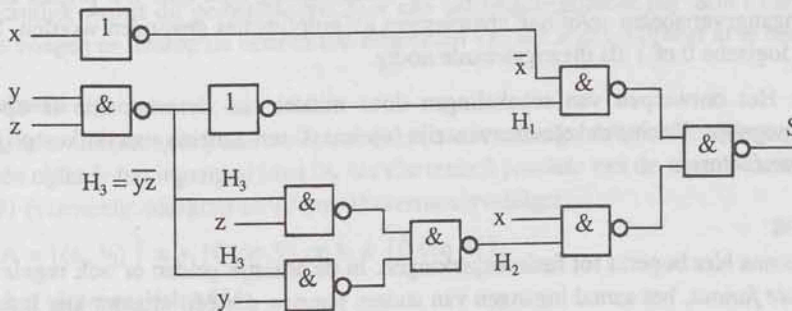
en

$$\begin{aligned} H_2 &= \overline{yz} \cdot z + \overline{yz} \cdot y \\ &= H_3 \cdot z + H_3 \cdot y \end{aligned} \quad (4.18.b)$$

waarbij

$$H_3 = \overline{H_1} \text{ of } H_1 = \overline{H_3} \quad (4.18.c)$$

Formule (4.18) is direct afbeeldbaar op 2-input NAND's en de hulpfuncties H_1 en H_2 ook. Het resultaat staat in figuur 4.19.



Figuur 4.19. Multilevel decompositie met NAND's. □

Het decompositieprobleem

Zij gegeven een verzameling poorten

{NOT, AND, OR, NAND, NOR, EXOR, ...}

elk met een maximale fanin. Het probleem, een functie S te realiseren in deze poorten, komt dan neer op het kiezen van een zodanige verzameling poorten uit de gegeven verzameling dat de functie S erop afbeeldbaar is en waarbij zo goed mogelijk wordt voldaan aan van tevoren vastgestelde optimalisatiefactoren in termen van:

- aantallen componenten,
- aantal poortniveaus in de schakeling,
- propagatietijd,
- dissipatie,
- kostprijs van de componenten,
- testbaarheid.

Het aantal mogelijke oplossingen is als regel zeer groot. Het is niet eenvoudig om hieruit de 'beste' te kiezen.

Toelichting

Op het uitgangsniveau van een te realiseren schakeling zit een poort met de uitgangsfunctie S . Nemen we hiervoor een 3-input NAND, dan moet gelden

$$S = \text{NAND}(H_1, H_2, H_3)$$

De functie S moet daartoe in (maximaal) een drietal hulpfuncties H_1 t/m H_3 'gesplitst' worden. In het voorafgaande hebben we aangetoond dat dit vaak op meer dan één manier mogelijk is. Vooraf is meestal niet te zeggen welke keuze tot de 'eenvoudigste' oplossing leidt. Vervolgens moeten de drie hulpfuncties H_1 t/m H_3 gerealiseerd worden. Voor elke hulpfunctie moet weer een component uit de gegeven verzameling als uitgangspoort gekozen worden, waarvoor weer nieuwe hulpfuncties nodig zijn, enz. Dit proces eindigt wanneer ingangsvariabelen en/of hun complement als hulpfuncties gevonden worden. Soms is ook een logische 0 of 1 als ingangswaarde nodig.

Conclusie: Het ontwerpen van schakelingen door middel van decompositie is op vele manieren mogelijk. Slechts enkele daarvan zijn 'optimaal' ten aanzien van de kostprijs- en optimalisatiefactoren. $\square$

Opmerking

We hebben ons hier beperkt tot fanin beperkingen. In de praktijk gelden er ook regels voor de *maximale fanout*, het aantal ingangen van andere poorten dat één uitgang kan instellen. Deze problemen zijn eenvoudig op te lossen door de uitgangspoort van een schakeling te dupliceren. $\square$

Voor literatuur over het ontwerpen onder fanin beperkingen zie [Davidson, 1969; Dietmeyer, 1969; Vink, 1978]. Decompositiemethoden zijn geïntroduceerd door [Kjelkerud, 1972]. Zijn methode vergde, gezien de toenmalige computers, te veel rekentijd. Vrij recent is decompositie weer in de belangstelling gekomen door [Brayton, 1982; Brayton, 1984].

4.6 Ontwerpvrijheid

In het voorafgaande hoofdstuk hebben we het begrip 'don't care' geïntroduceerd. Het is noodzakelijk dit begrip meer inhoud te geven, in relatie tot het *ontwerptraject*. Op het *architectuurniveau* is het van belang zo veel mogelijk ontwerpvrijheid te herkennen, maar deze vrijheid vooral nog niet in te vullen. Bij de *implementatie* wordt gebruik gemaakt van de gesignaleerde ontwerpvrijheid. Bij het uitwerken van een ontwerp wordt (een gedeelte van de) ontwerpvrijheid ingevuld, maar er kan ook nieuwe ontwerpvrijheid ontstaan. In de *realisatie* is gewoonlijk alle ontwerpvrijheid ingevuld, met name aan de ingangskant. Voor elke ingangscombinatie, of deze nu kan optreden of niet, zal de schakeling een uitgangswaarde produceren.

Voorbeeld

Een binaire vermenigvuldiger hoeft intern niet noodzakelijk met het binaire talstelsel te werken. De vermenigvuldiger moet alleen binair gecodeerde operanden kunnen accepteren en moet een binair gecodeerd resultaat afgeven. $\square$

In de *automatentheorie* (architectuurniveau!) wordt een *combinatorische schakeling* gewoonlijk beschreven door middel van een afbeelding van een ingangsalphabet (IA) op/in een uitgangsalphabet (UA). Het *ingangsalphabet* is dan de verzameling 'stimuli' aan de ingang(en) van de te ontwerpen schakeling en het *uitgangsalphabet* is de verzameling 'responsies' van de schakeling hierop. Een schakeling heet *combinatorisch* als bij elk element van het ingangsalphabet (maximaal) één element van het uitgangsalphabet behoort.

Wanneer aan sommige elementen van het ingangsalphabet geen element van het uitgangsalphabet is toegewezen spreken we van een don't care op het architectuurniveau. Gewoonlijk wordt dit gemodelleerd door aan het uitgangsalphabet het '*don't care element*' toe te voegen en daarop de betreffende elementen van het ingangsalphabet af te beelden.

Voorbeeld

Bij de *architectuurspecificatie* van een *decimale vermenigvuldiger* voor twee getallen van elk één cijfer is het ingangsalphabet IA het *Cartesisch produkt* van de twee verzamelingen $\{0 \text{ t/m } 9\}$ (vermenigvuldigital) en $\{0 \text{ t/m } 9\}$ (vermenigvuldiger),

$$IA = \{(a_i, b_j) \mid a_i \in \{0 \text{ t/m } 9\} \text{ en } b_j \in \{0 \text{ t/m } 9\}\} \quad (4.19)$$

en is het uitgangsalphabet UA

$$UA = \{(a_i b_j) \mid a_i \in \{0 \text{ t/m } 9\} \text{ en } b_j \in \{0 \text{ t/m } 9\}\} \quad (4.20)$$

Op het architectuurniveau zijn de ingangs- en uitgangsalphabetten van de vermenigvuldiger compleet gespecificeerd! (De verzameling UA is een deelverzameling van $\{0 \text{ t/m } 99\}$; dit kan van belang zijn voor de implementatie van te ontwerpen schakelingen.)

De *implementatie* van de vermenigvuldiger omvat onder andere de afbeelding naar het binaire niveau. Daarbij worden de operanden in de BCD of BIN code (hoofdstuk 1) voorgesteld en het resultaat in de BCD of BIN code. We nemen aan dat de operanden en het resultaat in de BIN code moeten staan.

Er treden nu opeens een heleboel niet-gespecificeerde situaties op. Van de 8-bit ingangscombinaties komen er 100 van de 256 voor. De andere 156 treden niet op. Verder komen er van de 7-bit uitgangscombinaties 37 van de 128 voor. *Op het implementatieniveau is de specificatie verre van compleet.*

Op het *realisatieniveau* kunnen we bijvoorbeeld beslissen een binaire vermenigvuldiger toe te passen. Nu mogen alle 256 combinaties aan de ingang voorkomen (hoewel ze in deze specifieke toepassing niet alle 256 gebruikt worden) en van de 256 mogelijke 8-bit combinaties komen er aan de uitgang 90 voor. De overige worden bij de gegeven binaire operanden niet gegenereerd. $\square$

Uit het voorafgaande mogen we concluderen:

- Elke architectuurspecificatie bevat ontwerpvrijheid. Het is belangrijk deze ontwerpvrijheid expliciet in kaart te brengen.
- Tijdens de implementatiefase wordt een deel van de ontwerpvrijheid *gemotiveerd* ingevuld.
Tevens ontstaat er in de implementatiefase vaak extra ontwerpvrijheid, onder meer wanneer de ingangs- en uitgangsalphabetten worden afgebeeld op bit strings.
- Bij de realisatie wordt alle ontwerpvrijheid aan deingangskant ingevuld. Voor elke ingangscombinatie genereert de schakeling een uitgangscombinatie.
- Ten aanzien van de verzameling uitgangscombinaties geldt dat vaak niet alle mogelijke k-bit combinaties gegenereerd worden. Dit levert don't care ingangswaarden op voor de achterliggende schakelingen.
- Vaak kan een gerealiseerde schakeling meer dan nodig is. We zeggen dan dat de architectuur afgebeeld is *in* de realisatie.

Traditioneel wordt *ontwerpvrijheid* benut om een schakeling te '*minimaliseren*', waarbij men dan meestal het aantal componenten op het oog heeft. Binnen een IC is de prijs van extra poorten zeer laag. Te veel nadruk op minimaliseren is dan fout. Beter is het om ontwerpvrijheid in te zetten voor aspecten als foutafhandeling, minimalisatie van de dissipatie, mogelijkheden om het produkt meer universeel toepasbaar te maken, enz. In de hoofdstukken 6 en 7 zullen we op deze aspecten van het invullen van ontwerpvrijheid terugkomen.

4.7 Permissible functions

Don't cares komen ook voort uit de *topologie van de gerealiseerde schakeling*. In de literatuur wordt dit aspect van ontwerpvrijheid besproken onder *permissible functions* [Muroga, 1989]. Onderstaande voorbeelden maken het begrip permissible function duidelijk. Bij dit onderwerp beperken we ons uitdrukkelijk tot de *statische toestand* van de schakeling, het gedrag als de schakeling geheel in rust is. Alle signalen hebben dan een logisch gedefinieerd signaalniveau. De schakelalgebra mag toegepast worden. Soms zitten er in de schakeling ook (redundante) poorten om een bepaald dynamisch gedrag te realiseren. Deze laten we hier uitdrukkelijk buiten beschouwing.

Twee combinatorische schakelingen heten (logisch) *uitwisselbaar* indien zij elkaar in een bepaalde logische omgeving kunnen vervangen. De uitwisselbaarheid geldt uitsluitend voor de onderzochte aspecten. Combinatorische schakelingen kunnen logisch uitwisselbaar zijn, terwijl ze op andere gebieden verschillende eigenschappen bezitten (dissipatie, propagatietijd, oppervlak, enz.). In dit hoofdstuk onderzoeken we logische uitwisselbaarheid. Hoofdstuk 5 spreekt een aantal aspecten van het elektrisch compatibel zijn.

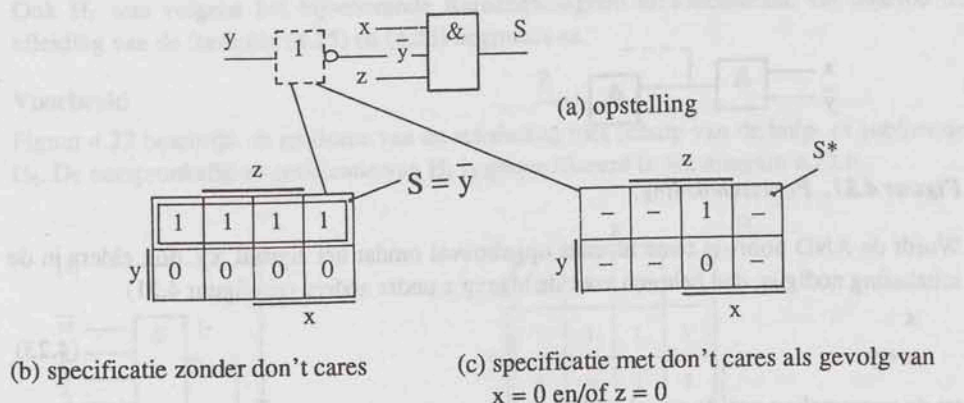
Voorbeeld

Staat op een van de ingangen van een AND poort een 0, dan is in de *statische toestand* het signaal op de andere ingang don't care. Immers, $0 \cdot \bar{z} = 0 \cdot z = 0$. Hetzelfde geldt voor een

NAND poort met een ingangswaarde 0 op een of meer van de ingangen en voor een OR/NOR poort voor ingangswaarde 1. $\square$

Voorbeeld

Figuur 4.20.a beschrijft een poort, die uit een gerealiseerde schakeling is gelicht. De poort realiseert de logische functie $S = x\bar{y}z$. In de rest van de schakeling is het signaal $\bar{y}$ niet beschikbaar. De gangbare manier om $\bar{y}$ op te wekken is de toepassing van een negator, waarmee $\bar{y}$ van het signaal y wordt afgeleid. Gezien het vorige voorbeeld is het signaal uit de negator dan een don't care als $x = 0$ en/of als $z = 0$. Deze ontwerpvrijheid is in figuur 4.20.b/c weergegeven.



Figuur 4.20. Permissible functions voor $\bar{y}$.

Indien op een ander punt in de schakeling een signaal dat voldoet aan de specificatie volgens figuur 4.20.c beschikbaar is, bijvoorbeeld $S^* = \bar{y} + x$, dan mag dit signaal gebruikt worden om de ingang $\bar{y}$ in te stellen. Immers:

$$\begin{aligned}
 S &= x \cdot (\bar{x} + \bar{y}) \cdot z \\
 &= x\bar{x}z + x\bar{y}z \\
 &= 0 + x\bar{y}z \\
 &= x\bar{y}z
 \end{aligned}
 \tag{4.21}$$

Aan de ingang met variabele $\bar{y}$ mag een hele verzameling van functies worden aangeboden, waaronder

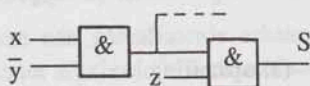
$$\begin{aligned}
 &\bar{y} \\
 &\bar{y} + \bar{x} \\
 &\bar{y} + \bar{z} \\
 &\bar{y} + \bar{x} + \bar{z}
 \end{aligned}
 \tag{4.22}$$

Deze functies behoren tot de 'verzameling van de permissible functions' voor ingang $\bar{y}$. Om precies te zijn, de verzameling van permissible functions voor de ingang met $\bar{y}$ erop

bevat $2^6 = 64$ verschillende functies. Een ervan is voldoende om de betreffende ingang in te stellen. Is zo'n signaal elders beschikbaar, dan is de negator om $\bar{y}$ uit y af te leiden overbodig. $\square$

Voor elke ingang van een poort kan zo'n *verzameling van permissible functions* worden opgesteld. Daaruit kan een keuze gemaakt worden voor het signaal op die ingang. Deze keuze is echter **niet** onafhankelijk van de keuze voor de andere ingangen. De verzameling van permissible functions is slechts op te stellen bij *gegeven signalen* op de andere ingangen en tevens moeten het *dominante ingangssignalen* zijn. Waarom de laatste toevoeging?

Voorbeeld



Figuur 4.21. Poortschakeling.

Wordt de AND poort in twee niveaus opgebouwd omdat het signaal $x\bar{y}$ ook elders in de schakeling nodig is, dan behoren voor de ingang x onder andere (zie figuur 4.21)

$$\begin{array}{l} x \\ x + y \end{array} \quad (4.23)$$

tot de verzameling van de permissible functions en voor de ingang $\bar{y}$ onder andere

$$\begin{array}{l} \bar{y} \\ \bar{y} + \bar{x} \end{array} \quad (4.24)$$

Of ook $x + \bar{z}$ en $x + y + \bar{z}$ resp. $\bar{y} + \bar{z}$ en $\bar{y} + \bar{x} + \bar{z}$ en alle ervan afgeleide functies tot de twee verzamelingen van permissible functions behoren kan slechts vastgesteld worden wanneer ook de andere door het signaal $x\bar{y}$ ingestelde punten onderzocht zijn. $\square$

Conclusies

- Op *ingangen van poorten* kunnen de ingangssignalen vervangen worden door andere signalen of logische functies, waarbij de werking van de poort in zijn omgeving niet verandert (statische beschouwingwijze). De ingang van een AND poort heeft voor zijn ingangssignaal/ingangsfunctie don't cares op die plaatsen, waarvoor een of meer van de andere ingangssignalen 0 zijn. Voor een OR poort is dit de ingangswaarde 1.
- De verzameling van permissible functions voor een poortingang is slechts gedefinieerd bij gegeven signalen of functies op de andere ingangen. Verzamelingen van permissible functions zijn niet onafhankelijk van elkaar te bepalen.
- Hetzelfde geldt voor poorten of deelschakelingen in grotere schakelingen, zoals we hierna zullen aantonen. $\square$

Veel *ontwerpvrijheid* kan via *permissible functions* opgespoord worden. Het is ontwerpvrijheid die afhangt van de *topologie van de gegeven schakeling*. Dit geldt niet alleen voor de ingangskant van een poort of deelschakeling. Als voorbeeld nemen we nogmaals de schakeling van figuur 4.16, waarin van de niet-redundante logische functie S,

$$S = (w\bar{x}yz + w\bar{y}z) + \bar{w}\bar{x}y\bar{z} + \bar{w}xz + wx\bar{z} \quad (4.25)$$

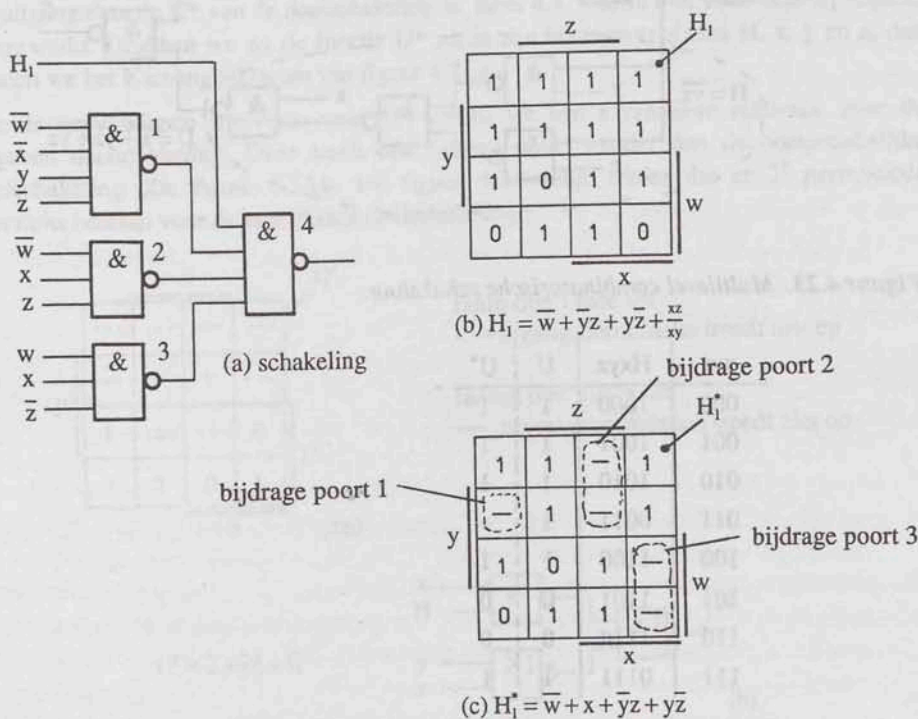
door middel van decompositie een realisatie is afgeleid in 4-input NAND poorten met behulp van de hulpfunctie H_1

$$H_1 = \overline{w\bar{x}yz + w\bar{y}z} \quad (4.26)$$

Ook H_1 was volgens het bijbehorende Karnaughdiagram niet-redundant. Ga daartoe de afleiding van de formules (4.25) en (4.26) nogmaals na.

Voorbeeld

Figuur 4.22 beschrijft de realisatie van de schakeling met behulp van de hulp- of subfunctie H_1 . De oorspronkelijke specificatie van H_1 is gespecificeerd in het diagram 4.22.b.



Figuur 4.22. Permissible functions.

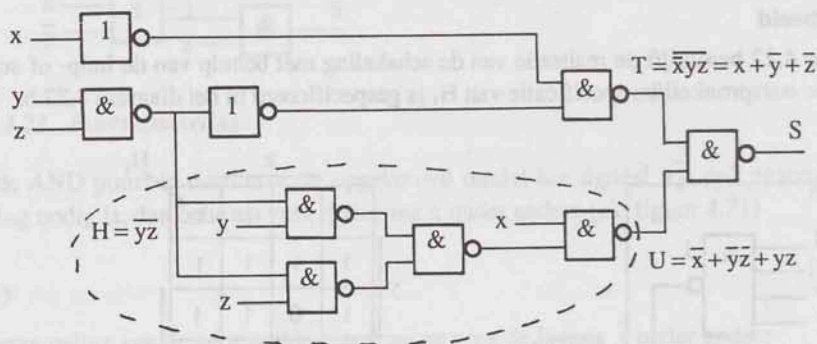
Toch bevat H_1 in de gegeven schakeling enige ontwerpvrijheid. Als een van de andere ingangen van poort 4 in figuur 4.22.a de waarde '0' heeft, dan is de functie H_1 een don't care. Verwerken we deze ontwerpvrijheid in de specificatie voor H_1 dan ontstaat figuur

4.22.c. Het blijkt dat er $2^5 = 32$ verschillende logische functies mogelijk zijn, die H_1 kunnen vervangen. Het hangt nu van de voor H_1 gekozen componenten af of we deze ontwerpvrijheid nuttig kunnen toepassen. $\square$

In paragraaf 4.13 gaan we na hoe en welke rol permissible functions spelen bij het testen van combinatorische schakelingen.

Het optimaliseren van combinatorische schakelingen

In figuur 4.23 staat een reeds eerder ontworpen schakeling (figuur 4.19). In figuur 4.23 hebben we een deelschakeling hiervan omcirkeld. Voor deze deelschakeling stellen we de verzameling van permissible functions op.



Figuur 4.23. Multilevel combinatorische schakeling.

xyz	Hxyz	U	U*
000	1000	1	1
001	1001	1	1
010	1010	1	1
011	0011	1	—
100	1100	1	1
101	1101	0	0
110	1110	0	0
111	0111	1	1

Tabel 4.4. Primaire en secundaire ingangscombinaties

De deelschakeling heeft drie *primaire* of *externe ingangen*, waarop de ingangsvariabelen x , y en z zijn aangesloten. Daarnaast is er één *secundaire* of *interne ingang*, waarop de functie H is aangesloten. Hiervoor geldt:

$$H = \overline{yz} = \overline{y} + \overline{z} \quad (4.27)$$

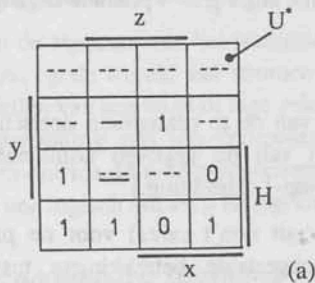
Stel dat x , y en z alle acht ingangscombinaties kunnen doorlopen. Dan volgen hieruit de ingangscombinaties voor de omcirkelde deelschakeling. Tabel 4.4 geeft een overzicht. We zien uit tabel 4.4 dat aan de ingangen van de deelschakeling slechts acht van de 16 mogelijke ingangscombinaties in $Hxyz$ optreden. De overige ingangscombinaties behoren tot de *fanin don't care set* van de geselecteerde deelschakeling. De deelschakeling zet de acht aangeboden combinaties om in de *secundaire* of *interne uitgangswaarde* U . Van het signaal U komen beide waarden voor.

De uitgangswaarde U van de deelschakeling is niet altijd nodig, met name niet als er een 0 op de andere ingang van de uitgangspoort voor S staat. Dit is het geval als de hulpfunctie T ,

$$T = x + \overline{y} + \overline{z} \quad (4.28)$$

de uitgangswaarde 0 heeft. Dit is zo voor $xyz = 011$. De ingangscombinatie $xyz = 011$ behoort tot de *fanout don't care set* van de deelschakeling, de verzameling van ingangscombinaties waarvoor de uitgang van de (deel)schakeling don't care is. Dit leidt tot de uitgangsfunctie U^* van de deelschakeling in tabel 4.4, waarin ook deze ontwerpvrijheid is verwerkt. Drukken we nu de functie U^* uit in zijn ingangsvariabelen H , x , y en z , dan vinden we het Karnaughdiagram van figuur 4.24.a.

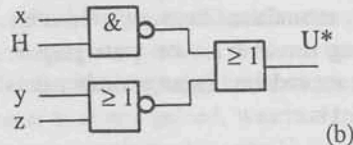
Via de eenvoudigste somvorm voor U^* vinden we een alternatieve realisatie voor de gegeven deelschakeling. Deze heeft één 2-input poort minder dan de oorspronkelijke deelschakeling. Zie figuur 4.24.b. Uit figuur 4.24 volgt verder dat er 2^9 *permissible functions* bestaan voor de omcirkelde deelschakeling.



$$U^* = \overline{x} + \overline{y} + \overline{H}$$

fanin don't care set
 --- ingangscombinatie treedt niet op

fanout don't care set
 — uitgangscmbinatie treedt niet op



Figuur 4.24. Alternatieven voor de geselecteerde deelschakeling met een realisatie.

Opmerking

Bepalen we voor twee of meer deelschakelingen de verzamelingen van permissible functions, dan mogen deze deelschakelingen *niet tegelijk* gereduceerd worden en door hun

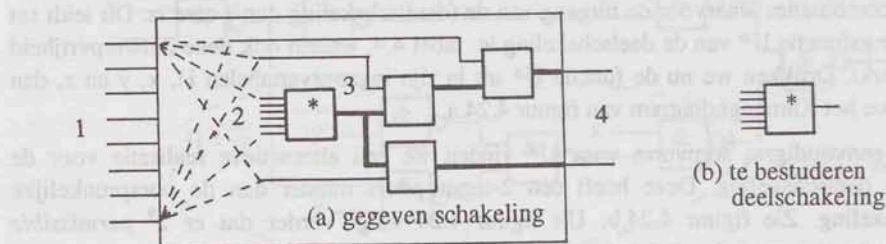
gereduceerde versies vervangen worden. De bij de ene deelschakeling gemaakte keuzes kunnen van invloed zijn op de verzameling van de permissible functions voor de andere deelschakeling.

Ontwerp bijvoorbeeld een negator op basis van een 2-input NOR en wel door op beide ingangen dezelfde variabele aan te sluiten. Stel vervolgens de verzamelingen van de permissible functions op voor beide ingangssignalen. Optimaliseer beide ingangssignalen onafhankelijk van elkaar. Wat zien we? □

In moderne ontwerpsoftware worden permissible functions vaak toegepast. Een initieel ontwerp wordt in een aantal rondes geoptimaliseerd door reductie van deelschakelingen.

Procedure voor het bepalen van de permissible functions

De procedure voor het bepalen van alle permissible functions voor een gegeven deelschakeling is als volgt. (Zie ook figuur 4.25 en paragraaf 4.12.)



1 primaire ingangen 2 secundaire ingangen 3 secundaire uitgangen 4 primaire uitgangen

Figuur 4.25. Het zoeken van permissible functions.

- Druk de *secundaire (interne) ingangsvariabelen* van de te bestuderen deelschakeling uit in de *primaire (externe) ingangsvariabelen* van de gegeven combinatorische schakeling. (Voor sommige variabelen is het een 1-op-1 afbeelding.)
- Bepaal de *don't care set* (DC set van input-output don't cares) voor de *primaire ingangscombinaties*. Beeld deze, middels de opgestelde betrekkingen tussen de primaire en secundaire ingangsvariabelen, af op de *secundaire ingangscombinaties*. Voor deze secundaire ingangscombinaties is de functie van de te bestuderen deelschakeling een *don't care*, mits geen andere primaire ingangscombinaties op de betreffende secundaire ingangscombinatie(s) afbeelden waarvoor de functie wel gespecificeerd is.
- Voor alle *niet-gegenereerde secundaire ingangscombinaties* is de uitgangswaarde van de te bestuderen deelschakeling *don't care*.
- Zoek het *pad* van de (*secundaire*) *uitgang* van de te bestuderen deelschakeling naar de (*primaire*) *uitgang* van de schakeling. Ga na of dit pad op enigerlei wijze *geblokkeerd* kan worden. Bepaal de primaire ingangscombinaties die hiertoe aanleiding geven.

- Beeld deze ingangscombinaties af op de secundaire ingangscombinaties van de deelschakeling. Deze secundaire ingangscombinaties behoren tot de fanout don't care set van de deelschakeling.
- Soms kan er meer dan één pad tegelijk geactiveerd worden van de secundaire naar de primaire uitgangen. Dan behoort een primaire ingangscombinatie dan en slechts dan tot de primaire ingangscombinaties die afbeelden op de fanout don't care set van de te bestuderen deelschakeling als de betreffende primaire ingangscombinatie *alle* paden naar de primaire uitgangen blokkeert.

Permissible functions zijn toepasbaar voor het optimaliseren van combinatorische logica. Ook bij het testen van combinatorische schakelingen spelen permissible functions een belangrijke rol.

4.8 De afbeelding op hardware

Het implementatietraject start met het omzetten van een gewenst gedrag (de architectuur) in een logische specificatie van het ontwerp. Op dit niveau zijn onze uitdrukkingsmiddelen onder andere proposities en logische bewerkingen. Tijdens het implementeren worden een aantal details uitgewerkt, zowel op functioneel niveau (hoe detecteer je capaciteitsoverschrijding bij een opteller bijvoorbeeld) als op realisatieniveau (aanpassingen aan reeds bekende fanin beperkingen). Uiteindelijk ontstaat er een *gedetailleerd logisch ontwerp* dat op hardware afgebeeld kan worden. Uit het voorafgaande is één ding heel duidelijk geworden. Voordat een ontwerp afbeeldbaar is moet er op logisch niveau het nodige voorwerk gedaan worden.

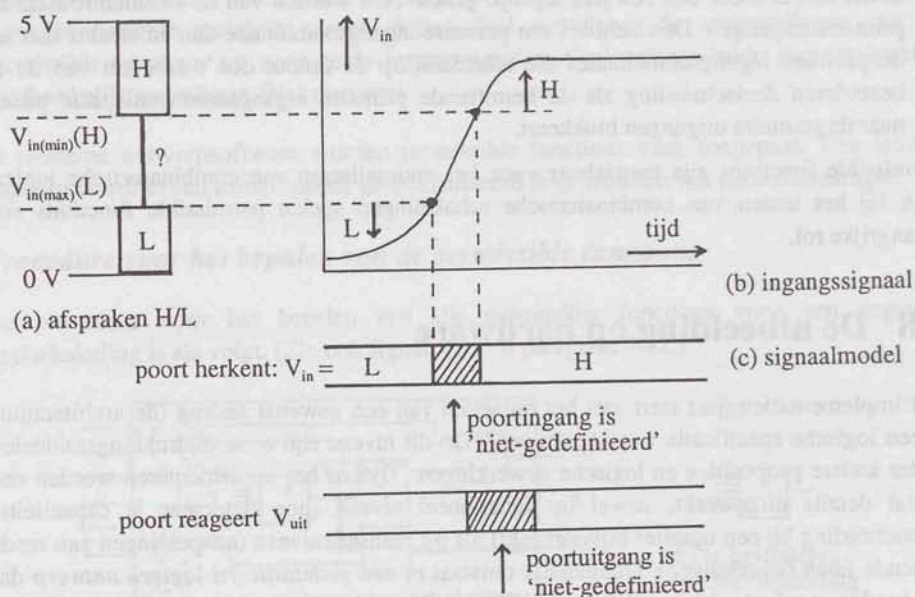
Een van de aspecten van het *realisatieniveau* is de afbeelding van het logische ontwerp op hardware, op de wereld van stromen en spanningen. Informatie-overdracht geschiedt door het instellen van een hoge of lage geleidbaarheid van transistoren, waardoor bijvoorbeeld de uitgangsspanning van een component naar laag (L) of hoog (H) gestuurd wordt. We beperken ons tot dit type componenten (ook wel aangeduid als *restoring logic*) en gaan na hoe we een logisch ontwerp hierop kunnen afbeelden.

Het signaalmodel: scheiding van logica en tijd

In figuur 4.26.a zijn de afspraken aangegeven, volgens welke alle componenten in een bepaalde logische familie hun ingangsspanningen interpreteren. Er is een met H aangeduid gebied, waarin de aangeboden ingangsspanning als H (hoog) herkend wordt. Dit zijn alle ingangsspanningen boven $V_{in(min)}(H)$. Evenzo is er een gebied, waarin de ingangsspanning als L (laag) herkend wordt. Dit zijn alle ingangsspanningen beneden $V_{in(max)}(L)$.

Tussen $V_{in(max)}(L)$ en $V_{in(min)}(H)$ is de ingangsspanning *logisch* niet gedefinieerd. De logische werking van de onderhavige component is diensgevolge ook 'onbepaald'. Gedurende de reactietijd van de poort is het uitgangssignaal van de poort daardoor eveneens onbepaald, ofwel '*niet-gedefinieerd*'. Zie figuur 4.26.b/c.

Als gevolg van de minimum/maximum propagatietijd van de poort valt het gebied waarin het uitgangssignaal niet gedefinieerd is wat later in de tijd. Zie ook de afleiding van het tijdsmodel in hoofdstuk 5.



Figuur 4.26. Signaalmodel met niet-gedefinieerd gebied.

Op het realisatieniveau hebben we dus niet alleen te maken met het *logische model* van een component, maar ook met het *signaalmodel* dat aangeeft wanneer en hoe lang de werking tijdens een ingangsverandering niet gedefinieerd is. We stellen de gedetailleerde behandeling van het signaalmodel uit tot hoofdstuk 5 en richten ons hier verder op de afbeelding van waarheidswaarden op spanningsniveaus. We beperken ons voorlopig tot de *statische toestand* van schakelingen.

Positieve logica en negatieve logica

Logische waarden kunnen op verschillende manieren voorgesteld worden door de spanningsniveaus H en L. Bij *positieve logica* spreekt men af om de logische 0 op L af te beelden en de logische 1 op H, dus

$$\begin{aligned} 0 &\leftrightarrow L \\ 1 &\leftrightarrow H \end{aligned} \quad \text{positieve logica afspraak} \quad (4.29)$$

De afspraak om positieve logica te hanteren geldt gewoonlijk voor het gehele ontwerp. Het alternatief is *negatieve logica*,

$$\begin{aligned} 0 &\leftrightarrow H \\ 1 &\leftrightarrow L \end{aligned} \quad \text{negatieve logica afspraak} \quad (4.30)$$

Ook deze afspraak geldt voor het gehele ontwerp. De praktijk heeft uitgewezen dat het door elkaar heen gebruiken van positieve en negatieve logica uiterst verwarrend is en tot veel fouten aanleiding geeft.

Voorbeeld

logische AND poort			y	z	S	
			L	L	L	bij afbeelding in positieve logica
			L	H	L	
			H	L	L	$0 \leftrightarrow L$
			H	H	H	$1 \leftrightarrow H$

y	z	S	
0	0	0	
0	1	0	
1	0	0	
1	1	1	

y	z	S	
H	H	H	bij afbeelding in negatieve logica
H	L	H	
L	H	H	$0 \leftrightarrow H$
L	L	L	$1 \leftrightarrow L$

Tabel 4.5. Afbeelding AND poort op in H/L werkende component.

Uit tabel 4.5 volgt dat de afspraak om positieve of negatieve logica toe te passen gevolgen heeft voor de te ontwerpen schakeling. De H/L tabellen zijn verschillend en de schakelingen dus ook! $\square$

Uit dit voorbeeld volgt dat we duidelijk moeten vastleggen welke 'logische conventie' is toegepast. In de praktijk wordt meestal vanuit positieve logica gewerkt. (Bedenk dat je in het verkeer zowel links als rechts kunt rijden, maar dat door elkaar heen rijden uiterst problematisch is.)

De logische formule bij een gegeven component

In databoeken worden poortschakelingen gewoonlijk met een H/L tabel beschreven. Men beschrijft hiermee de *fysische werking*. Door de ontwerper wordt hieraan een *logische interpretatie* gegeven en wel door een logische conventie te kiezen. Toch staat er in een databoek vaak een logische formule bij een component, meestal met de toevoeging 'in positive logic'. Vanwaar deze toevoeging?

Voorbeeld

Tabel 4.6 geeft aan dat de logische formule bij een component verschilt naar gelang er positieve of negatieve logica is toegepast.

H/L poort				
y	z	S	y	z S
L	L	L	0	0 0
L	H	H	0	1 1
H	L	H	1	0 1
H	H	L	1	1 0

$S = y \oplus z$
 bij interpretatie in
 positieve logica
 $L \leftrightarrow 0$
 $H \leftrightarrow 1$

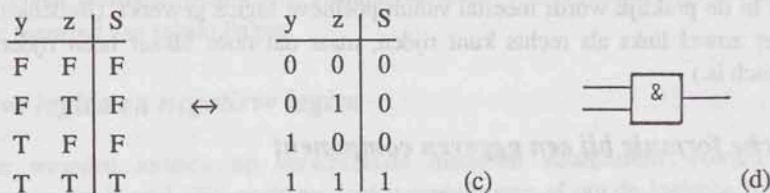
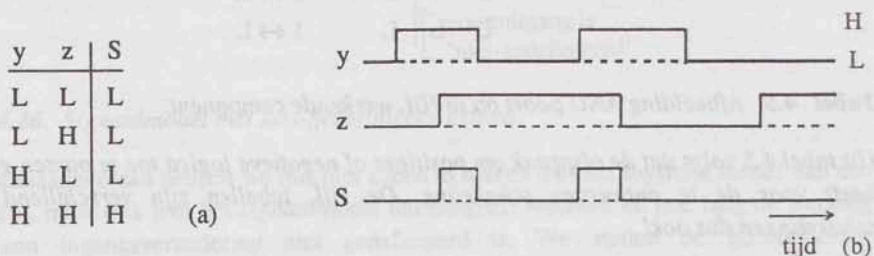
y	z	S		
1	1	1	y	z S
1	0	0	0	0 0
0	1	0	0	1 0
0	0	1	1	1 1

$S = \overline{y \oplus z}$
 bij interpretatie in
 negatieve logica
 $L \leftrightarrow 1$
 $H \leftrightarrow 0$

Tabel 4.6. Logische interpretatie van een fysische poort.

□

Voorbeeld

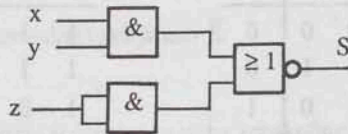


Figuur 4.27. Specificatie door middel van een tijddiagram.

In figuur 4.27.a staat de H/L tabel van een poort. Dezelfde werking wordt ook beschreven in het tijddiagram van figuur 4.27.b. Bij *positieve logica* volgen in figuur 4.27 uit tabel (a) de tabellen (c). Hierin staat de logische bewerking EN gespecificeerd. We mogen de gegeven poort (in positieve logica!) dus voorstellen met het symbool in figuur 4.27.d. De formule van deze poort is $S = y \cdot z$. Het is een AND poort. □

Voorbeeld

Figuur 4.28 beschrijft het logisch ontwerp van een schakeling.



Figuur 4.28. Poortschakeling.

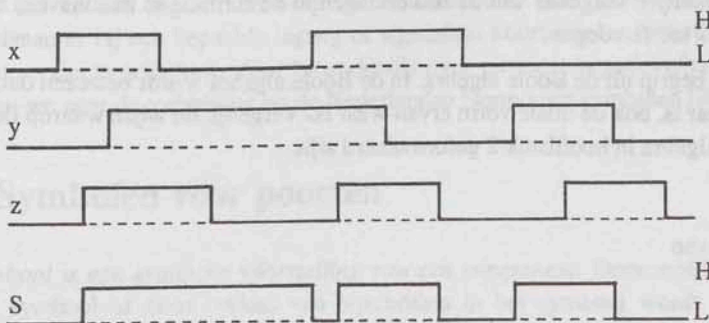
a.	x	y	z	S	b.	x	y	z	S	c.	x	y	z	S	d.	x	y	z	S
	L	L	L	L		L	L	L	L		L	L	L	H		L	L	L	H
	L	L	H	L		L	L	H	H		L	L	H	H		L	L	H	L
	L	H	L	L		L	H	L	L		L	H	L	H		L	H	L	H
	L	H	H	H		L	H	H	H		L	H	H	L		L	H	H	L
	H	L	L	L		H	L	L	L		H	L	L	H		H	L	L	H
	H	L	H	H		H	L	H	H		H	L	H	L		H	L	H	L
	H	H	L	L		H	H	L	H		H	H	L	H		H	H	L	L
	H	H	H	H		H	H	H	H		H	H	H	L		H	H	H	L

Tabel 4.7. H/L tabellen.

Indien negatieve logica wordt toegepast, welke van de vier tabellen in H/L van tabel 4.7 behoort dan bij de beschreven schakeling? Het antwoord (c) vinden we door eerst een logische waarheidstabel op te stellen en deze via negatieve logica af te beelden op H/L. □

Voorbeeld

Gegeven is het tijddiagram van een schakeling met ingangssignalen x, y en z en uitgangssignaal S. Zie figuur 4.29. Welke logische functie behoort in positieve en in negatieve logica bij deze schakeling?



Figuur 4.29. Tijddiagram.

Het antwoord vinden we door de bijbehorende H/L tabel op te stellen en hieruit de 0/1 tabellen voor positieve en negatieve logica af te leiden (zie tabel 4.8).

x	y	z	S	x	y	z	S	x	y	z	S
L	L	L	L	0	0	0	0	1	1	1	1
L	L	H	L	0	0	1	0	1	1	0	1
L	H	L	H	0	1	0	1	1	0	1	0
L	H	H	H	0	1	1	1	1	0	0	0
H	L	L	L	1	0	0	0	0	1	1	1
H	L	H	H	1	0	1	1	0	1	0	0
H	H	L	L	1	1	0	0	0	0	1	1
H	H	H	H	1	1	1	1	0	0	0	0

positieve logica S_1

negatieve logica S_2

Tabel 4.8. Van H/L tabel naar waarheidstabellen.

In positieve logica is de formule S_1

$$S_1 = xz + \bar{x}y \quad (4.31)$$

en in negatieve logica S_2

$$S_2 = xy + \bar{x}z \quad \square \quad (4.32)$$

Dualiteit

Over dit interpretatieprobleem rond positieve en negatieve logica valt wel iets meer te zeggen. We moeten daarvoor eerst het begrip duale vorm van een formule invoeren. De *duale vorm van een formule* ontstaat uit de formule door:

- elke 0 door 1 en elke 1 door 0 te vervangen;
- elke $\cdot$ door $+$ en elke $+$ door $\cdot$ te vervangen;
- de oorspronkelijke volgorde van de bewerkingen in de formule te handhaven, zo nodig door haakjes toe te voegen.

Dualiteit is een begrip uit de Boole algebra. In de Boole algebra wordt bewezen dat, als een vergelijking waar is, ook de duale vorm ervan waar is. Vergelijk de wijze waarop de wetten van de schakelalgebra in hoofdstuk 2 geformuleerd zijn.

Voorbeeld

De duale vorm van

$$\begin{aligned} S &= y \oplus z \\ &= \bar{y}z + y\bar{z} \end{aligned}$$

is

$$\begin{aligned}
 S_d &= (\bar{y} + z)(y + \bar{z}) \\
 &= \bar{y}z + y\bar{z} \\
 &= y \oplus z
 \end{aligned}
 \tag{4.33}$$

Deze vorm is (toevallig) gelijk aan $\bar{S}$. □

Stelling 4.1.

Gegeven een poort (combinatorische schakeling) met de bijbehorende H/L tabel. Dan zijn de formules, die volgens de positieve en de negatieve logica afspraak bij deze component behoren, elkaars duale vorm.

Voor het bewijs zie [Thijssen, 1988]. □

Toelichting

Passen we op formules (4.31) en (4.32) het dualiteitsprincipe toe, dan vinden we

$$\begin{aligned}
 S_{1d} &= (x + z)(\bar{x} + \bar{y}) \\
 &= xy + x\bar{z} + y\bar{z} \\
 &= xy + \bar{x}\bar{z} \\
 &= S_2
 \end{aligned}
 \tag{4.34}$$

Dit is een extra controle op de juistheid van de eerder afgeleide formules (4.31/32). □

Individuele logica

Bij de documentatie van gerealiseerde schakelingen wordt uitgegaan van *individuele logica*. Dan wordt bij elke signaalnaam in een tekening met (H) of (L) achter de naam aangegeven wanneer het signaal 'actief' is. Zo stelt

$$\langle \text{naam} \rangle (H) = \text{reset}(H) \tag{4.35}$$

bij een resetingang dat de resetfunctie in werking is bij een signaalniveau H op deze ingang.

Individuele logica wordt toegepast bij *onderhoudschema's*. Deze schema's zijn niet bedoeld om er een logisch ontwerp uit te distilleren, maar om direct te zien welk signaalniveau er bij een bepaalde ingang of signaallijn hoort om het betreffende signaal of de functie ervan te activeren. Voor meer details over positieve/negatieve/individuele logica verwijzen we naar de toelichting op de Nederlandse Norm voor symbolen [NPR 5160].

4.9 Symbolen voor poorten

Een *symbool* is een grafische voorstelling van een component. Door middel van de vorm van het symbool of door middel van bijschriften in het symbool wordt de betreffende component aangeduid. De oudste in de digitale techniek toegepaste symbolen zijn de *vormsymbolen* van figuur 4.30. Ieder type poort kreeg een eigen symbool [IEEE, 1976].

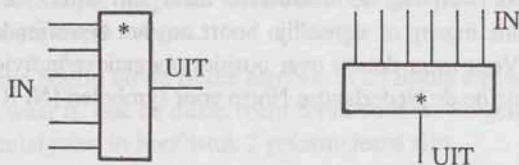
Naam	Symbool	Naam	Symbool
AND		NAND	
OR		NOR	
NOT		EXOR	

Figuur 4.30. IEEE vormsymbolen.

Naarmate er meer en complexere IC's geproduceerd werden ontstond er behoefte aan symbolen met meer mogelijkheden. De huidige symbolen zijn ontworpen door subcommittee 3A van Technical Committee 3 van het IEC (International Electrotechnical Commission) [IEC Standard 617-12; Kampel, 1985; Polis, 1981]. Er is een grammatica opgesteld om symbolen te construeren. In dit boek passen we deze IEC symbolen toe, vanwege de vele extra mogelijkheden van deze symbolen. Het volgende is een korte samenvatting van de regels voor het samenstellen van (poort)symbolen.

Vorm van de symbolen

De *grondvorm* van een symbool is een *rechthoekig kader*. De *afmetingen* ervan zijn vrij te kiezen en kunnen naar behoefte aan het aantal ingangen en het aantal uitgangen ervan aangepast worden. Ingangen worden links op het symboolkader getekend en de uitgangen rechts. De *richting van informatie-overdracht* is van 'links naar rechts'. Zo nodig mag ook van 'boven naar beneden' gewerkt worden, maar niet op één symbool van 'links naar rechts' en van 'boven naar beneden'. Zie figuur 4.31.



Figuur 4.31. Kader poortsymbolen.

In tekeningen komt het soms beter uit om de richting van de informatie-overdracht om te keren, vanwege de overzichtelijkheid van de tekening. Dit is toegestaan, mits met een pijltje in de verbindingen de richting wordt aangegeven.

Functionysymbolen

Binnen een symboolkader wordt 'informatie' altijd *logisch* doorgegeven. De logische functie van een symbool wordt aangeduid met een *functiesymbool* of *qualifying symbol* (* is de algemene aanduiding ervan). Voor de plaats van het functiesymbool in het kader zie figuur 4.32.a. Het functiesymbool geeft aan hoe de ingangsinformatie wordt bewerkt alvorens aan de uitgang te worden doorgegeven. Enkele voorbeelden van functiesymbolen staan in figuur 4.32.b.



(a) algemene vorm symbool met qualifying symbol

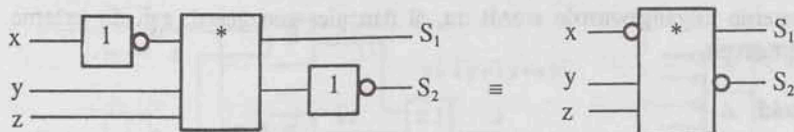
- | | |
|----------|---|
| & | logische AND: uitgang is 1 als alle ingangen 1 zijn |
| ≥ 1 | logische OR: uitgang is 1 als tenminste één ingang 1 is |
| 1 | uitgang is 1 als de ingang 1 is |
| =1 | EXOR operator: uitgang is 1 als precies één ingang 1 is |
| $= 2k+1$ | uitgang is 1 als een oneven aantal ingangen 1 is |
| $= 2k$ | uitgang is 1 als een even aantal ingangen 1 is |

(b) verschillende functiesymbolen met hun betekenis

Figuur 4.32. *Qualifying symbols.*

Het aanduiden van negaties

Een *negatie* aan een ingang of aan een uitgang keert de logische waarde ervan om (extern 0 $\leftrightarrow$ intern 1, enz.). Een negatie wordt met een cirkeltje tegen het symboolkader aangeduid. Dit cirkeltje heet de *negatie-indicator*. Ontbreekt de negatie-indicator aan een ingang of uitgang, dan wordt de logische waarde ongewijzigd doorgegeven. Zie figuur 4.33.



Figuur 4.33. *Het aanduiden van negaties.*

Figuur 4.34 toont de logische symbolen voor alle tot nu toe geïntroduceerde poorten.

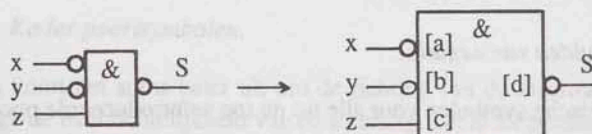
Naam	Logische functie	Symbool
AND	$S = y \cdot z$	
OR	$S = y + z$	
NOT	$S = \bar{z}$	
NAND	$S = \overline{y \cdot z}$	
NOR	$S = \overline{y + z}$	
EXOR	$S = y \oplus z$	

Figuur 4.34. Symbolen voor poorten.

Een symbool of een samenstelling van symbolen wordt als volgt geïnterpreteerd.

- Elke externe ingangscombinatie wordt omgezet in een interne ingangscombinatie. Hierbij keren negatie-indicatoren de logische waarde om.
- Het functiesymbool geeft de logische bewerking aan die wordt uitgevoerd op de interne logische waarden van de ingangen.
- Het resultaat van de bewerking wordt toegekend aan de interne uitgangswaarde.
- De interne uitgangswaarde wordt nu, al dan niet genegeerd, aan de externe uitgang doorgegeven.

Voorbeeld



Figuur 4.35. Symbool met negatie-indicatoren.

We beginnen met het opstellen van een waarheidstabel. In de eerste drie kolommen staan alle waarden van x , y en z .

x	y	z	a	b	c	d	S
0	0	0	1	1	0	0	1
0	0	1	1	1	1	1	0
0	1	0	1	0	0	0	1
0	1	1	1	0	1	0	1
1	0	0	0	1	0	0	1
1	0	1	0	1	1	0	1
1	1	0	0	0	0	0	1
1	1	1	0	0	1	0	1

$$S = \overline{\overline{x} \overline{y} z}$$

$$= x + y + \overline{z}$$

Tabel 4.9. Waarheidstabel voor input-output transfer.

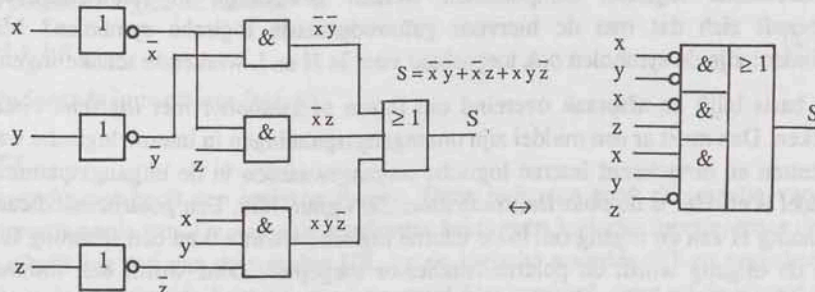
De waarden van x , y en z in tabel 4.9 zetten we om in de *interne logische variabelen* a , b en c . Daar dit *hulpvariabelen* zijn zetten we deze in IEC symbolen tussen vierkante haken. Op de variabelen a t/m c laten we de logische EN bewerking los. Dit levert de kolom van de *interne uitgang* d . De waarde van d wordt genegeerd doorgegeven aan de *externe uitgang* S van het symbool. Nu weten we het verband tussen x , y en z enerzijds en S anderzijds. De bij het symbool behorende logische functie is:

$$S = x + y + \overline{z} \quad \square (4.36)$$

Opmerking

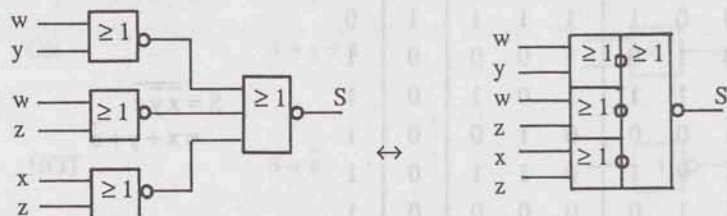
Alle *niet-genormaliseerde informatie* in een symbool staat tussen vierkante haken! Dit kan allerlei informatie betreffen, een aanduiding van het symbool met een naam of met een typenummer, een plaatsnummer, etc. Genormaliseerde informatie in het symbool staat niet tussen vierkante haken. Het onderscheid tussen genormaliseerde informatie (betekenis exclusief vastgelegd) en niet-genormaliseerde informatie is dus gemakkelijk te maken. $\square$

Embedded symbols



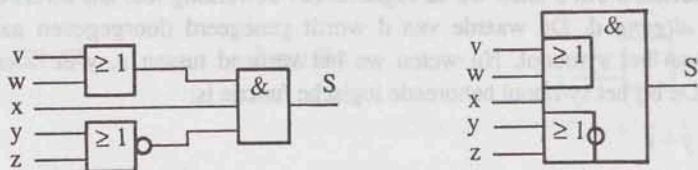
Figuur 4.36. Compacte tekenwijze.

Symboolkaders mogen tegen elkaar getekend worden. Er wordt dan informatie-overdracht verondersteld naar het kader erachter en wel in de richting van de *externe informatie-overdracht*. Figuur 4.36 geeft een toelichting. Bevindt zich daarbij op het *grensvlak* van twee poortsymbolen een *negatie-indicator*, dan wordt deze negatie-indicator dwars door de scheidslijn van de beide kaders getekend. Zie figuur 4.37.



Figuur 4.37. Negaties in grensvlakken.

Symbolen mogen ook in een ander kader getekend worden. Dat zijn de *embedded symbols*. Zie bijvoorbeeld figuur 4.38. Embedded symbols maken een tekening overzichtelijker. Veel externe verbindingen verdwijnen erdoor. De interne data-overdracht geschiedt altijd volgens de afgesproken regels.



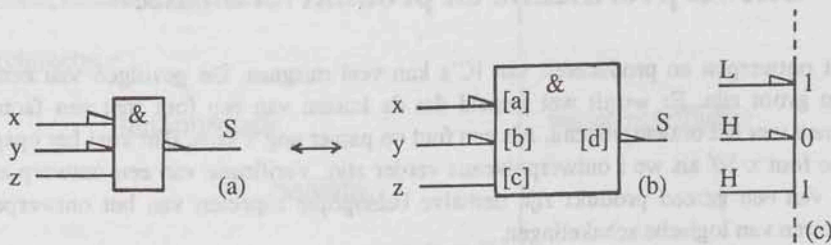
Figuur 4.38. Voorbeeld embedded symbols.

De 'grammatica' voor het opstellen van symbolen voor combinatorische schakelingen is met het bovenstaande beslist nog niet compleet. De rest zullen we waar nodig bij de betreffende onderwerpen introduceren. Zie ook [Polis, 1981; Kampel, 1985].

Signaaloverdracht op fysisch niveau

Elektronische logische componenten werken gewoonlijk in spanningsniveaus. Hoe verhoudt zich dat met de hiervoor geïntroduceerde logische symbolen? Met andere woorden: zijn de symbolen ook toepasbaar voor in H en L werkende schakelingen?

Als basis blijft de afspraak overeind om *intern in symbolen* met *logische* operatoren te werken. Dan moet er een middel zijn om ingangsspanningen in interne logische waarden om te zetten en omgekeerd interne logische uitgangswaarden in de uitgangsspanningen. Zo'n middel is er. Het is de *polariteitsindicator*. Zie figuur 4.39. Een polariteitsindicator zet een spanning H aan de ingang om in de interne logische waarde 0 en een spanning L in 1. Ook aan de uitgang wordt de polariteitsindicator toegepast. Daar wordt een interne logische waarde in een uitgangsspanning omgezet. Zonder polariteitsindicator aan een ingang of uitgang correspondeert H met 1 en L met 0, zowel aan de ingang als aan de uitgang.



Figuur 4.39. Symbolen met polariteitsindicator.

Bij het symbool in figuur 4.39 behoort tabel 4.10. De drie ingangssignalen x, y en z kunnen acht verschillende spanningscombinaties aanbieden. Deze combinaties worden omgezet in combinaties van interne logische waarden, uitgedrukt in de hulpvariabelen a t/m c.

extern			intern				extern
x	y	z	a	b	c	d	S
L	L	L	1	1	0	0	H
L	L	H	1	1	1	1	L
L	H	L	1	0	0	0	H
L	H	H	1	0	1	0	H
H	L	L	0	1	0	0	H
H	L	H	0	1	1	0	H
H	H	L	0	0	0	0	H
H	H	H	0	0	1	0	H

Tabel 4.10. Waarheidstabel.

De interne logische waarden a t/m c worden omgezet naar de interne uitgangswaarde d, met behulp van het functiesymbool &. Hieruit volgt de kolom onder d. Aan de uitgang staat de polariteitsindicator. Passen we deze toe, dan ontstaat de kolom onder S. Bij dit symbool behoort in positieve logica de formule

$$S = x + y + \bar{z} \quad (4.37)$$

Welke is de formule in negatieve logica?

Opmerking

De *negatie-indicator* heeft een logische functie. Deze indicator geeft de negatie van de logische signaalwaarde aan. De *polariteitsindicator* heeft **geen** logische functie, maar duidt slechts de afbeelding aan van de signalen H/L op de logische waarden 0/1 en omgekeerd. Als zodanig is de polariteitsindicator *logisch neutraal*. Het symbool stamt uit de wereld van spanningen en stromen en wordt dan ook exclusief toegepast op uitvoeringsschema's. □

4.10 Ontwerpverificatie en produktverificatie

Bij het ontwerpen en produceren van IC's kan veel misgaan. De gevolgen van een fout kunnen groot zijn. Er wordt wel gesteld dat de kosten van een fout met een factor 10 toenemen met het ontwerpniveau. Als een fout op papier nog x kost, dan kost het opsporen van die fout $x \cdot 10^1$ als we i ontwerpniveaus verder zijn. Verificatie van een ontwerp en het testen van een gereed produkt zijn derhalve belangrijke aspecten van het ontwerpen en produceren van logische schakelingen.

Fouten kunnen verschillende oorzaken hebben. Zo kan er in een ontwerp een fout zitten als gevolg van:

- een foute of incomplete architectuur;
- een foute implementatie, resulterend in fouten in het logisch ontwerp van de schakeling;
- een foute realisatie, als bij de uitwerking van het logisch ontwerp in een voorstel voor de schakeling fouten gemaakt zijn.

Voordat de produktie start dienen deze fouten uit het ontwerp te zijn. De activiteiten, welke nodig zijn om alle mogelijke fouten uit een ontwerp te halen, noemt men *ontwerpverificatie*.

Daarnaast worden er bij de produktie fouten gemaakt. In theorie zou het mogelijk moeten zijn foutloze produkten te maken. Zo'n productieproces is echter zeer kostbaar, zeker bij IC's, zodat men in de praktijk kiest voor een economische afweging tussen opbrengst (yield) van een proces en de kosten ervan [Ambler, 1992]. Een gevolg van deze beslissing is dat er tussen geproduceerde IC's altijd defecte exemplaren zitten. Om deze foute exemplaren uit de populatie van geproduceerde IC's te halen moet men *testen*.

Ontwerpverificatie

Ontwerpverificatie is een in principe *eenmalig te doorlopen traject*, dat begint met de verificatie van het programma van eisen en eindigt met het onderzoek van één of meer prototypes in de bedoelde toepassing van het produkt. De *verantwoordelijkheden* zijn duidelijk: die liggen bij de opdrachtgever voor zover het het programma van eisen betreft en bij de ontwerper voor zover het de uitwerking ervan tot en met een prototype betreft. In dit verband is het zinvol om *vooraf* met de opdrachtgever af te spreken wanneer het ontwerp of het produkt geaccepteerd wordt. Met name dient afgesproken te worden hoe het ontwerp geverifieerd gaat worden en hoe het eindprodukt getest gaat worden. De criteria voor goedkeuring moeten duidelijk zijn.

Een logisch ontwerper is medeverantwoordelijk voor de correctheid van het programma van eisen, alleen al vanuit het standpunt dat een tevreden opdrachtgever waarschijnlijk de volgende klant is. De logische ontwerper dient mee te denken over de vraag of de opdrachtgever zijn probleem/behoefte juist geformuleerd heeft. Zo nodig dient er om een verduidelijking en/of een aanvulling van het programma van eisen gevraagd te worden. Het ontwerpen volgt het volgende schema.

Requirements specificationarchitectuurimplementatierealisatieprototype IC ↓ontwerpverificatie

De *ontwerpverificatie* moet antwoord geven op de vraag of het ontwerp correct en volgens de bedoeling van de opdrachtgever op silicium gezet is. Om dit met succes te kunnen nagaan, binnen redelijke tijd, is het noodzakelijk dat ontwerpverificatie op *alle ontwerp-niveaus* wordt uitgevoerd. Bij voorkeur moeten de stappen hierbij niet te groot zijn. Geschiedt de verificatie over te grote ontwerp-stappen, dan is de fout als regel moeilijk op te sporen. Bij gebleken fout(en) moet gemiddeld ook een groter deel van het ontwerp herzien worden. Fouten zijn duurder naarmate ze later ontdekt worden!

Een van de eerste handelingen is de verificatie van het gekozen algoritme, met name de randvoorwaarden voor de toepassing ervan zoals effecten van afrondingsfouten bij numerieke bewerkingen en capaciteitsoverschrijding. Uit deze voorbeelden blijkt dat 'ontwerpen' en 'verificatie' voor een belangrijk deel *aanvullende activiteiten* zijn, die nooit helemaal te scheiden zijn.

Op logisch niveau houdt verificatie in dat onderzocht wordt of het eerste logische ontwerp, het *theoretisch logisch model*, correct is en niet conflicteert met het programma van eisen (requirements specification). Eindigt ook deze stap tot tevredenheid, dan wordt het logische model omgezet in een *realiseerbaar of direct afbeeldbaar logisch model*. Met name technologie-afhankelijke aspecten, zoals fanin beperkingen, worden nu verwerkt. Ook deze stap moet weer geverifieerd worden. Hierna wordt het logisch model op hardware afgebeeld. Het resultaat is een netwerk van transistoren en verbindingen. Op dit niveau geschiedt ontwerpverificatie meestal met behulp van simulatie via de netlist, dat is een formele beschrijving van het ontwerp op transistorniveau.

Vervolgens wordt de netlist omgezet in een layout. Vanuit de layout worden de parameters bepaald en kan worden vastgesteld of het ontwerp aan alle fysische eisen met betrekking tot snelheid, dissipatie en dergelijke voldoet. Hierna kan een prototype vervaardigd worden. De laatste stap is het uitproberen van dit prototype in de veronderstelde *applicatie-omgeving*.

Het is duidelijk dat een *top-down ontwerpstyl* een noodzakelijke voorwaarde is om ontwerpverificatie op *economisch verantwoorde wijze* te kunnen uitvoeren. Wanneer ontwerpverificatie over *kleine ontwerp-stappen* wordt uitgevoerd, dan hoeft er bij een fout slechts een klein deel van het ontwerp herzien te worden. Tevens valt de oorzaak van de fout dan gemakkelijker op te sporen. Ontwerpverificatie vereist tevens dat alle tussenliggende ontwerp-stappen goed gedocumenteerd zijn.

Produktverificatie of testen

De ontwerpverificatie eindigt met acceptatie van het ontwerp, of met de afkeuring ervan. Bij acceptatie kan de (serie)productie starten. Is de geproduceerde schakeling fout, dan moet het dus aan het productieproces liggen en niet aan het ontwerp.

Binnen de IC wereld is het gebruikelijk het testen van de IC's te baseren op het logisch ontwerp, of op de layout van de schakeling. Het testen zelf geschiedt door logische spanningen te meten onder een aantal bedrijfscondities. Daarnaast worden stromen gemeten en een aantal parameters. Het alternatief is IC's geheel of gedeeltelijk binnen hun applicatie-omgeving uit te proberen (applicatietesten). Deze aanpak geeft over het algemeen een geringere foutdekking en is meestal minder efficiënt.

In hoofdstuk 5 gaan we uitgebreider in op de relaties defect $\leftrightarrow$ afwijkend fysisch gedrag $\leftrightarrow$ afwijkend logisch gedrag. In de volgende paragrafen van dit hoofdstuk gaan we uit van de veronderstelling dat een *defect* een *afwijking van het logische gedrag* tot gevolg heeft en gaan we na hoe (bij kleine schakelingen) dit afwijkende gedrag gemeten kan worden. Het is van belang het testen vergaand te automatiseren en te optimaliseren. Immers, elk geproduceerd IC moet getest worden. Dit in tegenstelling tot de ontwerpverificatie, die meestal eenmalig is.

Correct-by-design

Bij het ontwerpen van digitale schakelingen wordt meer en meer gebruikt gemaakt van CAD (Computer Aided Design) software. Is zo'n software pakket correct en reeds langdurig in gebruik, dan zouden we erop mogen vertrouwen dat de betreffende ontwerpstappen foutloos worden uitgevoerd. De verificatie van het ontwerp wordt hiermee gedeeltelijk doorgeschoven naar de software. Is dit een juiste handelwijze? Illustratief is het volgende citaat uit [De Micheli, 1994]:

At a first glance, implementation verification should play a limited role when computer-aided synthesis methods are used. Indeed such methods should provide correctness by construction. This myth has fallen with the widespread use of synthesis systems. While synthesis algorithms have of course guaranteed properties of correctness, their software implementation (as well as the implementation of the operating systems and on which data bases they run) cannot be guaranteed to be error-free. Hence the need of implementation verification methods that support synthesis tools by performing validity checks.

4.11 Het testen van combinatorische schakelingen

Het uitgangspunt bij produktietesten is de aanname dat het ontwerp correct is. Het doel van het testen is zo efficiënt mogelijk goede van defecte IC's te onderscheiden. Daarbij is het 'zo maar' meten aan een IC of schakeling zinloos. Een testprocedure dient te berusten

op voorkennis over de wijze waarop het IC of printed circuit board ontworpen en gefabriceerd is en daarnaast op kennis van wat er fout kan gaan in het productieproces of bij de montage. Naast kennis over wat er fout kan gaan is ook kennis nodig over de kans van optreden van elke fout. Het eerste is nodig om een zo hoog mogelijke foutdekking te halen (fault coverage). De foutkansen zijn nodig om een testprocedure indien nodig te kunnen optimaliseren.

Defecten en fouten

Wanneer een produkt afwijkt van het ontwerp, dan is de oorzaak ervan meestal een defect. Een defect is de plaats waar en de wijze waarop de fysische structuur van de geproduceerde schakeling afwijkt van de ontworpen structuur, met als gevolg een afwijkend fysisch/logisch gedrag op die plaats. Dit afwijkende gedrag kan ook gevolgen hebben voor het fysisch/logisch gedrag op andere plaatsen.

Bij hardware defecten denken we onder andere aan een onderbroken verbinding (open), waardoor de geleider tussen twee punten verdwenen is, of aan een kortsluiting (short) tussen twee naburige geleiders. Het IC zal zich daardoor anders gedragen dan een foutloos IC. Dit afwijkende gedrag is meetbaar in de vorm van afwijkende spanningsniveaus en/of stroomsterktes, aan de dissipatie of aan de propagatietijden van de signalen door de schakeling. Daarnaast kan een defect zich ook manifesteren in de vorm van een hogere storingsgevoeligheid van het IC, bijvoorbeeld wanneer de interne logische niveaus afwijken van de gespecificeerde niveaus maar niet zoveel dat het tot een continu afwijkend logisch gedrag leidt. Zie het onderwerp AC en DC noise margins in de paragrafen 5.8 en 5.9.

Er zijn afwijkingen, die een heel IC beïnvloeden, bijvoorbeeld door het verkeerd uitlijnen van maskers. Dit meten we aan afwijkende parameters bij veel of alle signalen. Andere afwijkingen beperken zich tot een klein deel van het IC. Defecten worden daarom ingedeeld in lokale defecten (local defects) en globale defecten (global defects). Een lokaal defect, ook wel spot defect genoemd, beïnvloedt het IC op één plek. Het effect ervan wordt vaak elders gemeten. Een globaal defect beïnvloedt een groter deel van of het hele IC.

Defecten worden verder ingedeeld in hard defects en soft defects. Een hard defect is een defect dat zich continu manifesteert en het afwijkende gedrag is meestal gemakkelijk meetbaar. Bij een soft defect is dit niet zo. De afwijkingen zijn klein, de gevolgen ervan treden niet continu op en/of ze zijn moeilijk of niet extern meetbaar. Ze manifesteren zich ook wel als incidentele storingen.

We beperken ons hierna tot schakelingen met maximaal één hard defect. Daarvoor stellen we een foutmodel op. Dit is een model van wat er allemaal mis kan zijn in de schakeling als gevolg van het defect en waarbij tevens aangegeven wordt hoe we dit kunnen meten. Het bij een gegeven defect behorende afwijkende fysische of logische gedrag noemen we de fout (fault) van dat defect. Verschillende defecten kunnen tot dezelfde fout (fault) leiden. Deze defecten vormen dan één foutklasse (fault class). Het bestaan van foutklassen compliceert de diagnose, het onderzoek naar de plaats waar het defect zit dat bij een bepaalde fout behoort.

Een foutmodel voor combinatorische poortschakelingen

Poortschakelingen worden, volgens een nauwelijks uit te roeien traditie, gewoonlijk getest met het *stuck-at fault model* [Armstrong, 1966]. Een defect wordt daarbij gedacht een zodanig effect op de poortingang of poortuitgang te hebben dat de fout gemodelleerd kan worden met een vaste logische 0 of 1 op de betreffende ingang of uitgang.

Een *stuck-at-1* fout treedt op als er in een IC of op een printed circuit board bijvoorbeeld een kortsluiting (short) optreedt tussen een ingang en de positieve voedingsspanning V_{DD} . Idem voor een *stuck-at-0* fout bij een kortsluiting naar V_{SS} of aarde (ground). Dan is er sprake van een 'echte' hoge of lage spanning.

Daarnaast zijn er ook andere defecten, die zich als stuck-at fouten manifesteren. Bij veel 7400 TTL IC's trekken open ingangen automatisch naar een spanningsniveau dat als logische 1 geïnterpreteerd wordt. Open ingangen kwamen op printed circuit boards met TTL IC's veel voor, als gevolg van soldeerfouten (open contact) en draadbreek. Het stuck-at-0/1 foutmodel is daarmee inmiddels al vele jaren oud. Het is matig onderbouwd in die zin dat er veel andere foutklassen bestaan, maar het wordt desondanks nog veel toegepast [Beh, 1982; Tjin Koen Cheong, 1988; van Wijk, 1991].

Recent wordt het stuck-at-0/1 foutmodel vaak aangevuld met stuck-open fouten en andere fouten (hoofdstuk 5). Van alle bekende defecten is onderzocht welk gedeelte als stuck-at-0/1 fout modelleerbaar is. Gedetailleerd onderzoek heeft aangetoond [Beenker, 1994] dat 30-50% van alle defecten aldus modelleerbaar is. De *kwaliteit van een test* wordt onder andere uitgedrukt in het percentage van alle gemodelleerde fouten, dat met die test gedetecteerd wordt. *Het is duidelijk dat met 100% stuck-at foutendetectie het circuit niet foutvrij hoeft te zijn.* Voorlopig beperken we ons echter tot dit foutmodel.

Meestal veronderstelt men bij onderzoek aan het opzetten van testen dat er één defect in het IC zit. Dit maakt het genereren van testen gemakkelijker. Aanvullend onderzoek dient aan te tonen of deze generalisatie verantwoord is, met name of fouten elkaar kunnen maskeren. De kans hierop moet zeer klein zijn.

4.12 Testen op basis van het stuck-at-0/1 foutmodel

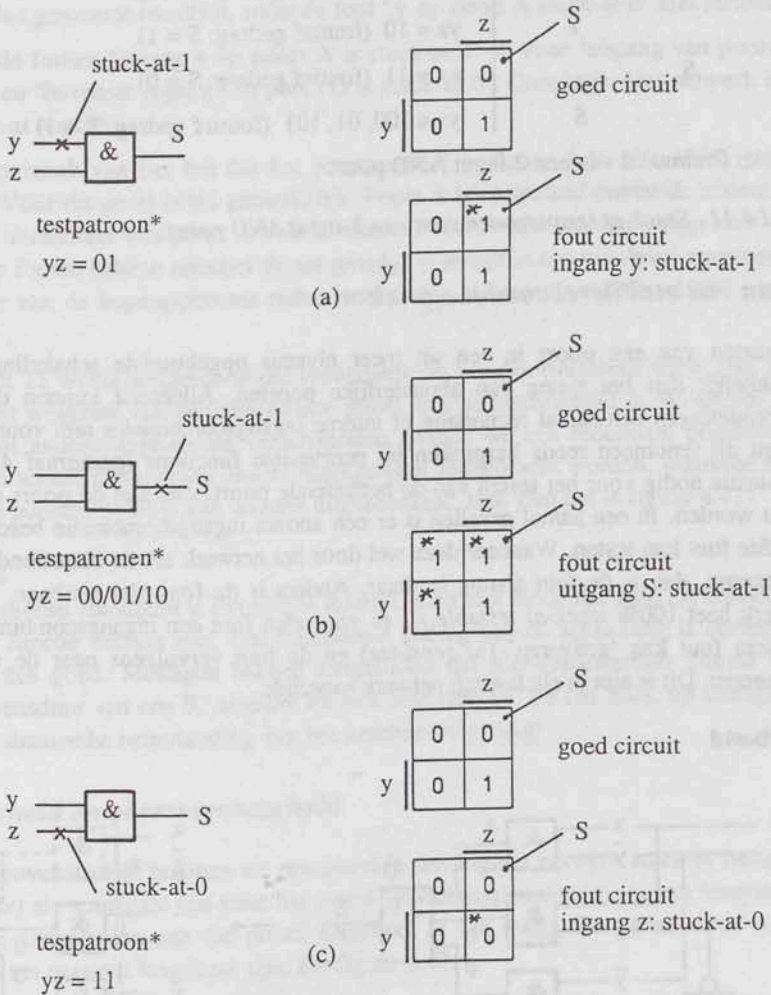
Een *testpatroon* is een ingangscombinatie, waarmee we onderscheid kunnen maken tussen een goed circuit en een circuit met één fout (hard defect) volgens het opgestelde foutmodel. In figuur 4.40 zijn een drietal fouten in een AND poort aangegeven. Voor elke fout is het logische gedrag van het goede circuit en van het circuit met de betreffende fout beschreven. Uit de Karnaughdiagrammen volgt dan welke ingangscombinatie(s) kan/kunnen dienen als testpatroon/testpatronen voor het betreffende defect. Sommige fouten moeten met één specifiek ingangspatroon getest worden, terwijl er voor andere fouten een keuze gemaakt kan worden uit verschillende testpatronen.

Tabel 4.11 geeft een overzicht voor alle stuck-at fouten van een AND poort. We zien dat een minimale set van testpatronen voor een 2-input AND poort is:

$$\text{Set van testpatronen AND: } yz \in \{01, 10, 11\} \quad (4.38)$$

We hoeven dus niet alle ingangscombinaties aan te bieden. Voor een 2-input OR poort is de minimale set van testpatronen

$$\text{Set van testpatronen OR: } yz \in \{00, 01, 10\} \quad (4.39)$$



Figuur 4.42. Testpatronen voor AND poort met stuck-at fouten.

Tabel 4.11 bevat de zes fouten, welke volgens het stuck-at 0/1 foutmodel in een 2-input AND poort kunnen voorkomen. Drie ervan worden getest met de ingangscombinatie $yz = 11$. In deze drie gevallen is de uitgang bij het optreden van één fout gelijk aan $S = 0$. Het is derhalve niet mogelijk door middel van input-output metingen deze drie fouten van elkaar te onderscheiden. Diagnose, waar zit de fout, is dus niet altijd mogelijk. De fouten $y/\text{stuck-at-0}$, $z/\text{stuck-at-0}$ en $S/\text{stuck-at-0}$ vormen één fault class.

Stuck-at-0	stuck-at-1	geschikt testpatroon/patronen
y		yz = 11 (foutief gedrag: S = 0)
	y	yz = 01 (foutief gedrag: S = 1)
z		yz = 11 (foutief gedrag: S = 0)
	z	yz = 10 (foutief gedrag: S = 1)
S		yz = 11 (foutief gedrag: S = 0)
	S	yz ∈ {00, 01, 10} (foutief gedrag: S = 1)

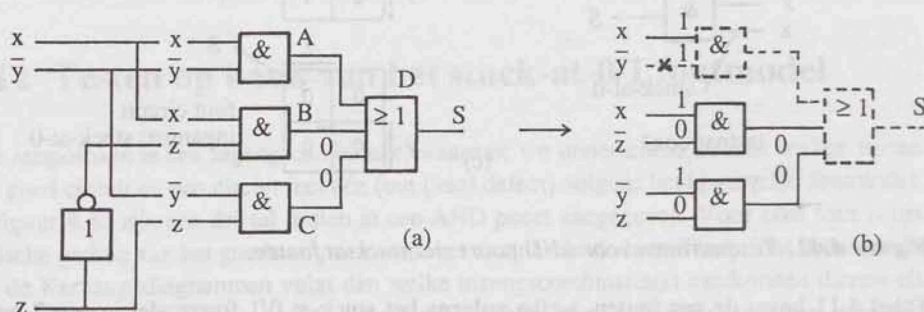
Stuck-at foutmodel van een 2-input AND poort.

Tabel 4.11. Stuck-at testpatronen voor een 2-input AND poort.

Testen van multilevel combinatoriek

Het testen van een poort in een uit meer niveaus opgebouwde schakeling is minder gemakkelijk dan het testen van afzonderlijke poorten. Allereerst kunnen door interne *vergrendelingen* een aantal secundaire of interne ingangscombinaties niet voorkomen. We hebben dit fenomeen reeds besproken bij permissible functions (paragraaf 4.7). Is zo'n combinatie nodig voor het testen van de betreffende poort, dan kan de poort niet volledig getest worden. In een aantal gevallen is er een andere ingangscombinatie beschikbaar, die dezelfde fout kan testen. Wanneer deze wel door het netwerk tot de betreffende poort kan propageren, dan is de fout alsnog testbaar. Anders is de fout niet testbaar. Een logisch netwerk heet 100% *stuck-at testable* als er voor elke fout een ingangscombinatie bestaat, die deze fout kan 'activeren' (*to sensitize*) en de fout vervolgens naar de uitgang laat propageren. Dit is niet in elk logisch netwerk mogelijk.

Voorbeeld



Figuur 4.41. Redundante schakeling.

De ingang $\bar{y}$ van de poort A in figuur 4.41.a wordt verondersteld stuck-at-0 te zijn. Deze fout is slechts observeerbaar aan de uitgang S van de schakeling als we aan $\bar{y}$ de waarde 1

toekennen. Om de fout naar de uitgang S te laten propageren moet de andere ingang van poort A de waarde $x = 1$ aannemen en moeten de andere twee ingangen van poort D de waarde 0 hebben. Gaan we de consequenties na, dan vinden we de in figuur 4.41.b toegewezen waarden. We zien nu een conflict ontstaan. De ingang $\bar{z}$ van poort B is 0, terwijl de ingang z van poort C ook 0 moet zijn. Er blijken geen andere ingangscombinaties te zijn met het gewenste resultaat, zodat de fout ' $\bar{y}$ op poort A stuck-at-0' niet testbaar is.

Idem voor de fouten 'ingang x op poort A is stuck-at-0' en voor 'uitgang van poort A is stuck-at-0' en 'bovenste ingang van poort D is stuck-at-0'. Conclusie: Het netwerk is niet 100% stuck-at testable.

Wat is de oorzaak van het feit dat het poortnetwerk in figuur 4.41 niet 100% stuck-at testable is? Voor dit geval is het gemakkelijk. Poort A is *redundant* omdat de andere twee poorten de mintermen van poort A ook bevatten. Poort A kan worden weggelaten en de niet-testbare fouten hebben effectief dit tot gevolg. Wanneer in een two-level poortnetwerk een of meer van de ingangspoorten redundant is, dan is het netwerk niet 100% stuck-at testable.

In hoofdstuk 11 wordt aangetoond dat redundantie in een poortnetwerk soms heel zinvol kan zijn. Het weglaten van een redundante poort, om de testbaarheid te verbeteren, leidt dan weer tot andere ongewenste verschijnselen. Willen we een redundant poortnetwerk toch kunnen testen, dan moeten *aparte testingangen* aangebracht worden, waarmee één of meer van de poorten tijdelijk kan worden uitgeschakeld. Hoe gaat dat in figuur 4.41?

Opmerking

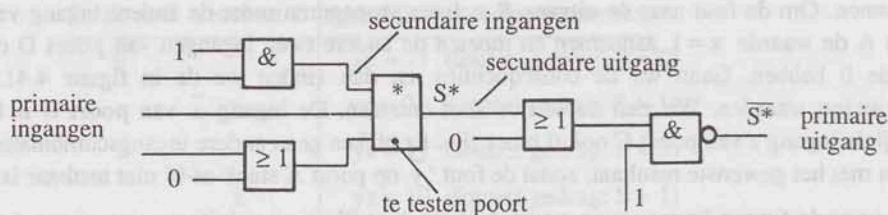
Een vaak gehoord standpunt is dat niet te testen fouten niet hinderlijk zijn. Zij beïnvloeden de logische functie immers niet. Het functionele gedrag van de schakeling is correct, het produkt is dus goed. Metingen hebben aangetoond dat niet-detecteerbare fouten onder meer de levensduur van een IC negatief kunnen beïnvloeden. En dat leidt, op termijn, wel tot een heel drastische beïnvloeding van het functionele gedrag! $\square$

Instelbaarheid en observeerbaarheid

Gezien het bovenstaande noemen we een poort in een logisch netwerk *stuck-at instelbaar* (*controllable*) als tenminste één voor het testen benodigde (minimale) set van testpatronen kan worden aangeboden aan die poort. Daarvoor is het nodig dat paden door eerdere poorten van het netwerk instelbaar zijn. Zie figuur 4.42.

Tevens moet er gelijktijdig tenminste één pad van de uitgang van de te testen poort naar een primaire uitgang van de schakeling zijn. Dan heet de uitgang van de betreffende poort *observeerbaar* (*observable*). Geldt dit voor alle poortuitgangen in een netwerk, dan noemen we het netwerk een *observable network*.

Een bekend algoritme om paden door een combinatorisch netwerk te activeren, zodanig dat een bepaalde poort/fout getest kan worden, is het *D algoritme* [Roth, 1966; Roth, 1967]. Voor een beschrijving van dit en meer recente *path-sensitization algorithms*, zie [Abramovici, 1990].



Figuur 4.42. Voorbeeld van een volledig instelbare en observeerbare poort.

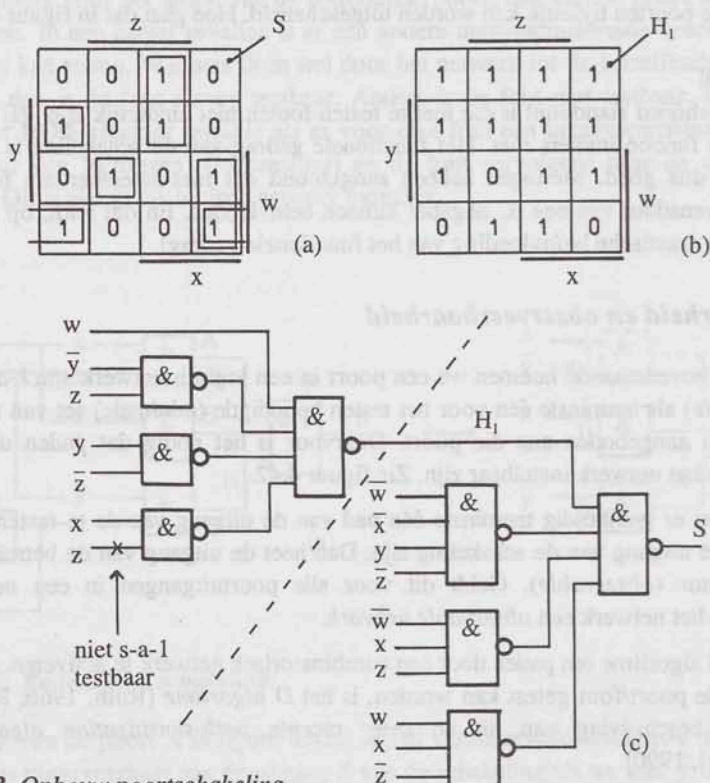
4.13 Permissible functions en stuck-at testbaarheid

In eerdere paragrafen hebben we de realisatie van de functie

$$S = (w \bar{x} y z + w \bar{y} z) + \bar{w} x y \bar{z} + \bar{w} x z + w x \bar{z} \quad (4.40)$$

als voorbeeld genomen. Zie figuur 4.16. De functie moet gerealiseerd worden met 4-input NAND's. Om deze realisatie mogelijk te maken is de hulpfunctie

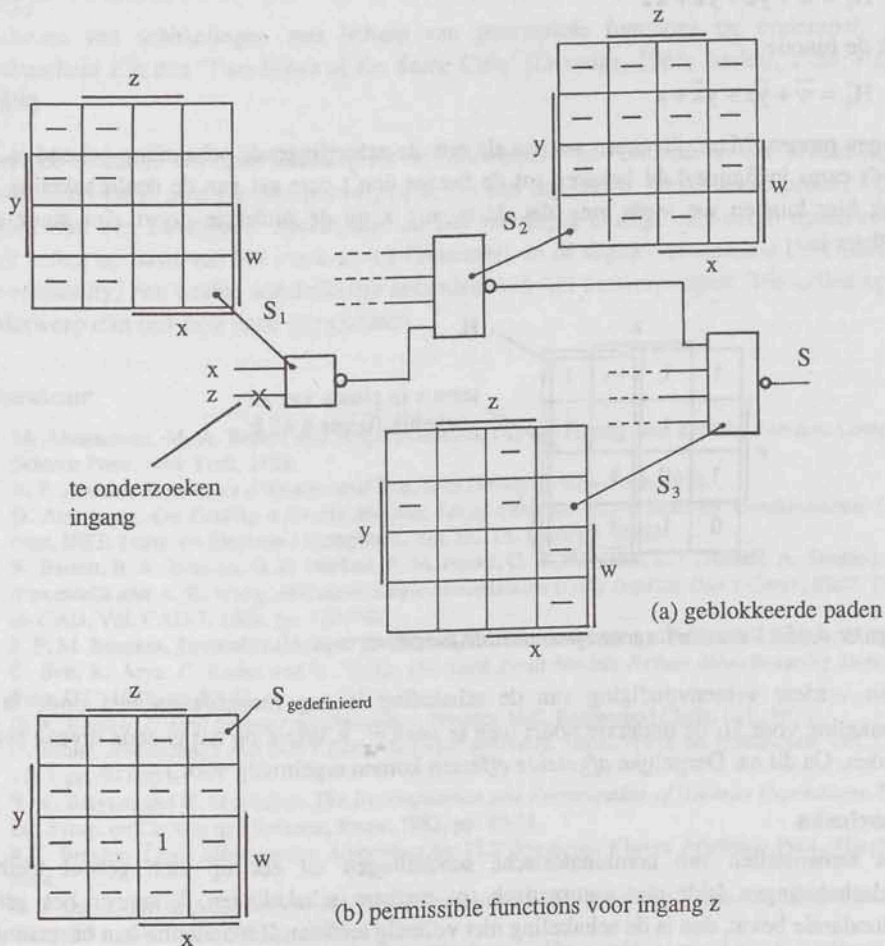
$$H_1 = \bar{w} \bar{x} y z + w \bar{y} z \quad (4.41)$$



Figuur 4.43. Opbouw poortschakeling.

ingevoerd. Zowel de functie S als de hulpfunctie H_1 zijn elk afzonderlijk in *niet-redundante* schakelingen gerealiseerd. Zie figuur 4.43.a/b.

Beide deelschakelingen in figuur 4.43, die voor de functie H_1 en de resterende schakeling voor de functie S met het signaal H_1 als ingangssignaal, zijn afzonderlijk *compleet stuck-at testbaar*. Samen zijn ze dat niet. De ingang z van de onderste poort van de deelschakeling H_1 is niet stuck-at-1 testbaar. Ga dit na door te proberen een pad in te stellen van deze ingang naar de uitgang S , waarbij $z = 0$ is om de fout 'z stuck-at-1' te kunnen testen. Dit blijkt niet te lukken voor het testpatroon van de fout, het patroon $xz = 10$.



Figuur 4.44. Afleiding permissible functions voor de ingang z .

In figuur 4.44 is afgeleid welke de verzameling van permissible functions is voor het ingangssignaal z op de onderste poort van deelschakeling H_1 . In de totale schakeling gaat het ingangssignaal z door drie NAND poorten. Voor iedere ingangscombinatie die een 0 op een van de ingangen van die poorten produceert is het ingangssignaal z don't care.

vorm van redundantie is redundantie in een gerealiseerde schakeling. Een gerealiseerde schakeling doet voor elke ingangscombinatie wat en bevat, als zodanig, geen don't cares in de ingangscombinaties.

Redundantie ten aanzien van testbaarheid kan worden voorkomen door van alle deelfuncties de verzameling van permissible functions op te stellen en daarin uitsluitend priemimplicanten voor een minimale dekking te nemen. Dus ook als een kleinere implicant (die minder enen omvat (NAND's) of minder nullen omvat (NOR's)) elders reeds gerealiseerd is en hergebruik van de poort tot besparingen kan leiden. Hergebruik van implicanten /poorten is riskant voor de stuck-at testbaarheid.

Reductie van schakelingen met behulp van permissible functions en onderzoek naar testbaarheid zijn dus 'Two Faces of the Same Coin' [Devadas, 1988; Barlett, 1988; Rajski, 1989]

Hiermee besluiten we onze inleiding tot het ontwerpen van combinatorische schakelingen. We hebben laten zien dat het ontwerpen ervan een top-down georiënteerd proces is, dat begint met een functionele specificatie en dat voorlopig eindigt met enige opmerkingen over testen op basis van het stuck-at-0/1 foutmodel. In de digitale techniek is DFT (design-for-testability) een beslist noodzakelijk onderdeel van het ontwerptraject. We zullen op dit onderwerp dan ook nog vaak terugkomen.

Literatuur

1. M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Testing and Testable Design*, Computer Science Press, New York, 1990.
2. A. P. Ambler, *Economics of Design and Test*, Ellis Horwood, New York, 1992.
3. D. Armstrong, *On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Nets*, IEEE Trans. on Electronic Computers, Vol. EC-15, 1966, pp. 66-93.
4. K. Barlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, C. R. Morrison, R. L. Rudell, A. Sangiovanni-Vincentelli and A. R. Wang, *Multilevel Logic Minimisation Using Implicit Don't Cares*, IEEE Trans. on CAD, Vol. CAD-7, 1988, pp. 723-740.
5. F. P. M. Beenker, *Testability Concepts for Digital ICs*, Kluwer Academic Publishers, Boston, 1995.
6. C. Beh, K. Arya, C. Radke and K. Torku, *Do Stuck Fault Models Reflect Manufacturing Defects?*, Proc. ITC, 1982, pp. 35-42.
7. G. A. Blaauw, *Digital System Implementation*, Prentice-Hall, Englewood Cliffs, N.J., 1976.
8. D. Brand, *Redundancy and Don't Cares in Logic Synthesis*, IEEE Trans. on Computers, Vol. C-32, 1983, pp. 947-952.
9. R. K. Brayton and C. McMullen, *The Decomposition and Factorization of Boolean Expressions*, Proc. Int. Symp. on Circuits and Systems, Rome, 1982, pp. 49-54.
10. R.K. Brayton, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publ., Hingham, 1984.
11. E. S. Davidson, *An Algorithm for NAND Decomposition under Network Constraints*, IEEE Trans. on Computers, Vol. C-18, 1969, pp. 1098-1109.
12. S. Devadas, H-K. T. Ma, A. R. Newton, A. Sangiovanni-Vincentelli, *Optimal Logic Synthesis and Testability, Two Faces of the Same Coin*, Proc. IEEE Int. Test Conference, 1988, pp. 4-11.
13. E. L. Dietmeyer and Y. H. Su, *Logic Design of Fanin Limited NAND Networks*, IEEE Trans. on Computers, Vol. C-18, 1969, pp. 11-22.
14. G. A. Dinklo, *Sleutelonderwerpen in de Informatietechnologie*, Samsom, Alphen aan den Rijn, 1994.
15. D. T. Ellis, *A Synthesis of Combinational Logic with NAND en NOR Elementen*, IEEE Trans. on Electronic Computers, Vol. EC-14, 1965, pp. 701-705.

16. J. F. Gimpel, *The Minimization of TANT Networks*, IEEE Trans. on Electronic Computers, Vol. EC-16, 1967, pp. 18-38.
17. L. Helleman, *A Catalog of Three-Variable Or-Invert and AND-Invert Logical Circuits*, IEEE Trans. on Electronic Computers, Vol. EC-12, 1963, pp. 198-223.
18. IEC Standard 617-12, *International Standard IEC 617-12 Graphical Symbols for Diagrams*, Genève, 1991, en latere aanvullingen.
19. IEEE, *Electrical and Electronics Graphic Symbols and Reference Designations*, IEEE Standard 315-1975, Institute of Electrical and Electronics Engineers, New York, 1976.
20. I. Kampel, *A Practical Introduction to the New Logic Symbols*, Butterworths, London, 1985.
21. E. Kjelkerud, *A Computer Program for the Synthesis of Switching Circuits by Decomposition*, IEEE Trans. on Computers, Vol. C-21, 1972, pp. 568-572.
22. H. P. Lee and E.S. Davidson, *A Transform for NAND Network Design*, IEEE Trans. on Computers, Vol. C-21, 1972, pp. 12-20.
23. G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994.
24. E. J. McCluskey, *Logical Design Theory of NOR Gate Networks with No Complemented Inputs*, Proc. 4th Ann. Symp. on Switching Theory and Logical Design, Chicago, 1963.
25. S. Muroga, Y. Kambayashi, H. C. Lai and J. M. Culliney, *The Transduction Method-Design of Logic Networks Based on Permissible Functions*, IEEE Trans. on Computers, Vol. C-38, 1989, pp. 1404-1424.
26. Z. Navabi, *VHDL, Analysis and Modelling of Digital Systems*, McGraw-Hill, New York, 1993.
27. NNI, Nederlands Normalisatie Instituut, *Nederlandse Praktijkrichtlijn NPR 5160*, 1994.
28. Philips, *High-Speed CMOS Databook*, Eindhoven, 1994.
29. L. R. Polis, *Logica Symboliek, Toelichting op het IEC Systeem*, Kluwer, Deventer, 1981.
30. J. Rajske, J. Vasudevamurthy, *Synthesis of Fully Testable Multilevel Circuits*, Fourth Technical Workshop in New Directions for IC Testing, Victoria, B.C., Canada, 1989, pp. 1-20.
31. J. P. Roth, *Diagnosis of Automata Failures: A Calculus and a Method*, IBM Journal of Research and Development, Vol. 10, 1966, pp. 278-291.
32. J. P. Roth, W. G. Bouricius and P. R. Schneider, *Programmed Algorithms to Compute Tests to Detect and Distinguish between Failures in Logic Circuits*, IEEE Trans. on Electronic Computers, Vol. EC-16, 1967, pp. 567-579.
33. H. Savoj and R. K. Brayton, *The Use of Observability and External Don't Cares for the Simplification of Multi-Level Networks*, 27th ACM/IEEE Design Automation Conference, 1990, pp. 297-301.
34. M. J. K. Tjin Koen Cheong, *Research into Fault Modeling Techniques*, Masters Thesis, Delft University 1-68340-28(1988)29, Delft, 1988.
35. A. P. Thijssen, H. A. Vink en C. H. Eversdijk, *Digitale Techniek, Van Probleemstelling tot Realisatie, deel I*, Delftse Uitgevers Maatschappij, Delft, 1988.
36. H. A. Vink, B. van den Dolder and J. Al, *Reduction of CC-tables using Multiple Implication*, IEEE Trans. on Computers, Vol. C-27, 1978, pp. 961-966.
37. H. A. Vink, *Minimal TANT Networks of Functions with Don't Cares and Some Complemented Input Variables*, IEEE Trans. on Computers, Vol. C-27, 1978, pp. 1073-1078.
38. L. van Wijk, *Defect Oriented Testing, a Method to Verify the Accuracy of Fault Models*, Masters Thesis, Delft University 1-68340-28(1991)02, Delft, 1991.

Opgaven hoofdstuk 4

4.1. Ontwerpen geschiedt bij voorkeur top-down. Men doet dit

- a. om de grenzen tussen architectuur, implementatie en realisatie beter te kunnen aangeven.
- b. omdat dan de ontwerpverificatie beter en meer gestructureerd kan plaatsvinden.
- c. omdat deze benadering beter aansluit bij het proces van kennisopbouw.
- d. omdat deze werkwijze altijd tot schakelingen van minimale omvang leidt.

4.2. Het ontwerpproces wordt onderscheiden in architectuur, implementatie en realisatie. Welk van de onderstaande beweringen is niet juist?

- Tijdens de implementatie worden de randvoorwaarden gespecificeerd, waarbinnen het ontwerp moet passen (fysisch, economisch, etc.).
- Tijdens de implementatie wordt onderzocht of het ontwerp in principe realiseerbaar is.
- Tijdens de implementatie wordt het gekozen algoritme nader uitgewerkt en wordt de toepasbaarheid ervan geverifieerd.
- Tijdens de implementatie speelt 'verificatie' een belangrijke rol.

4.3. Stel de logische formules op voor de som S_i en de carry C_{i+1} van een half adder en wel in de minimale produkt-van-sommenvorm. Beide formules bevatten in totaal

- zes variabelen.
- acht variabelen.
- achttien variabelen.
- een ander aantal dan de genoemde aantallen variabelen.

(N.B. De aanduidingen S_i en C_{i+1} niet meetellen. Komt een variabele twee keer voor, dan twee keer tellen.)

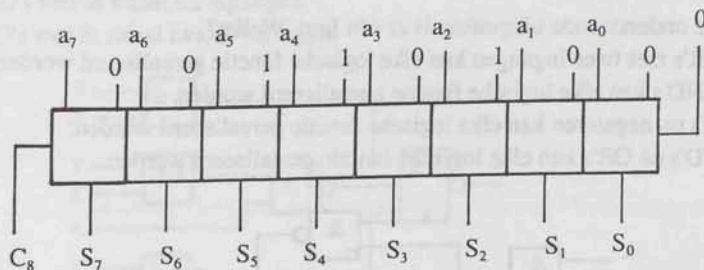
4.4. Bij een binair gecodeerd getal A,

$$A = a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0_{\text{BIN}}$$

moet een constante B,

$$B = 00110100_{\text{BIN}}$$

als 'offset' opgeteld worden. Hiervoor wordt een binaire opteller toegepast.



Voor de realisatie van deze schakeling zijn *minimaal* nodig

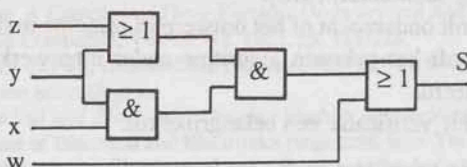
- acht full adders.
- zeven full adders en één half adder.
- drie full adders en vijf half adders.
- andere aantallen full adders en half adders.

4.5. In het theoretisch logisch model van een schakeling komt een 7-input AND poort voor. Ter beschikking staan AND poorten met maximaal drie ingangen. In hoeveel poorten/niveaus is de schakeling minimaal realiseerbaar?

- Twee poorten / drie niveaus.
- Drie poorten / twee niveaus.
- Drie poorten / drie niveaus.
- Vier poorten / drie niveaus.

(Bij meer goede antwoorden: Kies dat antwoord waarbij het aantal poorten en/of het aantal niveaus minimaal is.)

4.6. Bepaal de logische formule voor de volgende schakeling.



De minimale realisatie met 2-input AND en/of OR poorten kost

- één poort.
- twee poorten.
- drie poorten.
- vier poorten.

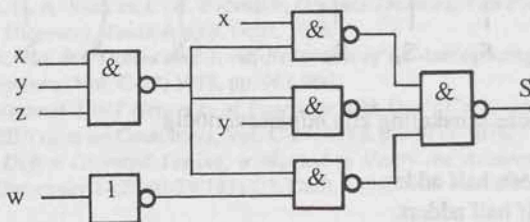
4.7. Soms bezit een poort (AND, OR, etc.) meer ingangen dan er nodig zijn. Voor deze ingangen geldt één van de vier onderstaande beweringen. Geef de juiste bewering aan.

- Niet gebruikte ingangen mogen bij geïntegreerde circuits altijd opengelaten worden.
- Niet gebruikte ingangen moeten via een weerstand met de hoogste voedingsspanning verbonden worden.
- Niet gebruikte ingangen moeten direct aan aarde gelegd worden of aan de laagste voedingsspanning.
- Geen van de drie vorige uitspraken is juist.

4.8. Van de onderstaande uitspraken is er één fout. Welke?

- Met NOR's met twee ingangen kan elke logische functie gerealiseerd worden.
- Met NAND's kan elke logische functie gerealiseerd worden.
- Met OR's en negatoren kan elke logische functie gerealiseerd worden.
- Met AND's en OR's kan elke logische functie gerealiseerd worden.

4.9.



De gegeven schakeling realiseert één van de volgende formules. Welke?

- $S = x\bar{y} + y\bar{z} + \bar{w}\bar{y}$.
- $S = (\bar{x} + y)(\bar{y} + z)(w + y)$.
- $S = xyz + w\bar{x}\bar{y}$.
- $S = \bar{x}y + x\bar{y} + y\bar{z} + \bar{w}\bar{y}$.

4.10. Gegeven is de logische functie S,

$$S = \bar{x}\bar{y} + yz + xz$$

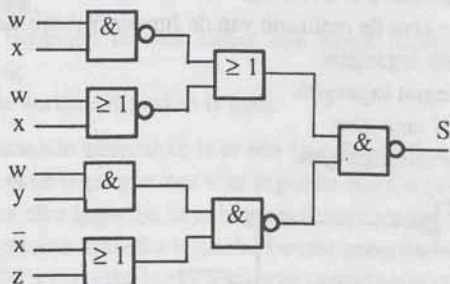
4.14.

	z			
	0	1	1	$\bar{w}$
y	0	—	1	1
	1	0	0	1
	—	1	—	—
	x			S

Bij de realisatie van de in het Karnaughdiagram gespecificeerde logische functie S worden uitsluitend NOR's gebruikt. De signalen w t/m z en $\bar{w}$ t/m $\bar{z}$ zijn beschikbaar. Bij een opbouw in twee niveaus kost de realisatie van de functie S minimaal

- vier NOR's met in totaal negen ingangen. (Ook interne ingangen tellen mee.)
- drie NOR's met in totaal acht ingangen.
- drie NOR's met in totaal zeven ingangen.
- drie NOR's met in totaal zes ingangen.

4.15.



De schakeling realiseert één van de volgende logische formules. Welke?

- $S = wx + wy.$
- $S = wx + w\bar{x} + wy + \bar{x}z.$
- $S = wx + \bar{w}\bar{x} + wy + \bar{x}z.$
- Geen van bovenstaande drie formules.

4.16. Voor de realisatie van de logische functie S ,

$$S = \bar{x}y + \bar{x}z + x\bar{y}$$

staan NAND's en NOR's ter beschikking, met één (negator), twee, drie of vier ingangen. De variabelen $\bar{x}$ t/m $\bar{z}$ zijn ook beschikbaar. In totaal kost de realisatie in NAND's of NOR's minimaal

- in NAND's: vier poorten met in totaal maximaal negen ingangen; in NOR's: drie poorten met in totaal maximaal zeven ingangen.
- in NAND's: vier poorten met in totaal tien ingangen; in NOR's: drie poorten met in totaal zeven ingangen.
- in NAND's: vier poorten met in totaal tien ingangen; in NOR's: vier poorten met in totaal acht ingangen.

d. in NAND's: vijf poorten met in totaal tien ingangen; in NOR's: drie poorten met in totaal acht ingangen.

4.17. Gegeven is de logische functie S,

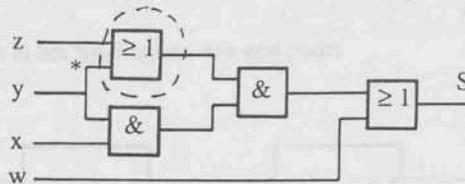
$$S = \overline{w} \overline{x} \overline{y} + w y \overline{z} + \overline{x} \overline{y} z + \overline{w} x y + x y \overline{z}$$

1
2
3
4
5

Voor de realisatie van deze functie S met 3-input NAND's is een decompositie van de formule nodig met haakjes, zodat drie 'termen' overblijven. Genegeerde variabelen zijn ook beschikbaar. Een gunstige decompositie is

- $S = (1 + 2) + 3 + (4 + 5).$
- $S = (1 + 3) + 2 + (4 + 5).$
- $S = (1 + 3) + (2 + 5) + (4 + 5).$
- Er bestaat een nog gunstiger decompositie.

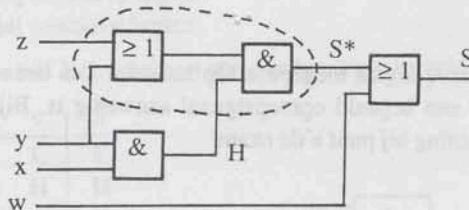
4.18.



Gegeven is een uit vier poorten bestaande schakeling. Bepaal de verzameling van permissible functions voor de met een * aangeduide *ingang* van de OR poort. Houd daarbij zowel rekening met ingangsdon't cares als met uitgangsdon't cares van de OR poort in het netwerk. Specificeer deze verzameling van permissible functions in een Karnaughdiagram voor vier variabelen. Hoeveel enen staan er in dit Karnaughdiagram?

- Twee.
- Vier.
- Twaalf.
- Een ander aantal dan onder a t/m c genoemd.

4.19.



In de gegeven schakeling is een deelschakeling omcirkeld. Deze deelschakeling heeft als ingangssignalen de primaire ingangssignalen y en z en het secundaire ingangssignaal H. In het uitgangspad voor de secundaire uitgang S* naar de uitgang S komt verder de primaire ingangsvariabele w voor.

Bepaal in een Karnaughdiagram in de variabelen w, y, z en H de verzameling van permissible functions voor de omcirkelde deelschakeling. Welk Karnaughdiagram is dat?

a.

	z			
	0	0	1	1
y	0	0	1	1
	—	—	—	—
	—	—	—	—
	H			

w

b.

	z			
	0	0	—	—
y	0	0	1	1
	—	—	1	1
	—	—	—	—
	H			

w

c.

	z			
	0	0	—	—
y	0	0	1	1
	—	—	—	—
	—	—	—	—
	H			

w

- d. Geen van de drie gegeven Karnaughdiagrammen is geheel correct.

4.20. Gegeven is de functie S,

$$S = w + x + \bar{y} + \bar{z}$$

In deze formule voor S wordt variabele $\bar{y}$ vervangen door een andere term. Van de onderstaande functies

$$S_1 = w + x + \text{NOR}(w, x, y) + \bar{z}$$

$$S_2 = w + x + \text{AND}(y, \bar{z}) + \bar{z}$$

$$S_3 = w + x + \text{NOR}(w, x, y, \bar{z}) + \bar{z}$$

is/zijn er

- geen gelijk aan S.
- één gelijk aan S.
- twee gelijk aan S.
- drie gelijk aan S.

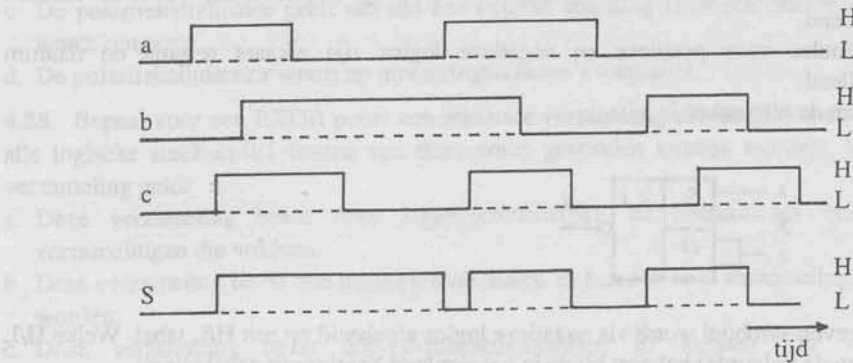
4.21. In een tekening is positieve logica toegepast. Op het punt x in het schema staat een spanningsniveau 'Laag' indien een bepaald oproepsignaal aanwezig is. Bij toepassing van positieve logica staat in de tekening bij punt x de naam

- OPROEP.
- $\overline{\text{OPROEP}}$
- OPROEP(H).
- OPROEP(L).

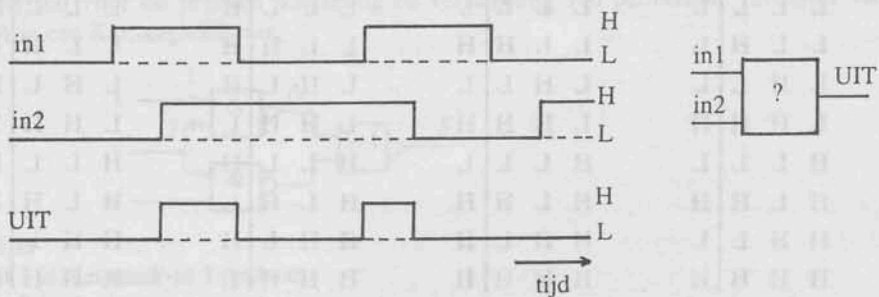
4.22. Gegeven is een tijddiagram van een schakeling met ingangssignalen a, b en c en uitgangssignaal S. Wanneer *negatieve logica* wordt toegepast, dan is de logische functie van deze schakeling

- $S = ab + \bar{a}c$.

- b. $S = ac + \bar{a}b$.
 c. $S = \bar{a} + b + c$.
 d. S is een andere dan de onder a t/m c gespecificeerde functies.



4.23. Gegeven is het tijddiagram van een poort.

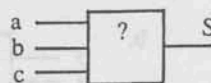


Dit is het tijddiagram van een

- a. OR poort bij positieve logica.
 b. OR poort bij negatieve logica.
 c. AND poort bij negatieve logica.
 d. NOR poort bij positieve logica.

4.24. Gegeven is de H/L tabel van een combinatorische schakeling.

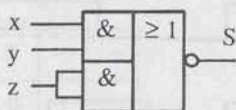
a	b	c	S
L	L	L	L
L	L	H	H
L	H	L	H
L	H	H	L
H	L	L	H
H	L	H	L
H	H	L	L
H	H	H	H



De werking van deze schakeling kan ook in een logische formule worden uitgedrukt. Men kan zich daarbij baseren op positieve of negatieve logica, hetgeen verschillende formules kan opleveren. Over deze bouwsteen worden enkele uitspraken gedaan. Welke is juist?

- De formules voor positieve en negatieve logica zijn identiek.
- De formules voor positieve en negatieve logica zijn elkaar duale vorm en daarom verschillend.
- De formules voor positieve en negatieve logica zijn elkaars negatie en daarom verschillend.
- Geen van de uitspraken is juist.

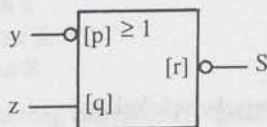
4.25.



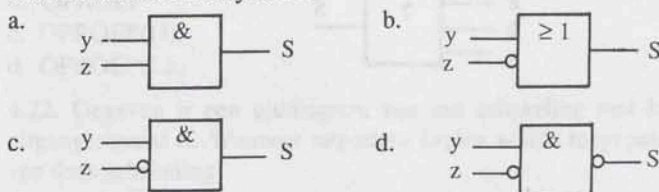
Het beschreven symbool wordt via negatieve logica afgebeeld op een H/L tabel. Welke H/L tabel (let op de volgorde) behoort bij de in het symbool beschreven schakeling?

a.	x	y	z	S	b.	x	y	z	S	c.	x	y	z	S	d.	x	y	z	S
	L	L	L	L		L	L	L	L		L	L	L	H		L	L	L	H
	L	L	H	L		L	L	H	H		L	L	H	H		L	L	H	L
	L	H	L	L		L	H	L	L		L	H	L	H		L	H	L	H
	L	H	H	H		L	H	H	H		L	H	H	L		L	H	H	L
	H	L	L	L		H	L	L	L		H	L	L	H		H	L	L	H
	H	L	H	H		H	L	H	H		H	L	H	L		H	L	H	L
	H	H	L	L		H	H	L	H		H	H	L	H		H	H	L	L
	H	H	H	H		H	H	H	H		H	H	H	L		H	H	H	L

4.26. Gegeven is een symbool volgens het IEC-systeem.



De interne signalen hebben hierbij een bepaalde logische waarde, die afhangt van de externe logische signaalwaarden en het functiesymbool. Dezelfde externe logische werking wordt beschreven door het symbool



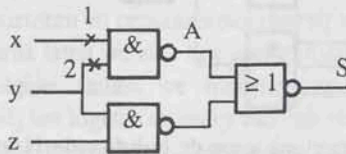
4.27. Over de polariteitsindicator worden vier beweringen gedaan. Welke ervan is juist?

- De polariteitsindicator geeft de polariteit van een signaal aan.
- De polariteitsindicator wordt toegepast op schema's in negatieve logica, om een onderscheid te kunnen maken met schema's in positieve logica.
- De polariteitsindicator geeft aan dat een externe spanning H in een interne logische 1 wordt omgezet.
- De polariteitsindicator wordt op uitvoeringsschema's toegepast.

4.28. Bepaal voor een EXOR poort een minimale verzameling van testpatronen, waarmee alle logische stuck-at-0/1 fouten van deze poort gevonden kunnen worden. Voor deze verzameling geldt:

- Deze verzameling bevat twee ingangscombinaties. Er bestaan zes verschillende verzamelingen die voldoen.
- Deze verzameling bevat drie ingangscombinaties. Er kan één zo'n verzameling gevonden worden.
- Deze verzameling bevat drie ingangscombinaties. Er kunnen vier van zulke verzamelingen gevonden worden.
- Deze verzameling bevat noodzakelijk alle vier de ingangscombinaties.

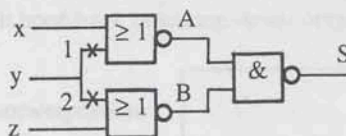
4.29. Bepaal voor de gegeven schakeling de verzameling van permissible functions van poort A in een Karnaughdiagram.



Dan blijkt

- Punt 1 is niet stuck-at-1 testbaar.
- Punt 2 is niet stuck-at-1 testbaar.
- De punten 1 en 2 zijn niet stuck-at-0 testbaar.
- Geen van de drie beweringen is juist.

4.30. Bepaal voor de gegeven schakeling de verzameling van permissible functions voor de poorten A en B in Karnaughdiagrammen.



Hieruit volgt

- Punt 1 is niet stuck-at-1 testbaar.
- Punt 1 of punt 2 is niet stuck-at-1 testbaar.
- De punten 1 en 2 zijn niet stuck-at-0 testbaar.
- Geen van de drie beweringen is juist.

(N.B. Er treedt maximaal één fout tegelijk op. Dat wil zeggen dat als punt 1 stuck-at-0/1 is het punt 2 normaal functioneert.)

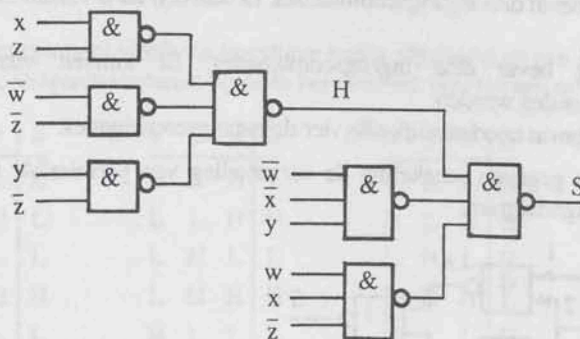
4.31 In een combinatorische schakeling blijkt een niet testbare stuck-at-1 fout te kunnen zitten. De betreffende poortingang moet dan in het ontwerp vervangen worden door een

- logische 0.
- logische 1.
- logische 0 of logische 1, afhankelijk van het soort poort.
- In het ontwerp dient een kleinere poort geplaatst te worden, waarbij de betreffende ingang is weggelaten.

4.32. Voor de realisatie van de logische functie S,

$$S = (\bar{x}z + w\bar{y}\bar{z}) + \bar{w}\bar{x}y + wx\bar{z}$$

staan NAND's ter beschikking met maximaal drie ingangen. De functie wordt op basis van de aangegeven decompositie gerealiseerd in de volgende schakeling.



Bepaal de verzameling van permissible functions voor de hulpfunctie H in een Karnaugh-diagram. Hieruit blijkt dat

- ingang z van de bovenste ingangspoort van de hulpfunctie H niet stuck-at-0 testbaar is.
- ingang $\bar{w}$ van de tweede ingangspoort van de hulpfunctie H niet stuck-at-1 testbaar is.
- twee ingangen niet stuck-at testbaar zijn.
- de bovenste ingangspoort van de hulpfunctie H overbodig is.

Stelling 4

De ene don't care is de andere niet

5

Het fysisch ontwerpniveau

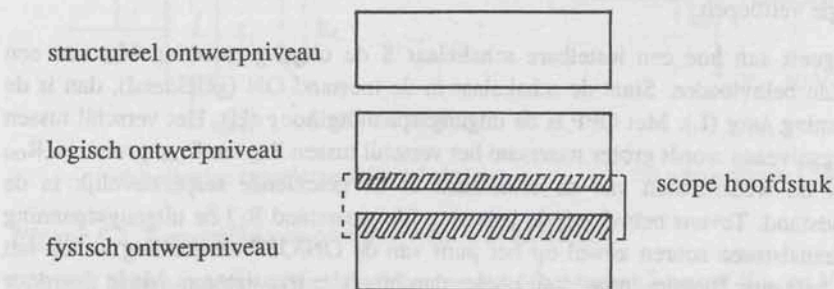
5.1 Het schakelen van spanningen

Bij het realiseren van een logisch ontwerp in een schakeling wordt de ontwerper geconfronteerd met de vraag wanneer en onder welke condities en randvoorwaarden het gespecificeerde logische gedrag ook daadwerkelijk door de ontworpen schakeling gerealiseerd wordt. Het onderwerp van dit hoofdstuk is een kwalitatieve verkenning van regels en voorwaarden, voortkomend uit de toegepaste technologie, die nodig zijn om een schakeling correct te doen functioneren. Daartoe is het nodig in te gaan op de elektronische eigenschappen van de toegepaste componenten.

Als eerste behandelen we enkele eigenschappen en kenmerken van logische families, met name van de bipolaire en de CMOS families. We gaan kort in op de werking van de toegepaste transistoren en beperken ons daarbij tot de externe werking in een toepassing als schakelaar. Daarna laten we zien hoe met transistoren logische poorten gerealiseerd kunnen worden. Vervolgens stellen we modellen op van de fysische eigenschappen van de componenten. Bij het logisch ontwerp van een schakeling hoeven we ons dan niet telkens in de fysische details te verdiepen. Met name krijgt het opstellen van een werkbaar model voor de timing van signalen en componenten veel aandacht.

Tot slot gaan we verder in op de testbaarheid van de gerealiseerde schakeling. Daarbij onderbouwen we dat voor een goed testbare schakeling het onderzoek naar de testbaarheid op het technologieniveau dient te beginnen met defect modellering, omvattend de analyse van de wijze waarop het defect de werking van de schakeling beïnvloedt. Meer en meer moet op het logische ontwerpniveau ingespeeld worden op de fysische eigenschappen van de componenten en, in het geval van testen, ook van hun defecten.

De plaats van dit hoofdstuk in het top-down ontwerptraject is als volgt.



We beperken de bespreking van de technologie tot het hoogst noodzakelijke. Voor meer details verwijzen we naar de bestaande handboeken [Shoji, 1988; Veendrick, 1990; Weste, 1993]. Het is echter niet mogelijk om logica te ontwerpen zonder enige kennis van de toegepaste technologie. De AND/OR $\rightarrow$ NAND/NOR problematiek is hier een goed voorbeeld van. De huidige trend in de industriële ontwerppraktijk is dat steeds meer van de technologie in silicon compilers voor de ontwerper verborgen blijft. Alleen bij het ontwerpen van high performance logica zal de ontwerper zich in de technologie moeten verdiepen.

Schakelaars

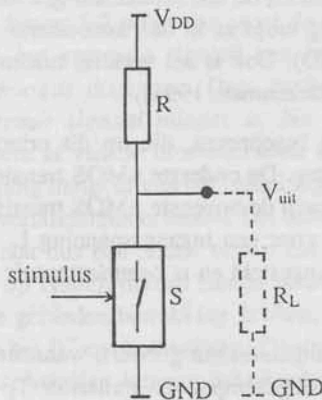
Een *ideale schakelaar* heeft in doorlaat een weerstand van $0\ \Omega$ en gesperd een oneindig hoge weerstand. Zo'n schakelaar kan willekeurig grote stromen schakelen, waarbij de schakelaar geen energie dissipeert. Daarnaast mag over de geopende schakelaar een willekeurig grote spanning staan. Tevens vertraagt en/of vervormt een ideale schakelaar het door te geven signaal niet. Dergelijke ideale schakelaars bestaan niet. Mechanische schakelaars benaderen dit ideaal het meest, maar zij vragen veel ruimte en zijn door hun bewegende delen onderhevig aan slijtage.

In de digitale techniek wordt meestal met transistoren geschakeld. De redenen hiervoor zijn de geringe afmetingen van transistoren, de veel hogere schakelsnelheden en de afwezigheid van bewegende delen. Als schakelaar toegepaste transistoren hebben echter een niet-ideaal gedrag. De weerstand in de doorlaatrichting is zeker geen $0\ \Omega$ en gesperd loopt er soms een zekere lekstroom. De dissipatie van de transistor-schakelaar is daarom niet te verwaarlozen. Evenzo is er sprake van enige signaalvertraging.

In *bipolaire transistoren* wordt de emitter-collectorstroom mede bepaald door de ingestelde basisstroom. Bij een geleidende transistor moet deze stroom in stand gehouden worden. Het in geleiding sturen en houden van een bipolaire transistor vergt voortdurend energie. In *MOS transistoren* moet er een bepaalde spanning op de gate staan om de stroom tussen drain en source in stand te houden. Is deze spanning eenmaal opgebouwd, dan is er om de stroom I_{DS} door de transistor in stand te houden geen energie meer nodig voor de instelling van het signaalniveau op de gate. Dit voorbeeld van hoe de technologie invloed heeft op de eigenschappen van de schakeling, in dit geval het mogelijk verschil in dissipatie tussen bipolair en CMOS, geeft aan dat het zinvol is dat we ons in de keuze van de technologie verdiepen.

Figuur 5.1 geeft aan hoe een instelbare schakelaar S de uitgangsspanning V_{uit} van een schakeling kan beïnvloeden. Staat de schakelaar in de toestand ON (geleidend), dan is de uitgangsspanning *laag* (L). Met OFF is de uitgangsspanning *hoog* (H). Het verschil tussen beide uitgangsniveaus wordt groter naarmate het verschil tussen R_{ON} en R_{OFF} groter is. R_{ON} en R_{OFF} zijn de weerstanden van de schakelaar in de geleidende respectievelijk in de geopende toestand. Tevens beïnvloedt de belasting (de weerstand R_L) de uitgangsspanning V_{uit} . MOS transistoren scoren zowel op het punt van de ON/OFF verhouding als op het punt van de belasting (hogere impedantie) beter dan bipolaire transistoren. Mede daardoor hebben we in de technologie een verschuiving gezien van de bipolaire naar de MOS

technologie. Een ander voordeel van CMOS is de hogere dichtheid van poorten op het IC oppervlak.



$$V_{uit} = \frac{R_{ON} // R_L}{R_{ON} // R_L + R} V_{DD}$$

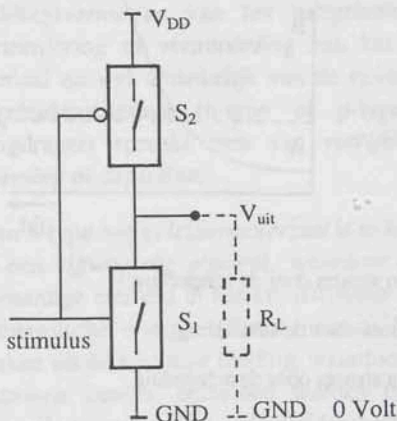
$$\text{met } R_{ON} \ll R \rightarrow V_{uit} \approx 0 \text{ V}$$

$$V_{uit} = \frac{R_{OFF} // R_L}{R_{OFF} // R_L + R} V_{DD}$$

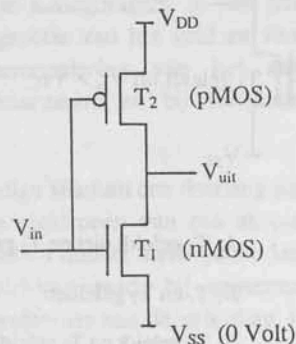
$$\text{met } R_{OFF} \gg R + R_L \text{ en } R_L \gg R \rightarrow V_{uit} \approx V_{DD}$$

Figuur 5.1. Het beïnvloeden van een spanningsniveau met een schakelaar.

Kenmerkend voor de in figuur 5.1 beschreven opstelling is dat er een groot verschil in eigenschappen bestaat voor de twee toestanden van de schakelaar. Is de schakelaar ingesteld op ON, dan loopt er een zekere stroom van V_{DD} naar aarde (ground/GND). Deze stroom is veel groter dan wanneer de schakelaar in de toestand OFF staat, met als gevolg een *hogere dissipatie* in de weerstand R bij de toestand ON ($R_L \gg R$ verondersteld). Is de weerstand R_L niet te verwaarlozen, dan heeft R_L ook invloed op de H/L (V_{ON}/V_{OFF}) verhouding van V_{uit} .



(a) principe complementaire schakelaar



(b) uitvoering in CMOS

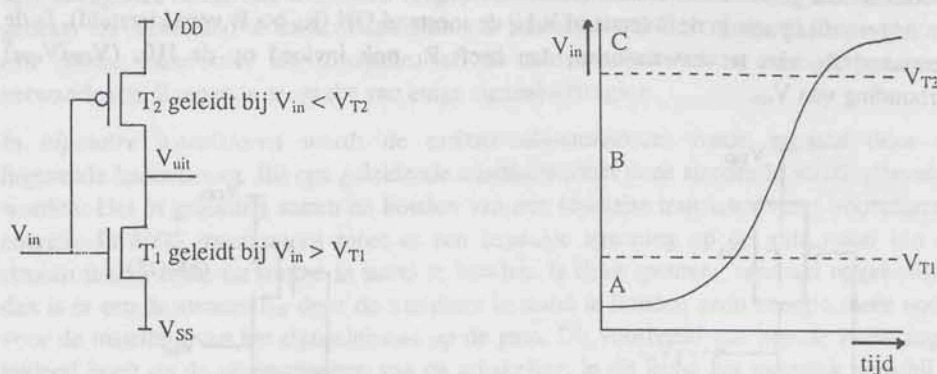
Figuur 5.2. Toepassing van complementaire schakelaars.

Wanlass heeft aangegeven dat een beter resultaat (het verschil tussen V_{ON} en V_{OFF}) ontstaat door gebruik te maken van twee *complementair werkende schakelaars*. Het principe van

complementaire schakelaars is aangegeven in figuur 5.2.a. De negatie-indicator op de bovenste schakelaar geeft aan dat de werking bij dezelfde externe stimulus ten opzichte van de onderste schakelaar complementair is, dat wil zeggen dat als de ene schakelaar geleidend is de andere geopend is, en andersom. In zo'n schakeling loopt er in de rusttoestand (de statische toestand) geen stroom van V_{DD} naar aarde (GND). Ook is het verschil tussen de uitgangsspanningen V_{ON} en V_{OFF} groter [Wanlass, 1963; Riezenman, 1991].

In figuur 5.2.b is een complementaire MOS schakeling beschreven, die op dit principe berust. Beide transistoren hebben een tegengestelde werking. De onderste nMOS transistor T_1 gaat geleiden bij een ingangsspanning H op de gate, terwijl de bovenste pMOS transistor T_2 er dan juist mee stopt. Het omgekeerde is het geval voor een ingangsspanning L. De uitgangsspanning wordt beurtelings door T_1 of door T_2 ingesteld en is *complementair ten opzichte van het ingangsniveau*.

Figuur 5.3 geeft gedetailleerder weer wat er in de *overgangstoestand* gebeurt, wanneer de ingangsspanning van laag (L) naar hoog (H) gaat. De omslagdrempel van transistor T_1 ligt op het spanningsniveau V_{T1} . Boven deze spanning geleidt deze transistor, eronder niet. De omslagdrempel van de complementaire transistor T_2 is V_{T2} . Hierboven geleidt T_2 niet, eronder wel. *Tussen de beide omslagdrempels is er dus een gebied waarin beide transistoren geleiden, met als gevolg dat er in de overgangstoestand van $L \rightarrow H$ een relatief grote stroom door de schakeling loopt.*



A: T_1 geleidt niet en T_2 geleidt $\rightarrow$ geen stroom door de schakeling

B: T_1 en T_2 geleiden $\rightarrow$ stroom door de schakeling

C: T_1 geleidt en T_2 geleidt niet $\rightarrow$ geen stroom door de schakeling

Figuur 5.3. Geleiding bij complementaire schakelaars.

Ook moeten we de belasting van een uitgang in rekening brengen. Hoe hoger de (capacitieve) belasting is, des te trager verloopt een uitgangsverandering (RC tijd). In het algemeen worden MOS transistoren op hun gate gestuurd. Deze vormen in de statische toestand, dat is de toestand waarin onder andere de gate capaciteiten opgeladen zijn,

effectief geen belasting voor de sturende schakeling. Dan is er *in de statische toestand* ook geen invloed meer op het uitgangsniveau van de sturende schakeling.

Uit figuur 5.3 volgt dat naast de eigenschappen van de schakelaar ook de *eigenschappen van het sturende signaal* van belang zijn. Tijdens het schakelen is er sprake van een *verhoogde dissipatie*. Deze verhoging wordt groter naarmate de flanksteilheid van het sturende signaal minder is. Nu hangt de flanksteilheid van het sturende signaal onder andere af van de bronweerstand en van de belasting van de uitgang. De belasting van de uitgang hangt af van het aantal ingangen dat het sturende signaal instelt en de capaciteit van de verbindingen ertussen. Dit aantal ligt vast in het logisch ontwerp van de schakeling. Er bestaat dus een relatie tussen het logisch ontwerp van de schakeling enerzijds en de eisen die op fysisch niveau aan de schakeling gesteld worden anderzijds. Deze eisen kunnen op vele gebieden betrekking hebben, waaronder de snelheid van de schakeling, het oppervlak van het IC en de dissipatie. Daarnaast verdienen ook factoren die de *betrouwbaarheid van de schakeling* kunnen beïnvloeden aandacht, met name marges voor ruis op de logische signaalniveaus. (Ruis te zien als een verzamelnaam voor alle versturende invloeden op de niveaus, waaronder door steile flanken uitgezonden stoorsignalen.) Om deze redenen moet de logisch ontwerper zich ook oriënteren op enkele fysische aspecten van het ontwerp.

5.2 De MOS transistor als schakelaar

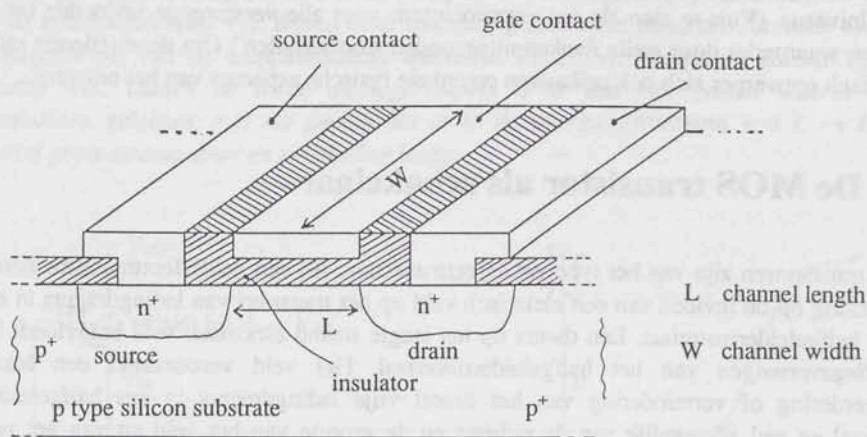
MOS transistoren zijn van het type *veldeffecttransistor*. Bij een veldeffecttransistor berust de werking op de invloed van een elektrisch veld op het transport van ladingdragers in een laagje halfgeleidermateriaal. Een dwars op het laagje staand elektrisch veld beïnvloedt het *geleidingsvermogen* van het halfgeleidermateriaal. Het veld veroorzaakt een lokale vermeerdering of vermindering van het aantal vrije ladingdragers in het halfgeleidermateriaal en wel afhankelijk van de richting en de grootte van het veld en van het type halfgeleidermateriaal (n-type of p-type). Bij vermeerdering van het aantal vrije ladingdragers spreekt men van *verrijking* of *enhancement* en bij vermindering van *verarming* of *depletion*.

In een *n-type halfgeleidermateriaal* is in het vierwaardige silicium een dotering aangebracht met een *vijfwaardig element*, waardoor een van de elektronen van een atoom van het vijfwaardige element in het kristalrooster van het silicium minder sterk gebonden is. Door de thermische energie (Fermi niveau) kan zo'n elektron reeds bij kamertemperatuur losraken uit de atomaire binding, waardoor het kan deelnemen aan de geleiding. Deze *vrije elektronen* kunnen beïnvloed worden door het elektrisch veld over het halfgeleidermateriaal. In een *p-type halfgeleider* bestaat de dotering uit een *driewaardig element*. In het kristalrooster is er dan sprake van een 'tekort' aan elektronen in de valentieband. Zo'n tekort noemt men een *gat*. Binnen de kristalstructuur van een p-type gedoteerd silicium kan een gat zich verplaatsen, hetgeen effectief geleiding betekent van positieve ladingen. Ook deze worden door het elektrisch veld beïnvloed, zij het tegengesteld ten opzichte van elektronen.

Een gegeven elektrisch veld heeft daarom een *tegengesteld effect* op het geleidingsvermogen van het *n-type* en het *p-type* halfgeleidermateriaal. Als elektronen worden aangetrokken, dan worden gaten juist afgestoten, en andersom. Zie voor een uitgebreidere toelichting op geleidingsmechanismen in halfgeleiders [Wissenburgh, 1991; Müller, 1977].

De MOS transistor

De structuur van een *nMOS transistor* is aangegeven in figuur 5.4. De transistor wordt gevormd in een *p-type* ondergrond, het *substraat*. Hierin zijn twee eilandjes van zwaarder gedoteerd *n-type* halfgeleidermateriaal gemaakt. Deze eilandjes heten *drain* en *source*. Tussen deze eilandjes is nog geen geleiding mogelijk, omdat in het *n-type* materiaal elektronen en in het *p-type* materiaal gaten voor de geleiding zorgen. (Een MOS transistor geleidt in twee richtingen. Daarom mogen de drain en source aansluitingen verwisseld worden. Men noemt die aansluiting in een schema de source waar de richting van de elektronenstroom vanaf gaat.)



Figuur 5.4. De structuur van een *nMOS transistor*.

Op het substraat tussen de twee eilandjes wordt een dunne laag isolatiemateriaal aangebracht (silicium oxyde), met daar bovenop een geleidende laag. Deze noemt men de *gate*. De combinatie gate-isolator-substraat vormt een condensator. Een positieve spanning op de gate ten opzichte van het substraat trekt negatieve ladingen in het substraat aan. De positieve ladingen (gaten) worden daarentegen afgestoten en dringen dieper het substraat in. Bij een bepaalde spanning op de gate vormt zich eronder een dun laagje vrije elektronen in het aanvankelijk *p-type* substraat. Men noemt deze situatie *inversie*. Het substraat onder de gate krijgt onder invloed van het veld het karakter van *n-type* materiaal, met elektronen als meerderheidsladingdragers. Hierdoor is er tussen drain en source sprake van een aaneengesloten gebied van het *n-type*. Tussen de eilandjes is er nu een *geleidend kanaal*, waarin geleiding via elektronen plaatsvindt. De gate spanning waarbij sprake is van de vorming van een geleidend kanaal wordt de *drempelspanning* (*threshold voltage*) genoemd.

Een *pMOS transistor* wordt op een p-type substraat gevormd door in dat substraat eerst een gebied van het n-type te maken, met daarin twee eilandjes van het p-type. Bij een pMOS transistor moeten we voor enig inzicht in de werking het n-type halfgeleidermateriaal vervangen denken door p-type materiaal en p-type materiaal door n-type. Ook moet om de transistor in geleiding te brengen de spanning op de gate van polariteit wisselen. Het principe komt overeen met nMOS, zij het dat gaten en elektronen van rol wisselen. In pMOS zorgen gaten voor de geleiding in het kanaal.

Elektronen bewegen zich veel gemakkelijker in het halfgeleidermateriaal dan gaten. Als gevolg hiervan is het *geleidingsvermogen van nMOS transistoren* beduidend beter. Bij gelijke afmetingen schakelen nMOS transistoren sneller dan pMOS transistoren. Aanvankelijk heeft men daarom geprobeerd met uitsluitend nMOS een logische familie op te bouwen. De voordelen van CMOS, onder andere de lagere dissipatie in de statische toestand en de hogere dichtheid van poorten op het IC oppervlak, bleken echter zo groot te zijn dat nMOS vrijwel verdwenen is.

De *afmetingen van het kanaal* tussen drain en source zijn bepalend voor de elektrische eigenschappen van de MOS transistor. Zo moet, om een bepaalde stroom te kunnen schakelen, het kanaal een minimale breedte hebben. Daarnaast is de breedte/lengte verhouding W/L belangrijk voor de verhouding tussen de instelspanning op de gate en de drain-source stroom door de transistor. Hiermee worden onder meer de dynamische eigenschappen van de transistor als schakelaar bepaald.

Ook de *dikte van de isolator* tussen gate en substraat is belangrijk. Hoe dunner de laag is, des te groter is het veld bij een bepaalde instelspanning. Meestal is het laagje oxyde minder dan $0,1 \mu\text{m}$ dik. Een te dun laagje leidt tot veel uitval, vanwege kortsluiting of doorslag van het oxyde. Een soortgelijk probleem doet zich voor in het kanaal. Tijdens bedrijf komt regelmatig $V_{DD} - V_{SS}$ over het kanaal te staan. Door de steeds kleiner wordende afmetingen kan het veld in het kanaal te groot worden, met mogelijk doorslag. Dit is een van de motieven om de voedingsspanning te verlagen. Voor meer gegevens over deze technologie-aspecten zie [Bakoglu, 1990; Shoji, 1988; Weste, 1993].

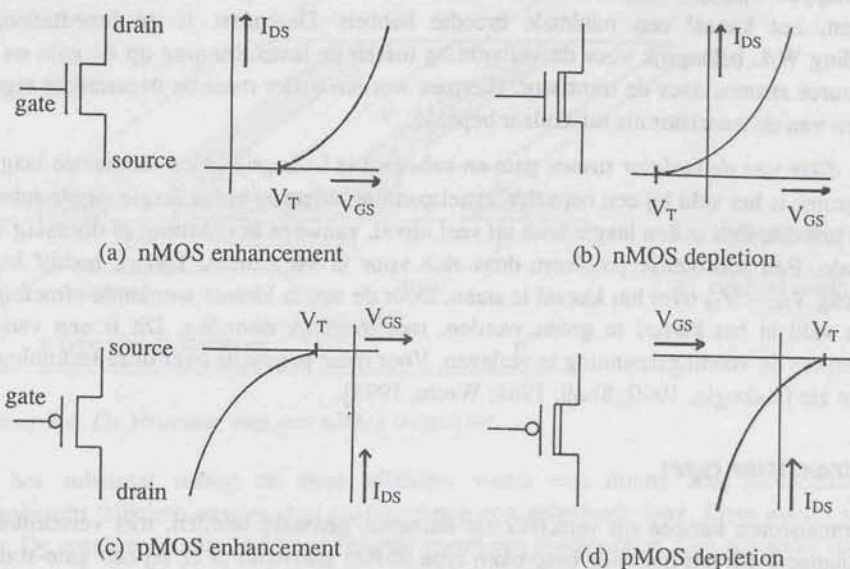
MOS transistor types

MOS transistoren kunnen op verschillende manieren gemaakt worden, met verschillende eigenschappen. Bij het hiervoor besproken type nMOS transistor is er bij een gate-source spanning van $V_{GS} = 0$ Volt geen geleidingsvermogen aanwezig. Het n-type kanaal (inversie) wordt pas gevormd bij een spanning $V_{GS} > V_T$, waarbij de drempelspanning V_T een *positieve waarde* heeft. We noemen dit type een *normally-OFF* of *enhancement transistor*. Door een geschikte dotering is het ook mogelijk dat zich reeds bij een gate-source spanning $V_{GS} = 0$ Volt een geleidend n-type kanaal gevormd heeft. Zo'n transistor kan slechts door een spanning $V_{GS} < V_T$ gesperd worden, waarbij V_T een *negatieve waarde* heeft. We spreken dan van een *normally-ON* of *depletion transistor*.

Enhancement en depletion types bestaan in principe zowel in nMOS als in pMOS. We onderscheiden dan ook vier types transistoren:

n-kanaal in p-type substraat	enhancement (normally-OFF)
	depletion (normally-ON)
en	
p-kanaal in n-type substraat	enhancement (normally-OFF)
	depletion (normally-ON)

Afhankelijk van het type moet er een positieve of een negatieve instelspanning op de gate gezet worden om de transistor in geleiding te brengen respectievelijk uit geleiding te halen. Figuur 5.5 geeft een overzicht van de spanning/stroom karakteristiek van de verschillende soorten MOS transistoren met de daarbij behorende tekensymbolen.

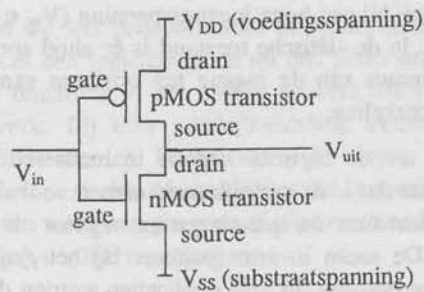


Figuur 5.5. Overzicht van types MOS transistoren met hun symbolen.

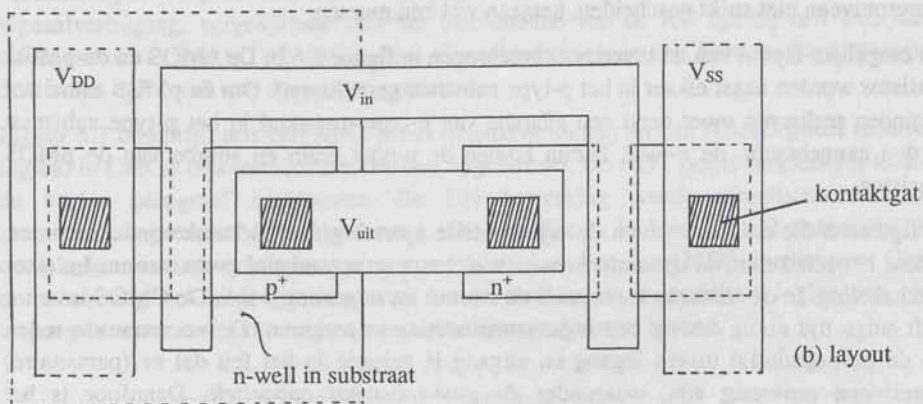
Soms wordt met wat uitgebreidere transistorsymbolen gewerkt, vooral daar waar men dieper op het fysische gedrag en de werking van de transistoren ingaat. Zo hebben MOS transistoren nog een vierde aansluitpunt waarop een spanning gezet kan worden, naast V_{DD} en V_{SS} . Voor een globaal inzicht in het gedrag van MOS transistoren in een *logische omgeving* kan deze spanning buiten beschouwing blijven. Tevens stellen we V_{SS} op 0 Volt. Voor 'digitale' toepassingen treft men daarom meestal de symbolen van figuur 5.5 aan. De *negatie-indicator*, in de vorm van het cirkeltje bij de gate aansluiting, duidt bij pMOS transistoren op het *complementair* zijn van de werking ten opzichte van nMOS

transistoren. (In feite wordt een logisch symbool toegepast om de fysische werking aan te duiden. Hiervoor is de polariteitsindicator bedoeld. Dit slordig gebruik van symbolen komt veel voor.)

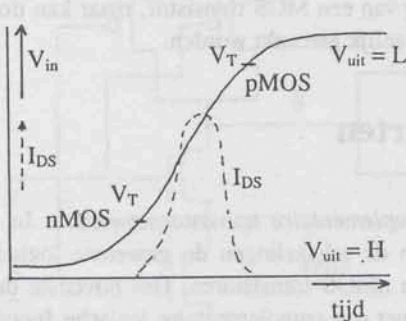
De CMOS inverter



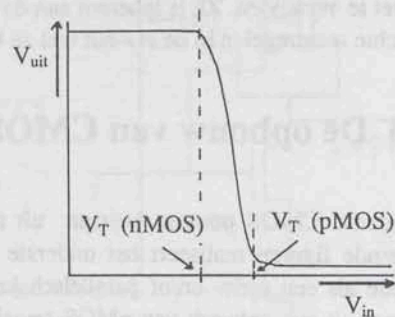
(a) schakeling



(b) layout



(c) schakelstroom



(d) transfer karakteristiek

Figuur 5.6. Realisatie van een CMOS inverter

Figuur 5.6 beschrijft een in CMOS uitgevoerde *invertorschakeling*. Het schema in figuur 5.6.a bestaat uit twee complementaire transistoren, een pMOS en een nMOS transistor. Wanneer deingangsspanning V_{in} in een van de twee gedefinieerde logische gebieden ligt, H of L, dan is er geen geleidend pad tussen V_{DD} en V_{SS} . Afhankelijk van de instelspanning geleidt de nMOS of de pMOS transistor. Bij een lageingangsspanning ($V_{in} \in L$) geleidt de pMOS transistor. Daardoor wordt de uitgangsspanning V_{uit} naar V_{DD} getrokken. V_{in} en V_{uit} liggen dan in verschillende logische niveaus. Ook bij een hogeingangsspanning ($V_{in} \in H$) keert de uitgangsspanning om. V_{uit} is dan laag. In de statische toestand is er altijd sprake van omkering of *inversie* van de spanningsniveaus van de ingang ten opzichte van de uitgang. Vandaar de naam *inverter* voor deze schakeling.

De schakeling in figuur 5.6 wordt toegepast om de *logische negatie* te realiseren (bij positieve zowel als bij negatieve logica). Vandaar dat in de praktijk twee namen voor deze schakeling in gebruik zijn, *inverter* als we kijken naar de spanningen en *negator* als we kijken naar de toegekende logische functie. De naam inverter behoort bij het *fysisch ontwerpniveau* en negator bij het *logisch ontwerpniveau*. In veel publicaties worden deze namen door elkaar heen gebruikt. Daarmee worden het fysische en het logische ontwerpniveau niet strikt gescheiden, hetgeen wel zou moeten.

Een mogelijke layout van de inverter is beschreven in figuur 5.6.b. De nMOS en de pMOS transistor worden naast elkaar in het p-type substraat gerealiseerd. Om de pMOS transistor te kunnen realiseren moet eerst een eilandje van n-type materiaal in het p-type substraat worden aangebracht, de *n-well*. Hierin komen de p-type drain en source van de pMOS transistor.

De figuren 5.6.c en 5.6.d geven de bijbehorende spanning/stroom karakteristieken weer. Tijdens het schakelen, de *dynamische toestand*, loopt er een relatief grote stroom I_{DS} door de schakeling. In de *statische toestand* is de stroom I_{DS} nagenoeg 0 μA . De CMOS inverter heeft enige tijd nodig om op een ingangsverandering te reageren. De voornaamste reden van de propagatietijd tussen ingang en uitgang is gelegen in het feit dat er (parasitaire) capaciteiten aanwezig zijn, waaronder de *gate-substraat capaciteit*. Daardoor is het opbouwen van de spanningen gebonden aan een zekere vertragingstijd. Deze gate capaciteit is niet te vermijden. Zij is inherent aan de werking van een MOS transistor, maar kan door gerichte maatregelen in de lay-out wel zo klein mogelijk gemaakt worden.

5.3 De opbouw van CMOS poorten

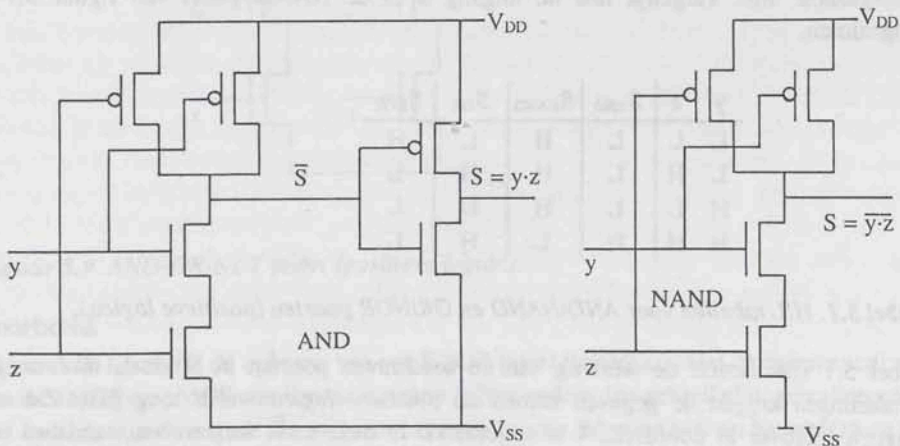
Standaard CMOS poorten bestaan uit twee *complementaire* transistornetwerken. In de volgende figuren realiseert het onderste deel van de schakelingen de gewenste logische functie als een serie- en/of parallelschakeling van nMOS transistoren. Het bovenste deel bestaat uit een netwerk van pMOS transistoren met een complementaire logische functie. Daarmee wordt bereikt dat er in de statische toestand geen geleidend pad tussen V_{DD} en V_{SS} bestaat. *In feite zijn hierbij extra transistoren toegepast om een vermindering van de dissipatie in de statische toestand te bereiken*

In de jaren 80 heeft men ook geïnvesteerd in de ontwikkeling van de *nMOS technologie* [Mavor, 1983]. Daarbij realiseerde het nMOS deel van een poort de gewenste logische functie en was het bovenste pMOS deel vervangen door een als weerstand geschakelde transistor. Vergelijk figuur 5.1. Deze schakelingen hadden als bezwaar dat er in de actieve of geleidende toestand van het logische nMOS netwerk een stroom door de poort ging. Onder andere vanwege de *lagere dissipatie* is men overgegaan op de CMOS technologie.

Gaan we wat gedetailleerder in op het schakelen van de transistoren, dan zien we dat in CMOS eeningangssignaal op een poort een even aantal gates moet instellen. Tegenover elke transistor in het nMOS netwerk staat een pMOS transistor in het complementaire netwerk. Bij elke schakelhandeling moeten dan twee capaciteiten C_{GS} opgeladen of ontladen worden. Hierbij wordt de meeste energie gedissipeerd. In het geval van nMOS behoeft er slechts één gate ingesteld te worden. Boven een bepaalde frequentie kan CMOS daarom meer energie gaan dissiperen dan nMOS. Deze zogenaamde *cross-over frequency* ligt gewoonlijk zeer hoog.

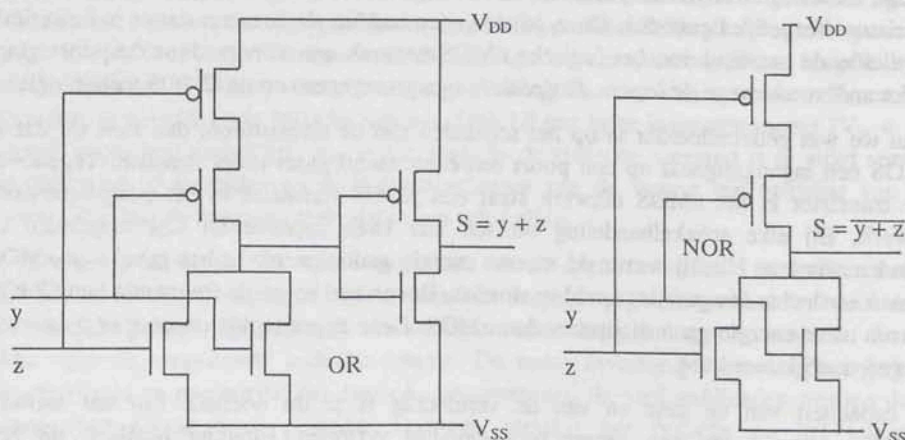
De capaciteit van de gate en van de verbinding is er de oorzaak van dat signaalveranderingen volgens een zekere propagatietijd verlopen. Effectief betekent dit een signaalvertraging, vergelijkbaar met het mechanisme van de RC tijd bij een weerstand-condensator netwerkje. We zullen deze signaalvertraging in een later stadium (paragraaf 5.9) in het tijdmodel van de poorten opnemen.

Figuur 5.7 beschrijft hoe een AND poort (positieve logica) en een NAND poort (positieve logica) in CMOS met transistoren worden opgebouwd. De NOT poort (negator) is reeds in de vorige paragraaf beschreven. De EN bewerking wordt gerealiseerd met een *serieschakeling* van nMOS transistoren. Dit principe is in hoofdstuk 2 toegelicht. Het complementaire netwerk bestaat uit een *parallelschakeling* van pMOS transistoren. Omdat de schakeling het uitgangsniveau van nature omkeert is in positieve logica een extra uitgangsinvertor nodig om tot het gewenste uitgangsniveau te komen.



Figuur 5.7. Opbouw van een AND en een NAND poort in CMOS (positieve logica).

Figuur 5.8 beschrijft de opbouw van een OR en een NOR poort. Na het voorafgaande spreekt deze figuur voor zich.



Figuur 5.8. Opbouw van een OR en een NOR poort in CMOS (positieve logica).

Uit figuur 5.7 en figuur 5.8 zien we dat het voordelen biedt om logische functies af te beelden op NAND en NOR poorten in plaats van op AND en OR poorten. De afbeeldingsprocedure voor NAND's en NOR's is reeds in hoofdstuk 4 besproken. De AND poort en de OR poort vergen bij toepassing van positieve logica altijd een extra negator om op het gewenste uitgangsniveau te komen.

Ten aanzien van de figuren 5.7, 5.8 en volgende geldt dat er bij kruisende lijnen geen galvanische verbinding op het kruispunt verondersteld wordt. Om een verbinding tussen vier en meer lijnen aan te geven tekent men de verbindingen als aparte aansluitingen op de 'doorgaande' lijn. Vergelijk hoe de uitgang S in de NAND poort van figuur 5.7 is aangesloten.

y	z	S_{AND}	S_{NAND}	S_{OR}	S_{NOR}
L	L	L	H	L	H
L	H	L	H	H	L
H	L	L	H	H	L
H	H	H	L	H	L

Tabel 5.1. H/L tabellen voor AND/NAND en OR/NOR poorten (positieve logica).

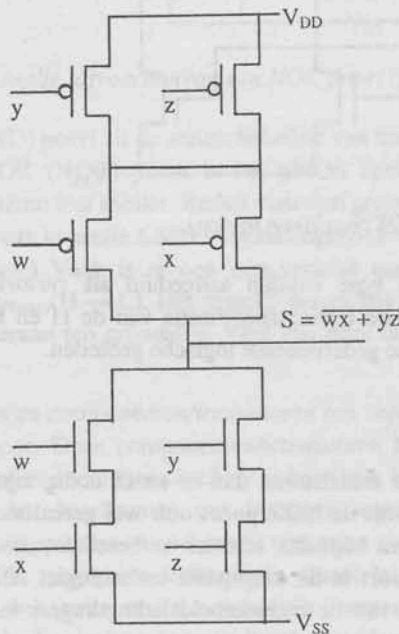
Tabel 5.1 specificeert de werking van de beschreven poorten in H en L niveaus. De schakelingen krijgen de gegeven namen als *positieve logica* wordt toegepast. Zie ook hetgeen hierover in hoofdstuk 4 is opgemerkt. In deze tekst werken we standaard met positieve logica, tenzij uitdrukkelijk anders is vermeld. Daarom laten we de vermelding van de logica afspraak meestal weg.

Complexe poorten

Meer complexe poorten worden volgens hetzelfde basisprincipe opgebouwd. In figuur 5.9 is de *AND-OR-NOT* poort beschreven. Deze poort heeft als logische functie

$$S = \overline{wx + yz} \quad (5.1)$$

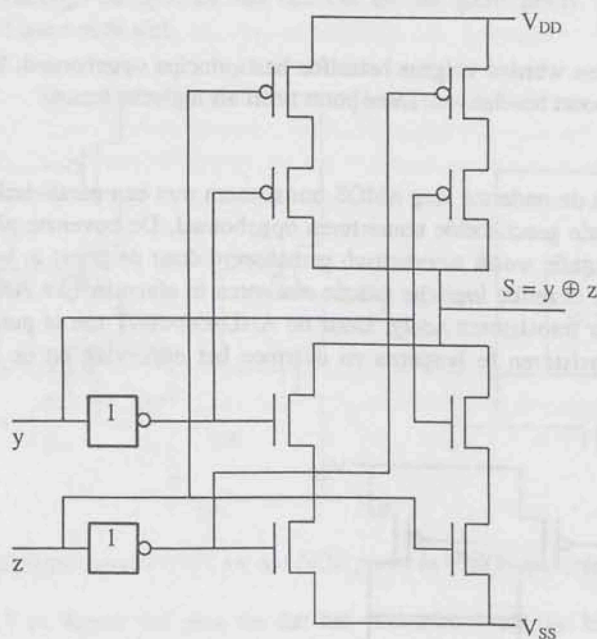
Deze functie wordt in de onderste laag nMOS transistoren met een parallelschakeling van twee takken van in serie geschakelde transistoren opgebouwd. De bovenste pMOS laag is complementair. De negatie wordt automatisch gerealiseerd door de poort in het midden af te takken. Zouden we dezelfde logische functie realiseren in afzonderlijke AND/OR/NOT poorten, dan zijn meer transistoren nodig. Door de AND-OR-NOT toe te passen is het in principe mogelijk transistoren te besparen en daarmee het oppervlak en de dissipatie te verminderen.



Figuur 5.9. AND-OR-NOT poort (positieve logica).

Voorbeeld.

Figuur 5.10 beschrijft de opbouw van een EXOR poort in CMOS. Het complementair zijn van de nMOS en pMOS transistornetwerken is hier anders dan gebruikelijk gerealiseerd en wel op basis van de negatie $\bar{S}$ van de te realiseren functie. Meestal hebben het nMOS en het pMOS deel een *duale structuur*, waarbij een serie/parallelschakeling van nMOS transistoren correspondeert met een parallel/serieschakeling van pMOS transistoren.

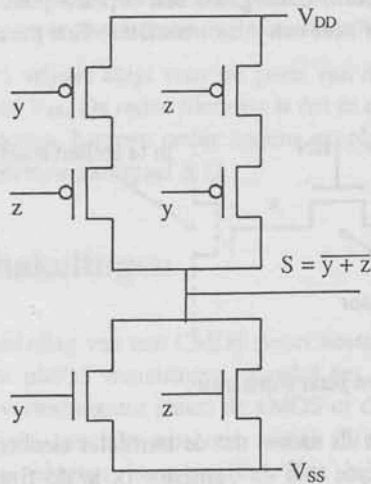


Figuur 5.10. EXOR poort in CMOS (positieve logica). □

Poorten van het hier besproken type worden aangeduid als *restoring logic*. Bij eeningangsspanning, die voldoet aan deingangsspecificatie van de H en L gebieden, ligt de uitgangsspanning automatisch in de gedefinieerde logische gebieden.

Extra voorzieningen

Soms ziet men in schema's meer transistoren dan er strikt nodig zijn op grond van de gewenste logische functie. Zo wordt de NOR poort ook wel gerealiseerd in de vorm van figuur 5.11. Dit wordt gedaan om bepaalde effecten te bereiken, die gebaseerd zijn op fysische eigenschappen van de poort in de toegepaste technologie. Allereerst is de reeds eerder genoemde beweeglijkheid van de meerderheidsladingdragers in pMOS beduidend minder dan in nMOS. Een opgebouwde vertraging door een serieschakeling van transistoren, die van nature in een serieschakeling altijd plaatsvindt [Shoji, 1988], is in het pMOS deel daarom hinderlijker dan in het nMOS deel. Daarnaast maakt het voor de (worst case) vertraging tussen de ingangsverandering en de bijbehorende uitgangsreactie nog uit op welke ingangstransistor in een seriële tak eeningangssignaal is aangesloten. Daarom is in figuur 5.11 de bovenste tak twee maal uitgevoerd en zijn de aansluitingen van de ingangssignalen in beide takken verwisseld. Dit voorbeeld geeft aan hoe met het inzetten van extra transistoren een snellere poort kan worden gemaakt. (Hetzelfde effect kan bereikt worden door de W/L verhouding van de transistoren aan te passen.) Een bijkomend voordeel is dat de minimale/maximale propagatietijd van ingang naar uitgang voor beide ingangssignalen hetzelfde is.



Figuur 5.11. Een snelle uitvoering van een NOR poort (positieve logica).

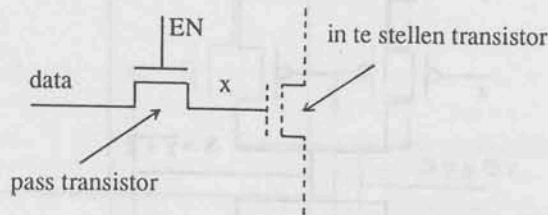
In een AND (NAND) poort zit de serieschakeling van transistoren in het nMOS deel van de poort en in een OR (NOR) poort in het pMOS deel. Van nature schakelt een AND (NAND) poort daarom iets sneller. Reden voor een geringe voorkeur voor de NAND poort boven de NOR poort in snelle CMOS schakelingen. (Een zelfde voorkeur geldt overigens ook bij TTL logica.) Vaak is er ook een verschil tussen de (maximale) propagatietijd $t_{P(max)}(L \rightarrow H)$ en $t_{P(max)}(H \rightarrow L)$. Het verschil tussen het geleidingsvermogen van nMOS en pMOS ligt ook hieraan ten grondslag. Veel van deze details worden besproken in [Shoji, 1988].

Het inzetten van extra componenten/transistoren om bepaalde effecten te bereiken komt in de praktijk veel voor. Deze componenten/transistoren hebben dan geen logische functie, maar dienen om een of ander fysisch effect te bereiken, bijvoorbeeld een versnelling van de werking of voor het beschermen van ingangen tegen een overspanning of statische elektriciteit. Het interpreteren van schema's met extra componenten is zonder enige kennis van de fysica en de toegepaste technologie uiterst lastig. Fabrikanten zouden voor de duidelijkheid dan ook twee soorten schema's moeten leveren, *logische schema's* en *uitvoeringsschema's*. Logische schema's dienen dan als toelichting op de logische werking en de uitvoeringsschema's laten de definitieve realisatie zien, inclusief de eventuele toevoegingen. In de IEC symbolen voor logische schakelingen wordt daarom onderscheid gemaakt tussen symbolen voor logische of ontwerpschema's en symbolen voor uitvoeringsschema's. Zie paragraaf 4.9.

5.4 De pass transistor en de transmission gate

In CMOS komt ook de in figuur 5.12 beschreven opstelling voor, waarbij een transistor wordt toegepast voor het al dan niet doorgeven van *datasignalen*. Met het enable signaal

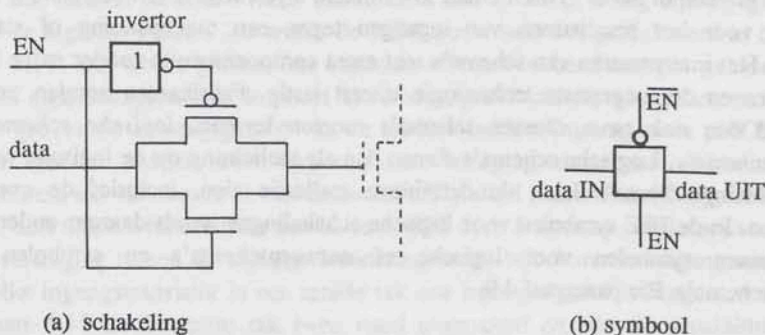
EN wordt ingesteld of een (extern) datasignaal een bepaald punt wel of niet beïnvloedt. Zo'n aldus toegepaste transistor heet een *pass transistor*. Een pass transistor is niet direct met V_{DD} of met V_{SS} verbonden.



Figuur 5.12. Toepassing van een pass transistor.

Een nMOS pass transistor heeft als nadeel dat de transistor slechts geleidt als de spanning tussen de gate (EN) en de source van de transistor (x in de figuur) boven de drempelspanning V_T blijft. Dit houdt in dat punt x slechts tot de spanning V_{EN} minus V_T kan worden opgeladen. Dit spanningsverschil heet de *threshold drop*. Daarboven geleidt de pass transistor niet meer. Een H niveau van data wordt als $H - V_T$ (slechter) doorgelaten. Daarentegen wordt bij $V_{EN} = H$ een L niveau van data goed doorgelaten. Het punt x ontlad zich daarbij tot het L niveau van het datasignaal. Zie [Weste, 1993, p.p. 86-90].

Een als pass transistor toegepaste pMOS transistor heeft de eigenschap dat een L niveau slecht wordt doorgelaten en een H niveau goed. De doorlaateigenschappen van een nMOS en een pMOS pass transistor zijn *complementair*. Het ligt daarom voor de hand de goede eigenschappen van beide types te combineren. De combinatie wordt aangeduid als *transmission gate*. Deze bestaat uit de parallelschakeling van een nMOS en een pMOS transistor. Zie figuur 5.13. In een transmission gate worden beide logische niveaus van de datasignalen goed doorgelaten.



Figuur 5.13. Toepassing van een transmission gate.

Transmission gates worden veel toegepast bij de realisatie van latch en flip-flop geheugenelementen. Zie de hoofdstukken 9 en 10. Ook bij het realiseren van een selector om data uit verschillende bronnen te kunnen selecteren worden pass transistoren en/of transmission

gates toegepast. We bespreken dit in hoofdstuk 6. In het algemeen schakelt restoring logic sneller, maar er zijn soms meer transistoren nodig om dezelfde functie te realiseren.

Transmission gates zitten vrijwel altijd voor de gates van de in te stellen transistoren. Zij schakelen niet naar V_{DD} of V_{SS} . De reden hiervoor is dat er dan in de statische toestand een grote stroom kan gaan lopen, hetgeen onder andere gevolgen heeft voor de testbaarheid van de schakeling. Zie hiervoor paragraaf 5.12.

5.5 Uitgangsschakelingen

De *normale uitgangsschakeling* van een CMOS poort bestaat uit de twee complementaire netwerken van nMOS en pMOS transistoren, waarbij het uitgangssignaal in het midden wordt afgetakt. Bij de invertor/negator stuurt de nMOS of de pMOS transistor de uitgang. Zo'n uitgangsschakeling is geschikt om een aantal (10 - 20) ingangen (gates van transistoren) van andere poorten in te stellen. Een bijkomend effect van dit type uitgangsschakeling is dat het de polariteit van het uitgangssignaal omkeert ten opzichte van het ingangssignaal. Soms is dit *omkeereffect* bedoeld (bij een invertor), soms is het ongewenst en moet het omkeren ongedaan gemaakt worden. Vandaar de eerder besproken AND/OR en NAND/NOR problematiek. (De standaarduitgang in TTL is de *totem pole* uitgang. Deze complementeert het uitgangsniveau ook.)

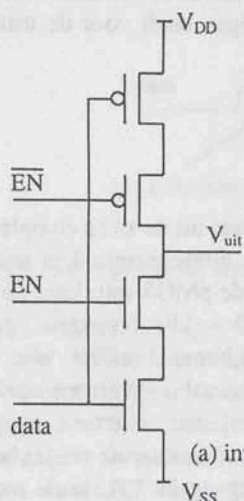
Aan de uitgangen van CMOS poorten worden vaak extra invertoren toegevoegd, die dienen als *uitgangsbuffer*. Zo'n buffer kan een grotere stroom leveren en verhoogt zo de *belastbaarheid* van de uitgang. Dit heeft als effect dat de uitgang veel sneller schakelt. Dit verhoogt de *schakelsnelheid* van een CMOS schakeling niet onaanzienlijk en bovendien vermindert het de *dissipatie* in de dynamische toestand. Uitgangsbuffers worden vooral toegepast op poorten in SSI/MSI IC's, bedoeld voor PCB montage. Intern in CMOS IC's worden buffers alleen toegepast als er door een uitgangssignaal een grote belasting moet worden ingesteld, bijvoorbeeld een groot aantal verschillende ingangen [Hedenstierna, 1987].

Normale uitgangsschakelingen zijn in staat een beperkt aantal ingangen in te stellen. We treffen deze constructies aan als één uitgang een vast aantal ingangen moet instellen. Het komt echter in veel schakelingen voor dat er aan de ingang van een module een keuze gemaakt moet worden uit data van verschillende bronnen. Daartoe kan een aparte selector (hoofdstuk 6) toegepast worden, die instelt welke data-ingang wordt doorgegeven. Met een aangepaste uitgangsconstructie kan ook een *selectiemechanisme* gerealiseerd worden. De volgende figuren laten uitvoeringen hiervan zien.

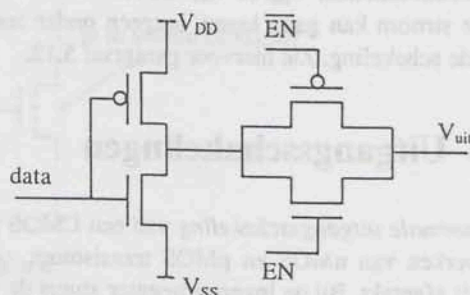
De 3-state uitgang

Figuur 5.14.a beschrijft een invertor/negator met een *3-state uitgang*. De schakeling heeft naast de data-ingang een *enable ingang* EN. Bij $EN = 1$ (positieve logica) verricht de invertor/negator zijn normale functie. Beide enable transistoren geleiden en de ingangsdata is in staat de uitgang in te stellen op het complementaire signaalniveau. Bij $EN = 0$ zijn de

enable transistoren gesperd. Dit heeft tot gevolg dat de uitgang van de schakeling niet meer met V_{DD} of met V_{SS} verbonden kan worden. De uitgang is afgeschakeld (zweeft) en is niet in staat een daaraan verbonden signaallijn actief in te stellen.



(a) inverter met 3-state uitgang

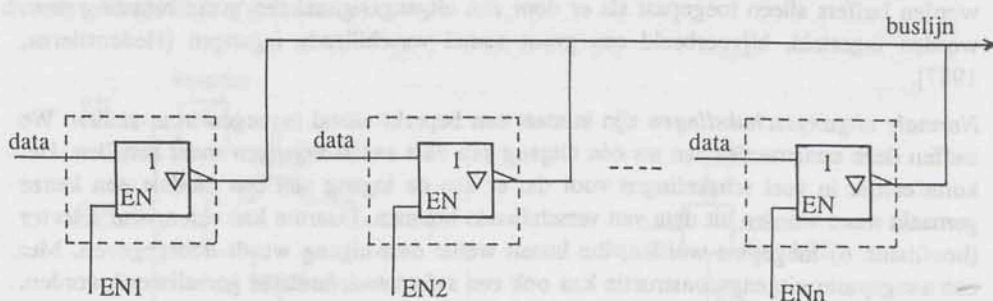


(b) inverter met 3-state uitgang



▽ aanduiding van 3-state uitgang

(c) symbool voor inverter met 3-state uitgang



(d) uitvoering 3-state buslijn

Figuur 5.14. Inverter/negator met 3-state uitgang.

Figuur 5.14.b beschrijft een alternatieve uitvoering, waarbij gebruik gemaakt wordt van een transmission gate aan de uitgang. Ook andere uitvoeringen van 3-state uitgangen komen voor. Zie CMOS databoeken.

Figuur 5.14.c beschrijft het IEC symbool voor 3-state uitgangen. Een driehoekje bij een uitgang geeft aan dat het een 3-state uitgang betreft. De bijbehorende enable ingang wordt met EN aangeduid, het is een *output enable*.

Figuur 5.14.d laat zien hoe 3-state uitgangen toegepast worden om een *selectiemechanisme* te implementeren. Met een enable signaal EN op elke invertor is instelbaar welke invertor data op de signaallijn mag zetten. Tijdens het verzenden van data moeten de uitgangen van de andere invertoren of poorten in 'disable' staan. De 3-state uitgang wordt veel toegepast op buslijnen. Een *buslijn* is een signaallijn, die door verschillende zenders wordt gebruikt om data over te dragen. Het is een gedeelde verbindinglijn (tijdmultiplexing).

Op een 3-state buslijn mag slechts één poort zenden. Wanneer twee of meer poorten tegelijk zouden zenden dan wordt, als ze de buslijn naar een tegengesteld niveau sturen, het niveau van de lijn *onbepaald*. Dit heeft twee gevolgen. Allereerst wordt de voedingsspanning min of meer kortgesloten via de uitgang van de ene poort, de buslijn en de uitgang van de andere poort. Daarnaast zien alle door de buslijn ingestelde ingangen een niet-gedefinieerd ingangsniveau, waardoor ook van de componenten achter deze ingangen zowel het nMOS als het pMOS deel in geleiding gestuurd kan worden. Als deze situatie lang duurt dan leidt de te hoge dissipatie tot het mogelijk defect raken van het IC.

Wanneer alle op een signaallijn aangesloten 3-state uitgangen afgeschakeld zijn zweeft de signaallijn. Het logische niveau is dan onbepaald. *Zwevende lijnen* pakken door inductie of door een capacitieve koppeling met andere signaallijnen gemakkelijk *storingen* op. Het wordt daarom aanbevolen zwevende lijnen te vermijden en altijd één poortuitgang op de buslijn actief te laten zijn. Het alternatief is het voorbelasten van de signaal- of buslijn met een weerstand, die een zwevende lijn naar een gedefinieerd logisch niveau trekt. Dit heeft echter gevolgen voor de maximale frequentie van de signalen op de buslijn, door de grotere belasting. Daarnaast is de dissipatie in de statische toestand hoger.

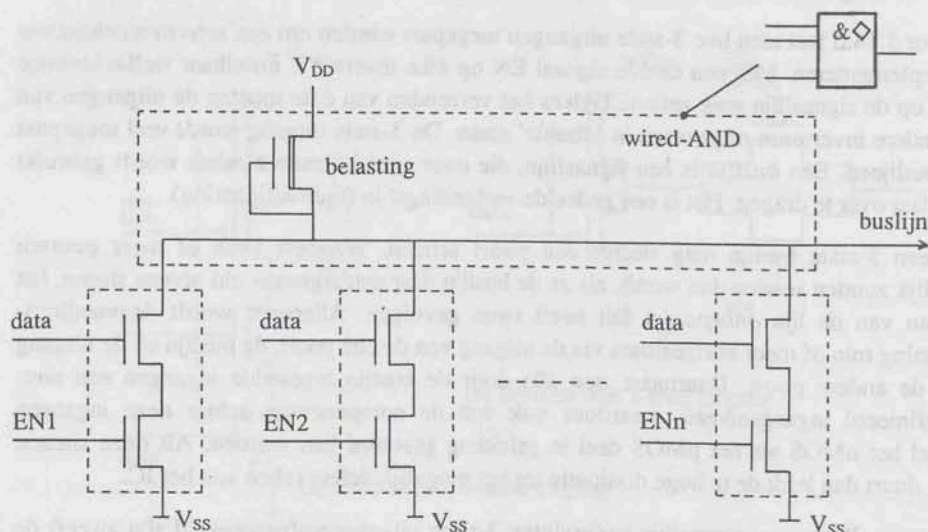
De open-drain (open-collector) uitgang

Een alternatief voor de 3-state uitgang is de *open-drain* uitgang (*open collector* bij TTL). Hierbij is het pMOS gedeelte van de uitgang of poort weggelaten. Om de uitgang toch naar een gedefinieerd H niveau te kunnen sturen wordt een gemeenschappelijke als weerstand geschakelde transistor toegevoegd. Zie figuur 5.15.

Deze uitgangsconstructie heeft de eigenschap dat, als meer dan één zender actief is op de signaallijn, de *laagste uitgang* wint. Zo'n constructie heet een *active-low wiring*. Logisch komt dit neer op een *wired-AND* constructie (positieve logica). In de I²L technologie berust de realisatie van de AND functie altijd op de *wired-AND*. In het transistorschema is er dan geen AND poort zichtbaar, maar wel in het logische ontwerpschema. Daarom wordt er op uitvoeringsschema's een apart symbool bij de uitgang van de aldus gerealiseerde *wired-AND* functie geplaatst. Zie het symbool in figuur 5.15.

Ook met deze constructie is een selectie te maken uit een aantal zenders/aanbieders van data. Tijdens zenden moeten de niet-geselecteerde zenders op het uitgangsniveau H

ingesteld worden. Tevens moet de belastingsweerstand van de bus aangepast worden aan het aantal op de bus aangesloten zenders.



Figuur 5.15. Open-drain uitgang.

In de praktijk komen zowel open-drain als 3-state uitgangen voor. Het voordeel van 3-state is dat er nooit een (relatief) grote stroom door de uitgang loopt. Daarentegen is open-drain wat robuuster bij instelconflicten. Ook de schakelsnelheden verschillen, in het voordeel van 3-state. Het hangt van de toepassing af welke uitgangsconstructie gekozen wordt. Zie ook [Gustavson, 1983]. Uitdrukkelijk wijzen we op het feit dat bij een open-drain constructie de stroom veel groter kan zijn dan de I_{DDQ} ruststroom in CMOS. Het doen van I_{DDQ} metingen om defecten op te sporen (paragraaf 5.12) wordt daardoor bemoeilijkt.

5.6 Overzicht van TTL logische families

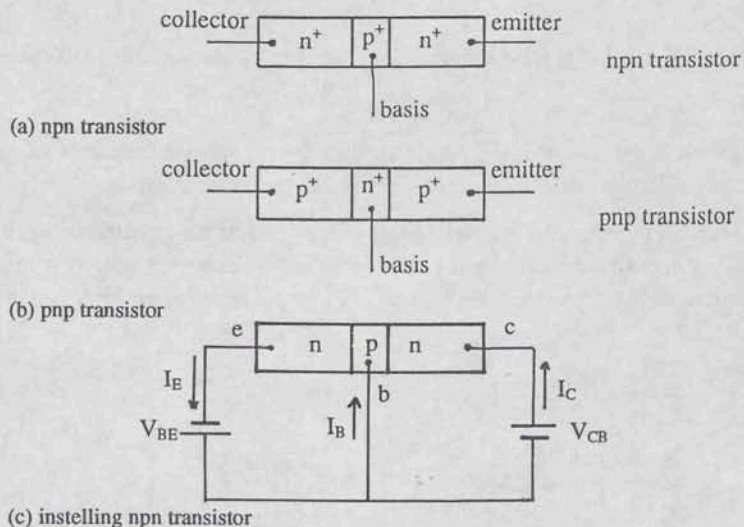
Voor ± 1965 werden elektronische logische schakelingen opgebouwd uit discrete diodes en transistoren, aangevuld met weerstanden en condensatoren voor de instelling. Bekende families uit die tijd waren de RTL (Resistor-Transistor-Logic) en de DTL (Diode-Transistor-Logic) families. De complexiteit van de modules was gering en ging het poortniveau nauwelijks te boven. Miniaturisatie van de afzonderlijke componenten en montage ervan op een klein PCB (Printed Circuit Board) leidde tot een hanteerbaar bouwdoosysteem. Sinds de opkomst van de bipolaire TTL familie in 1965 (Texas Instruments) zijn RTL en DTL verdwenen.

TTL was gedurende een aantal jaren de meest uitgebreide logische familie. Een grote diversiteit van functies was (en is gedeeltelijk nog) beschikbaar, variërend van poorten

(SSI, *small scale integrated circuits*), het registerniveau (MSI, *medium scale integrated circuits*) tot redelijk complexe logische schakelingen (LSI/VLSI, (*very*) *large scale integrated circuits*). Door de opkomst van de CMOS technologie en de daaraan gekoppelde voordelen is de belangstelling voor de bipolaire technologie afgenomen. We noemen slechts de lagere dissipatie en de hogere integratie-dichtheid van CMOS als redenen voor deze ontwikkeling. Daarnaast is er een ontwikkeling van IC's in de richting van een combinatie van bipolair en CMOS (BiCMOS). Vaak dienen de bipolaire stukjes om extra drivers te maken. We laten deze ontwikkeling hier onbesproken.

De bipolaire transistor

Er bestaan twee basisvormen van *bipolaire transistoren*, de npn en de pnp transistor. De npn transistor is opgebouwd uit een dunne laag p-type halfgeleidermateriaal tussen twee lagen n-type halfgeleidermateriaal. Deze dunne laag noemt men de *basis*. Bij de pnp transistor is het precies andersom, dan is de basis van het n-type halfgeleidermateriaal. Zie de figuur 5.16.a/b.



Figuur 5.16. Opbouw bipolaire transistoren.

Figuur 5.15.c geeft de gebruikelijke instelling van een npn transistor met twee (gelijk)spanningsbronnen V_{BE} en V_{CB} . De collector-basisovergang (collectorjunctie) is bij de aangebrachte spanning V_{CB} gesperd. Dit is te vergelijken met een gesperde diode. Er loopt daarbij slechts een zeer geringe lekstroom door de collectoraansluiting.

Brengt men ook de aangegeven spanning V_{BE} aan, dan is de emitter-basisovergang (emitterjunctie) te beschouwen als een in doorlaat geschakelde diode. Via de emitterjunctie worden dan elektronen in de basis geïnjecteerd. Deze combineren met de in de basis aanwezige gaten, maar als de basis dun genoeg is zal gemiddeld een bepaald percentage van

de in de basis geïnjecteerde elektronen dit recombinatieproces overleven en zullen er elektronen in de buurt van de collectorjunctie komen. Daar worden zij door het elektrisch veld van de collectorjunctie weggevangen. Uitwendig gaat er dan een collectorstroom I_C lopen. Op deze wijze kan de stroom door de collector via de basisspanning V_{BE} ingesteld of onderdrukt worden.

Samengevat: Een positieve basisspanning V_{BE} (in de praktijk moet deze boven een bepaalde *drempelspanning* uitkomen) geeft aanleiding tot een stroom I_C door de transistor. Een negatieve basisspanning daarentegen spert de emitterjunctie, waardoor ook de transistor gesperd raakt. Vanwege deze eigenschappen is de transistor als elektronische schakelaar te gebruiken. Bij dit geleidingsmechanisme spelen gaten een soortgelijke rol. Omdat zowel elektronen als gaten bijdragen tot de stroomgeleiding noemt men deze transistoren bipolaire transistoren. De werking van pnp transistoren is complementair ten opzichte van npn transistoren. We bespreken ze niet apart [Wissenburgh, 1995].

Bij een npn bipolaire transistor heeft een basisspanning V_{BE} boven de drempelspanning een collectorstroom I_C tot gevolg, welke in een groot gebied vrijwel *lineair evenredig* is met de ingestelde basisstroom I_B ,

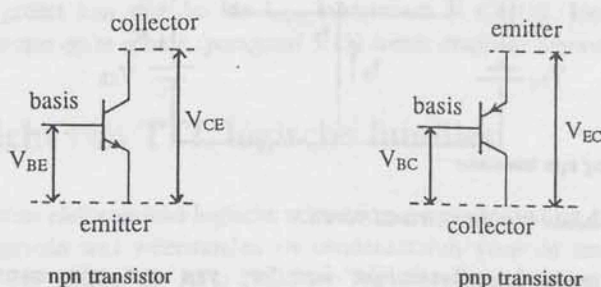
$$I_C = B_F \cdot I_B \quad (5.2)$$

waarbij de evenredigheidsfactor (*stroomversterkingsfactor*) B_F een waarde heeft tussen ca

$$100 < B_F < 500 \quad (5.3)$$

Boven een bepaalde waarde van I_B neemt de stroom I_C niet meer evenredig toe en is de transistor *verzadigd*. In bipolaire transistoren is er altijd een basisstroom I_B .

In *gesperde toestand*, bij een basis-emitterspanning lager dan de drempelspanning, loopt er een zekere *lekstroom* door de transistor. Deze is veel kleiner dan de stroom in voorwaartse richting en wordt in de modellen voor het logische ontwerpniveau meestal verwaarloosd.



Figuur 5.17. Symbolen voor bipolaire transistoren.

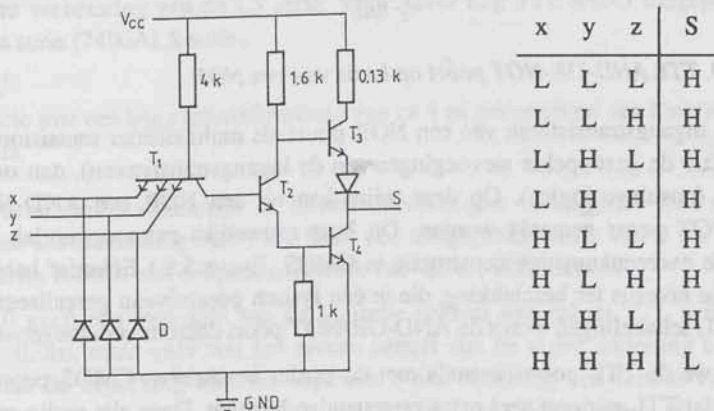
Figuur 5.17 geeft de *tekensymbolen* voor npn en pnp bipolaire transistoren. In de digitale techniek worden transistoren slechts in twee toestanden toegepast, open/in verzadiging gestuurd of gesperd. Vandaar dat we ze globaal kunnen modelleren met een 'ideale schakelaar' en twee weerstanden. De lage weerstand geldt voor de doorlaatrichting en de hoge weerstand voor de lekstroom, welke er loopt wanneer de transistor gesperd is. We

merken uitdrukkelijk op dat deze modellen slechts gelden voor de *statische toestand*. Tijdens de overgang van de ene toestand naar de andere toestand is de logische werking derhalve niet gedefinieerd. Op lagere ontwerpniveaus worden uitgebreidere transistor-modellen toegepast, waarmee ook de dynamische eigenschappen zoals de schakelsnelheid kunnen worden bestudeerd.

Poorten in de TTL technologie

Het basiselement in de bipolaire TTL technologie is de NAND poort. Figuur 5.18 geeft het schema en de H/L tabel ervan. Deze schakeling realiseert in positieve logica de NIET-EN bewerking. De schakeling werkt als volgt. De ingangstransistor T_1 is een *multi-emitter transistor*. Hiermee wordt de logische bewerking EN gerealiseerd, via een *active-low wiring* van de emitters van de ingangstransistor. Na de ingangstransistor T_1 komt de uitgangstrap. Deze heeft als functie het verkregen logische signaal te *versterken* en een uitgangssignaal S te produceren met een tot H of tot L behorend uitgangsniveau. De transistor T_2 stuurt, afhankelijk van de instelling van zijn basis door transistor T_1 , transistor T_3 of transistor T_4 in verzadiging. Geleidt transistor T_3 , dan is de uitgangsspanning S op het logische niveau H. Geleidt T_4 , dan is het uitgangsniveau L. De functie van transistor T_2 wordt wel aangeduid als *phase-splitter*. De gehele uitgangstrap wordt *totem pole uitgang* genoemd. Andere uitgangstrappen, die het equivalent van de 3-state en open-drain uitgang realiseren, zijn ook beschikbaar.

De totem pole keert de uitgangsspanning om. Om een AND poort te realiseren moet er een extra trap worden toegevoegd, die deze omkering ongedaan maakt. Ook hier zien we een uitgesproken voorkeur voor de NAND poort boven de AND poort. Hetzelfde geldt in de TTL technologie voor de NOR ten opzichte van de OR. Zie ook paragraaf 4.4.

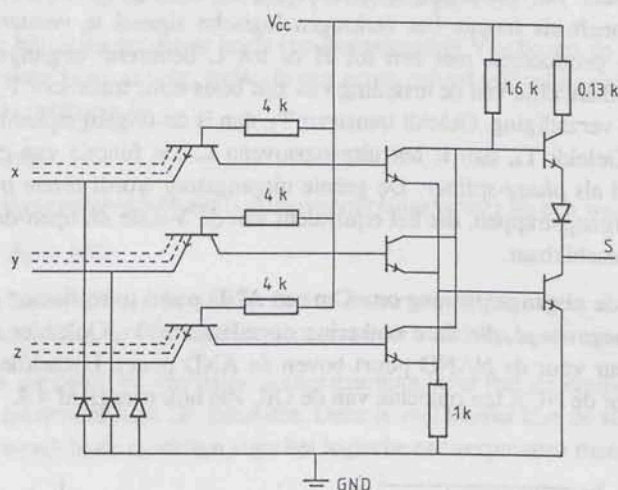


Figuur 5.18. NAND poort in TTL technologie.

De diodes D aan de ingangen in figuur 5.18 hebben als functie de ingangen te beschermen tegen negatieve spanningspieken. Deze treden onder andere op als gevolg van reflecties op

signaallijnen. Een te ver negatief gaandeingangsspanning kan het IC vernietigen. Een diode verhindert dat eeningangsspanning te ver naar beneden getrokken wordt. Wederom een voorbeeld van componenten die geen logische functie hebben, maar wel essentieel zijn voor de goede werking van het IC. Overigens is ook bij CMOS IC's een dergelijke ingangsportectie noodzakelijk.

Figuur 5.19 toont hoe het schema van een TTL NOR kan worden uitgebreid naar een TTL AND-OR-NOT constructie. Drie paren van twee transistoren realiseren de 3-input OR functie. De uitgangstrap versterkt het signaal verder. Een bijkomend effect is dat de uitgangsspanning wordt geïnverteerd. Een vergelijking met figuur 5.18 leert dat de NAND poort eenvoudiger is dan de NOR poort. In TTL is er daarom een sterke voorkeur voor de NAND als 'enige' poort om grotere schakelingen te maken.



Figuur 5.19. TTL AND-OR-NOT poort op basis van een NOR.

Wanneer de ingangstransistoren van een NOR poort als multi-emitter transistoren worden uitgevoerd (zie de gestippelde toevoegingen aan de ingangstransistoren), dan ontstaat een wired-AND (positieve logica). Op deze wijze kan uit een NOR een AND-NOR ofwel AND-OR-NOT poort gemaakt worden. Dit kost nauwelijks extra oppervlak in het IC. (Vergelijk de overeenkomstige constructie in CMOS, figuur 5.9.) Effectief hebben we nu twee logische niveaus ter beschikking, die in één fysisch poortniveau gerealiseerd worden. In snelle TTL schakelingen wordt de AND-OR-NOT poort daarom veel toegepast.

Vergelijken we de TTL poortschema's met de eerder beschreven CMOS poortschema's, dan valt op dat TTL poorten veel extra weerstanden bezitten. Deze zijn nodig om de juiste spanningsniveaus in te stellen. CMOS heeft deze weerstanden niet nodig. Nu zijn weerstanden relatief grote componenten op het IC oppervlak. Vandaar dat in CMOS de dichtheid van poorten groter is dan in TTL. Een van de redenen voor de overgang op CMOS.

TTL logische families

De gegeven weerstandswaarden in de vorige schema's corresponderen met die van standaard TTL, de N serie. Hogere weerstandswaarden geven kleinere stromen, een lagere dissipatie en een langzamere schakeling. Lagere waarden resulteren in hogere stromen en een snellere schakeling. Op deze wijze kan de dissipatie en/of de snelheid beïnvloed worden. Zo zijn in de loop der jaren verschillende series TTL schakelingen ontstaan, geoptimaliseerd naar snelheid, dissipatie of andere criteria.

Standaard TTL.

Propagatietijd ca. 10 ns per poort. Deze serie was de eerste en wordt aangeduid als de standaard TTL serie (7400N serie).

Low-power TTL.

De weerstandswaarden zijn hoger gekozen. De dissipatie is lager, de propagatietijd neemt toe (7400L serie).

High-speed TTL.

Lagere weerstandswaarden. De dissipatie is wat groter. Deze serie is minder gevoelig voor ruis op de logische niveaus (7400H serie).

Schottky-clamped TTL.

Schottky diodes verhinderen dat de transistoren te ver in verzadiging komen. Hierdoor zijn hogere schakelsnelheden mogelijk (7400S serie).

Low-power Schottky TTL.

Een combinatie van de L en de S serie. Heeft snel na de introductie ervan de N serie verdrongen (7400LS serie).

Advanced low-power Schottky TTL.

Een verdere verbetering van de LS serie. Voor zover nog TTL wordt toegepast is het de aanbevolen serie (7400ALS serie).

Fast TTL.

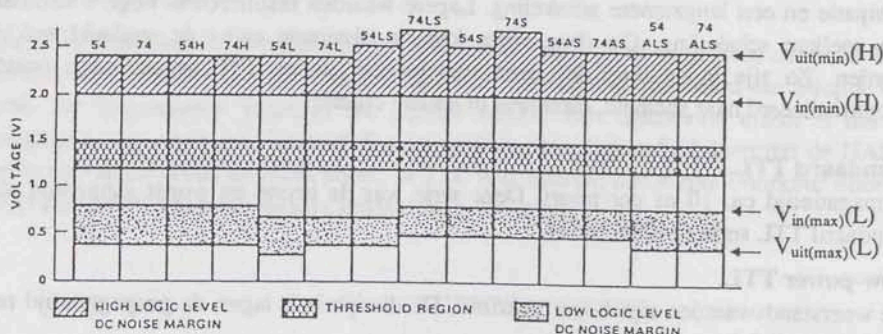
Recente serie met een hoge schakelsnelheid van ca 3 ns per poort en een klokfrequentie van ca 100 MHz.

De beschreven series waren/zijn in diverse uitvoeringen verkrijgbaar met verschillende specificaties. De aanduiding met 74xx heeft een temperatuurbereik van 0 tot 70° Celsius. De 54xx series hebben een temperatuurbereik van -55 tot 125° Celsius.

Figuur 5.20 geeft een overzicht van de logische niveaus van enkele TTL series. We zien kleine verschillen, maar voor wat het niveau betreft zijn de series onderling uitwisselbaar. Daarbij geldt de beperking dat, als bepaalde types onderling een fanout van 10 of 20 hebben, dit ten aanzien van aansturing van poorten uit andere series niet geldt. Soms kunnen er dan meer aangestuurd worden en omgekeerd minder, of andersom.

Bij de oudere TTL series ligt de ontwikkeling stil en is de productie vrijwel gestopt. De meer recente CMOS technologie krijgt alle aandacht en heeft TTL verdrongen. Hetzelfde geldt in meer of mindere mate voor de andere bipolaire series als I²L [Hart, 1972] en ECL.

In de rest van dit hoofdstuk beperken we ons tot CMOS. Recente ontwikkelingen duiden in een richting, waarbij bipolair en CMOS op de wafer gecombineerd worden (BiCMOS). Dan kunnen de voordelen van beide benut worden.



Figuur 5.20. Overzicht logische niveaus van de TTL familie (bron: Texas Instruments).

5.7 Overzicht van CMOS logische families

In CMOS heeft zich een met TTL vergelijkbare ontwikkeling voorgedaan. Aanvankelijk was de lage schakelsnelheid van CMOS ten opzichte van TTL een groot bezwaar. CMOS bleef steken op 15 tot 40 ns vertraging per poort, vergeleken met 1.5 tot 10 ns per poort in de verschillende TTL families. De ontwikkeling van de CMOS technologie heeft dit verschil echter weggewerkt. Dank zij een betere processing, kleinere afmetingen en de ontwikkeling van de technologie zijn ook in submicron CMOS hoge schakelsnelheden mogelijk.

Figuur 5.21 geeft een overzicht van recent ontwikkelde logische families, ingedeeld naar voedingsspanning en schakelsnelheid (bron: Philips Semiconductors). We geven hieronder een summier overzicht van de kenmerken van de verschillende CMOS logische families. Voor meer informatie zie [Philips, 1986; Philips, 1995] en de databoeken van elke serie afzonderlijk.

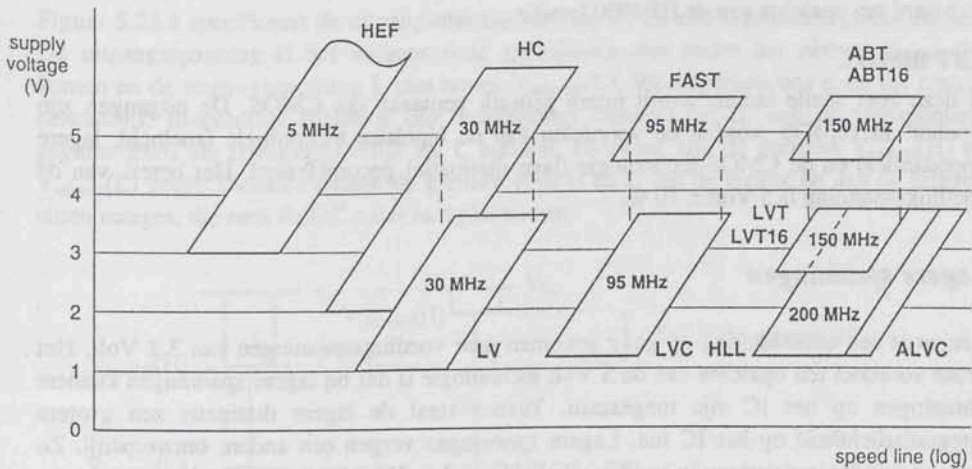
CMOS 4000.

Een oudere familie CMOS componenten. Vergeleken met LS-TTL kenmerkt deze serie zich door lagere schakelsnelheden. De propagatietijd hangt bovendien sterk af van de capacitieve belasting van de poortuitgang. De serie werkt bij een voedingsspanning van 3 tot 15 Volt, hetgeen een zeer ruim bereik is.

HE4000B.

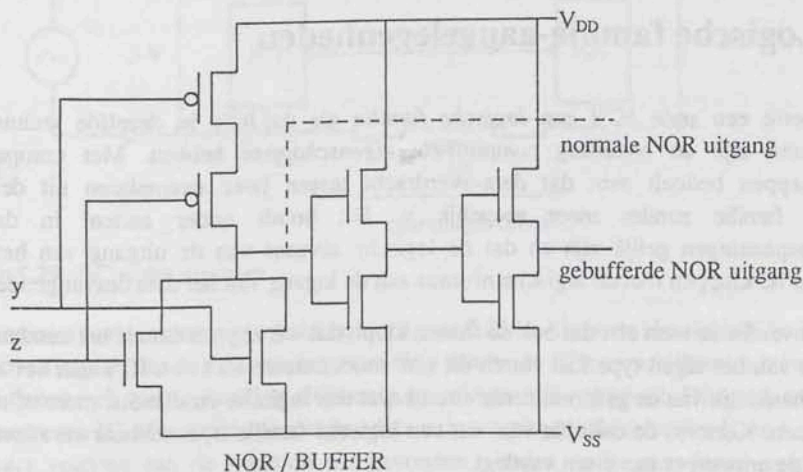
Een sterk verbeterde uitvoering van de 4000 familie is de HE4000B familie, waarin gebruik gemaakt wordt van de LOCMOS technologie (Local Oxidation CMOS). De toevoeging van extra uitgangsbuffers zorgt ervoor dat de uitgangen een hogere capacitieve belasting

kunnen hebben. Dit resulteert in snellere signaaltransities en mede daardoor een lagere dissipatie in de dynamische toestand van de schakeling. De grote afstand tussen de H en L niveaus zorgt tevens voor uitstekende *ruismarges*, hetgeen de robuustheid van schakelingen bevordert. (Zie ook paragraaf 5.8.)



Figuur 5.21. Recente logische families.

Figuur 5.22 toont het schema van een NOR poort met een gebufferde uitgang. Wederom een voorbeeld van het uitrusten van transistoren/oppervlak tegen betere prestaties. (Bij poorten met uitgangsbuffers gaat de voorkeur weer uit naar AND's en OR's boven NAND's en NOR's.)



Figuur 5.22. NOR poort met gebufferde uitgangen.

HCMOS.

Een met TTL compatibele familie, waarin series beschikbaar zijn met uitgangsniveaus die vergelijkbaar zijn met TTL uitgangsniveaus. HCT types kunnen worden toegepast daar waar een interface tussen TTL IC's en CMOS IC's nodig is. De fanout is dan vergelijkbaar met LS-TTL. Alle types hebben gebufferde uitgangen. De schakelsnelheden zijn belangrijk verbeterd ten opzichte van de HE4000 familie.

ABT BiMOS.

In deze zeer snelle familie wordt intern gebruik gemaakt van CMOS. De uitgangen zijn bipolair. In BiMOS worden de voordelen van de bipolaire technologie (snelheid, lagere impedanties) en de CMOS technologie (lage dissipatie) gecombineerd. Het bereik van de voedingsspanning is 5 Volt $\pm$ 10 %.

Lagere spanningen

Recent is een ontwikkeling op gang gekomen naar voedingsspanningen van 3.3 Volt. Het grote voordeel ten opzichte van de 5 Volt technologie is dat bij lagere spanningen kleinere afmetingen op het IC zijn toegestaan. Tevens staat de lagere dissipatie een grotere integratiedichtheid op het IC toe. Lagere spanningen vergen een andere ontwerpstyl. Zo kunnen minder transistoren in serie geschakeld worden. Het voert te ver om hier nu op in te gaan.

Een vijftal series is beschikbaar, LV-HCMOS (Low Voltage HCMOS), LVC (Low Voltage CMOS, compatibel met de TTL Fast logica), HLL (High speed, Low voltage, Low power), ALVC (Advanced LVC, momenteel de snelste familie met een schakelsnelheid tot 200 MHz) en LVT/LVT16 (een snelle uitvoering van de ABT/ABT16 familie).

5.8 Logische familie-aangelegenheden

Men noemt een serie IC's een *logische familie* als de IC's in dezelfde technologie gerealiseerd zijn en onderling *compatibele eigenschappen* hebben. Met compatibele eigenschappen bedoelt men dat data-overdracht tussen twee exemplaren uit dezelfde logische familie zonder meer mogelijk is. Dit houdt onder andere in dat de voedingsspanningen gelijk zijn en dat de logische niveaus van de uitgang van het data zendende IC kloppen met de logische niveaus aan de ingang van het data ontvangende IC.

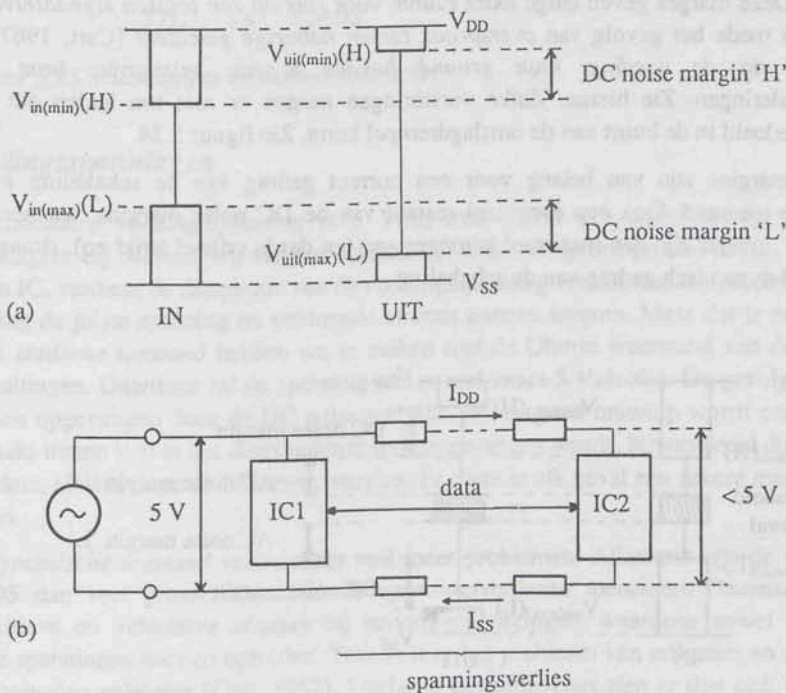
Een stap verder en men eist dat ook de fanout klopt, dat wil zeggen dat als het zendende IC tien IC's van het eigen type kan sturen dit ook moet kunnen met tien IC's van het andere type. Afhankelijk van de geformuleerde eisen bevat een logische familie dus meer of minder exemplaren. Kortom, de definitie van wat een logische familie is, is rekbaar en afhankelijk van wat de ontwerper aan eisen vastlegt.

Ons standpunt is dat een verantwoorde interpretatie van het begrip '*compatibel*' betekent dat de IC's *niet stuk gaan* als zij op elkaar aangesloten worden. Elke uitbreiding van deze

betekenis heeft zonder nader onderzoek een speculatief element in zich. We zullen deze uitspraak toelichten aan de hand van een beschouwing over *ruismarges*.

Ruismarges

Figuur 5.23.a specificeert de uitgangsniveaus van een IC uit een bepaalde logische familie. De uitgangsspanning H zal volgens deze specificatie niet onder het niveau $V_{uit(min)}(H)$ komen en de uitgangsspanning L niet boven $V_{uit(max)}(L)$. We beperken ons even tot *typical conditions*. In dezelfde figuur is ook aangegeven wanneer dit IC eeningangsspanning gegarandeerd als H respectievelijk als L herkent. Hiervoor zijn de grenzen $V_{in(min)}(H)$ en $V_{in(max)}(L)$ gespecificeerd. Tussen de grenzen voor H en L aan de ingang en aan de uitgang zitten marges, die men de *DC noise margins* noemt.



Figuur 5.23. DC noise margins.

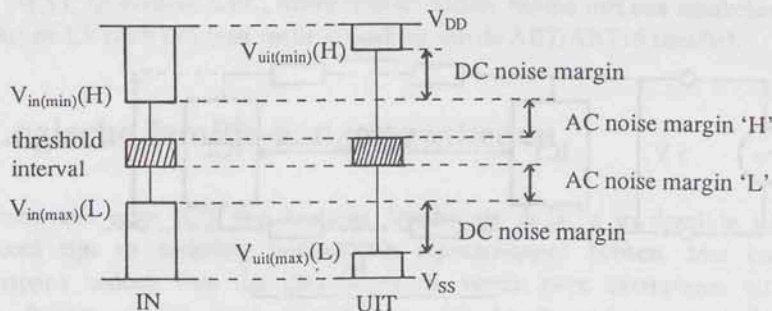
Het waarom van deze marges is geschetst in figuur 5.23.b. Als twee identieke IC's data aan elkaar overdragen, dan kan de voedingsspanning van beide IC's verschillen omdat er in het distributienetwerk van de voeding altijd enig spanningsverlies optreedt. Dit geldt trouwens ook binnen een IC. Genoemd spanningsverlies veroorzaakt dat de feitelijke H en L niveaus niet exact voldoen aan de waarden die gevonden worden onder *standaard omgevingscondities*, de *typical conditions*. Dit verschil heeft niet alleen betrekking op de statische toestand van de schakeling, maar ook en vooral op de *dynamische toestand*. Tijdens het schakelen zijn de spanningsverliezen in de voeding veel groter. Het is van het grootste

belang dat de logische spanningsniveaus dan blijven voldoen aan deingangsspecificaties. Om dit te kunnen garanderen moeten de DC noise margins voldoende groot zijn.

Het is duidelijk dat, als twee IC's data aan elkaar overdragen, de IC's compatibel moeten zijn. Het begrip '*compatibel*' vereist dus dat er voldoende grote DC noise margins zijn. Anders kan de kans op fouten toenemen als gevolg van niveauverschuivingen, ruis en overspraak op de signaalniveaus. Anderzijds, als een signaal tot een tijdstip t nodig is om een bepaalde component in te stellen, dan moet dit gelden voor de logische waarde van het signaal *inclusief zekere ruismarges!* In de praktijk wordt dit laatste heel vaak vergeten. Vandaar onze eerdere uitspraak over de interpretatie van het begrip compatibel.

Naast de DC noise margins hanteert men ook het begrip *AC noise margins*. Dit is het spanningsverschil tussen de omslagdrempel (beter: het gebied waarin de omslagdrempel van een ingang onder alle mogelijke condities ligt) en de spanningsniveaus $V_{in(min)}(H)$ en $V_{in(max)}(L)$. Deze marges geven enige extra ruimte voor ruis op een logisch signaalniveau. Deze ruis is mede het gevolg van overspraak tussen naburige geleiders [Catt, 1967] of overspraak via de voeding. Ook ground bounce is een belangrijke bron van niveauveranderingen. Zie hierna. Zulke verstoringen mogen er niet toe leiden dat een signaal onbedoeld in de buurt van de omslagdrempel komt. Zie figuur 5.24.

AC noise margins zijn van belang voor een correct gedrag van de schakeling in de dynamische toestand. Ook een eventueel restant van de DC noise margins, wanneer de 'level shifts' minder zijn dan maximaal is toegestaan (en dat is vrijwel altijd zo), draagt bij tot een goed dynamisch gedrag van de schakeling.



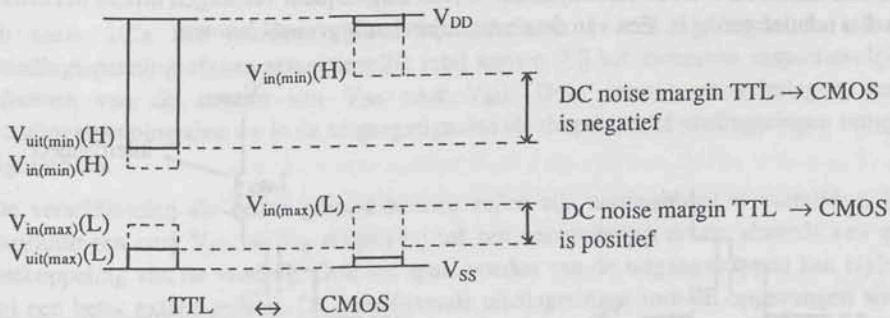
Figuur 5.24. AC noise margins.

Ruismarges zijn essentieel voor een *robuuste schakeling*. Een gebrek aan robuustheid is in de praktijk heel lastig op te sporen, omdat de *foutkans* uiterst klein kan zijn. Door metingen is dit vrijwel niet te constateren. Zie ook [Beenker, 1995; Hill, 1986; Ott, 1976].

Voorbeeld

Figuur 5.25 geeft een voorbeeld van de DC noise margins bij data-overdracht van een TTL naar een CMOS IC. We zien dat data-overdracht in de ene richting mogelijk is en in de

andere richting niet. Bij het uittesten blijkt vaak dat er in principe data-overdracht mogelijk is [Funk, 1986]. Alleen wordt dan niet aan hoge eisen van robuustheid en/of van reproduceerbaarheid voldaan.



Figuur 5.25. Ruismarges en data-overdracht.

Voedingsspanningen

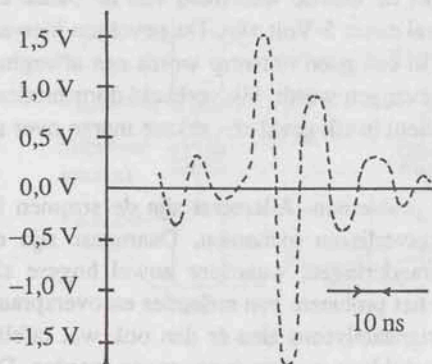
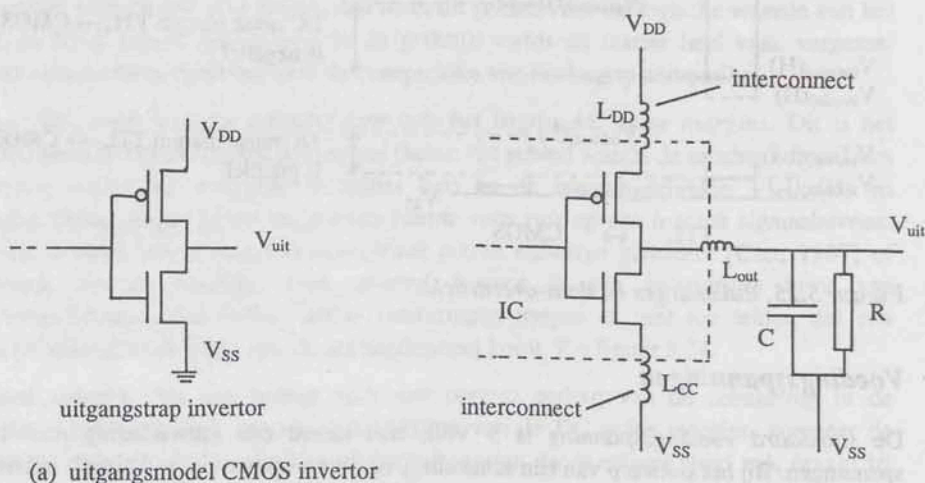
De *standaard voedingsspanning* is 5 Volt, met recent een ontwikkeling naar lagere spanningen. Bij het ontwerp van een schakeling, of dit nu op een printed circuit board is of in een IC, verdient de distributie van de voedingsspanning veel aandacht. Allereerst moet de voeding de juiste spanning en voldoende stroom kunnen leveren. Maar dat is niet genoeg. In de *statische toestand* hebben we te maken met de Ohmse weerstand van de banen en aansluitingen. Daardoor zal de spanning niet overal exact 5 Volt zijn. De gevolgen hiervan worden opgevangen door de DC noise margins. In een goed ontwerp wordt een afweging gemaakt tussen wat in het distributienetwerk opgevangen wordt, bijvoorbeeld door bredere geleiders, en wat door de DC noise margins. Er dient in elk geval een zekere marge over te blijven.

De *dynamische toestand* veroorzaakt veel meer problemen. Allereerst zijn de stromen in CMOS dan veel groter, waardoor de spanningsverliezen toenemen. Daarnaast zijn er capacitieve en inductieve effecten bij stroomveranderingen, waardoor zowel hogere als lagere spanningen kunnen optreden. Tevens is er het probleem van reflecties en overspraak naar naburige geleiders [Catt, 1967]. Logische signaalniveaus zien er dan ook wat rafelig uit. Genoemde verschijnselen moeten door de beschikbare marges opgevangen worden. De 'marges' zijn heel essentieel om een robuuste schakeling te kunnen ontwerpen. Bij zeer snelle logica gaat een toenemend deel van de ontwerpactiviteiten in het bestrijden en/of compenseren van genoemde verschijnselen zitten.

Tijdens het schakelen van CMOS zijn de *stroomveranderingen* ten opzichte van de ruststroom zeer groot. De voeding moet deze veranderingen goed kunnen volgen. De voeding moet daartoe hoogfrequent 'ontkoppeld' worden. Over de voeding van elk IC op een board wordt daartoe een condensator met goede hoogfrequent eigenschappen aangebracht [Funk, 1986/2].

Ground bounce

Bij snelle IC's komen een aantal hoogfrequent eigenschappen van onder andere de verbindingen naar voren, die een sterk versturende invloed kunnen hebben op de data-overdracht tussen IC's. Deze verschijnselen blijven laagfrequent verborgen omdat het effect ervan dan relatief gering is. Een van deze verschijnselen is *ground bounce*.



Figuur 5.26. Ground bounce en afgeleide verschijnselen.

Figuur 5.26.a beschrijft het uitgangsmodel van een CMOS invertor. De toevoerdraden van de voedingsspanningen naar het IC hebben een zekere zelfinductie. Hetzelfde is het geval voor de verbinding van de datalijn naar andere IC's. De zelfinducties L_{DD} , L_{SS} en L_{OUT} zijn zeer klein, maar daar de impedantie ervan evenredig is met ωL wordt de invloed met het toenemen van de hoogfrequent componenten in het spectrum van de signalen (zie de Inleiding) steeds groter. Tijdens het schakelen van CMOS zijn de stromen relatief zeer

groot, waardoor over de zelfinducties L_{DD} en L_{SS} inductiespanningen ontstaan. Deze inductiespanningen zijn evenredig met

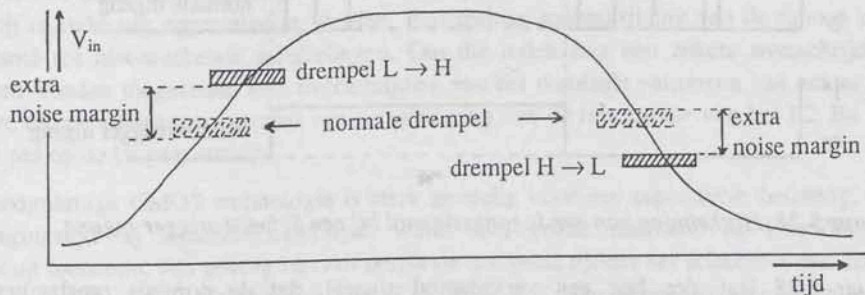
$$V_{ind} = -L \frac{di}{dt} \quad (5.4.)$$

In snelle IC's kan hierdoor tijdens het schakelen 1 à 2 Volt van de effectieve voedingsspanning afgaan respectievelijk erbij komen (bij het toenemen respectievelijk het afnemen van de stroom van V_{DD} naar V_{SS}). Deze interne veranderingen van de voedingsspanning zien we in de uitgangssignalen als rimpels en/of uitslingeringen terug. Zie figuur 5.26.

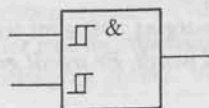
De verschijnselen die onder ground bounce vallen zijn nooit geheel te vermijden. Korte aansluitingen naar V_{DD} en V_{SS} dragen bij tot een vermindering ervan, alsmede een goede ont koppeling van de voeding. Ook het apart voeden van de uitgangstrappen kan bijdragen tot een beter extern gedrag. De overblijvende uitslingeringen moeten opgevangen worden door voldoende grote AC/DC noise margins. Dit heeft consequenties voor de tijdsmodellen van signalen en poorten. Zie hiervoor paragraaf 5.9. Een verdere behandeling van de hier geschetste problematiek valt buiten het kader van deze tekst. Zie bijvoorbeeld [National, 1989; Shear, 1989].

Bij het ontwerpen van printed circuit boards voor snelle logica speelt de hiervoor genoemde problematiek ook sterk mee. Het is een lastig en gedeeltelijk wat ongrijpbaar onderwerp. Het boek 'High-Speed Digital Design' [Johnson, 1993] heeft niet voor niets de ondertitel 'A Handbook of Black Magic' meegekregen. Zie verder de literatuur [Goedbloed, 1990; Williams, 1992] over EMC (Elektro-Magnetische Compatibiliteit).

Schmitt-trigger ingangen



(a) principe Schmitt-trigger ingang



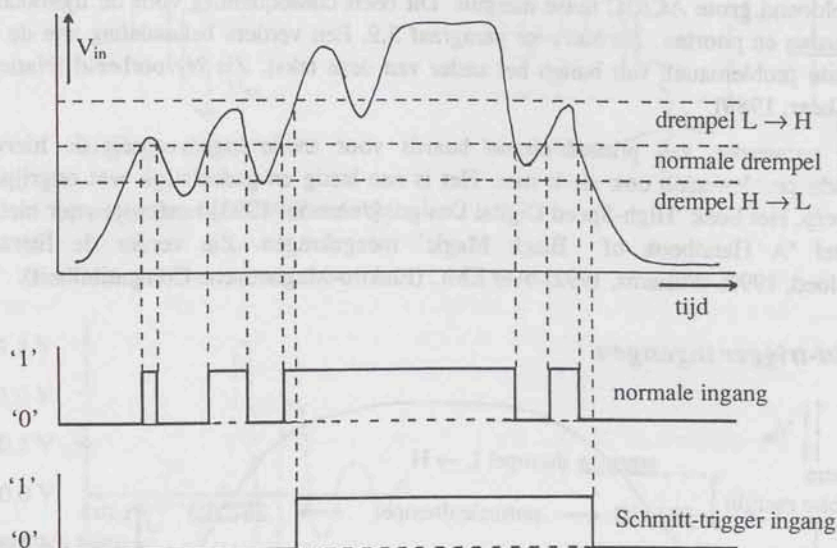
symbol voor
Schmitt-trigger ingang

(b) AND poort met Schmitt-trigger ingangen

Figuur 5.27. Schmitt-trigger ingangen.

Voor bepaalde toepassingen zijn de normale ruismarges niet voldoende om de optredende ruis en/of niveauverschuivingen op te vangen. Met name is dit het geval als ingangen gestuurd worden door signalen, die over relatief lange verbindingen gaan zoals bij bussen. Dan past men *Schmitt-trigger ingangen* toe. Bij een Schmitt-trigger ingang ligt de $L \rightarrow H$ drempel op een hoger niveau dan gebruikelijk en de $H \rightarrow L$ drempel op een lager niveau. Zie figuur 5.27 voor een toelichting. Schmitt-trigger ingangen hebben het volgende effect. Stel de $L \rightarrow H$ drempel is de geldende referentiedrempel (het huidige signaalniveau is L). De referentiedrempel verandert in de $H \rightarrow L$ drempel zodra hetingangssignaal de $L \rightarrow H$ drempel is gepasseerd. Daarna blijft deingangsspanning als 'H' gezien worden totdat de lagere $H \rightarrow L$ drempel weer gepasseerd is. Schmitt-trigger ingangen voegen meestal ≥ 0.4 Volt aan de DC ruismarges toe. Vooral bij lange *buslijnen* kan deze extra marge heel essentieel zijn voor een robuuste werking van de schakeling [Bundalo, 1989].

Voorbeeld



Figuur 5.28. Herkenning van eeningangssignaal bij een Schmitt-trigger ingang.

Figuur 5.28 laat zien hoe een veranderend signaal, dat de normale omslagdrempel overschrijdt, door een Schmitt-trigger ingang herkend wordt. □

Schmitt-trigger ingangen geven grotere AC/DC noise margins. Dit gaat ten koste van het (vroegste) tijdstip waarop een signaalovergang herkend wordt. Er treedt enige flank jitter op. Ook hier heeft robuustheid dus een prijs!

Compatibele impedanties.

Standaard CMOS is een hoogohmiger technologie dan de bipolaire TTL technologie [Malvino, 1986]. Bij data-overdracht tussen CMOS en TTL stuurt een hoogohmige bron een laagohmige ontvanger of andersom. Er is sprake van een zekere misaanpassing. Dit kan gevolgen hebben voor het *dynamische gedrag* tijdens het schakelen. Metingen tonen aan dat als CMOS TTL stuurt, het nieuwe signaalniveau pas veel later bereikt wordt dan uit een optelling van de propagatietijden van de afzonderlijke IC's volgt. Een extra propagatietijd van 20 tot 50 ns is heel normaal. Andersom is er geen toename van de propagatietijd. Hetzelfde komt voor bij data-overdracht tussen IC's uit verschillende series van één technologie. De verschillen zijn dan alleen kleiner. Zie ook [Funk, 1986; Havasy, 1986].

Ook deze impedantieverschillen moeten meegenomen worden bij het onderzoek naar het compatibel zijn van componenten. Welke invloed hebben impedantieverschillen op de timing parameters? Een bijkomend gevolg is dat minder steile flanken in CMOS tot een sterke verhoging van de dissipatie leiden. Een afgeleid effect van het meer of minder incompatibel zijn van de impedanties van de componenten.

Fanin en fanout

Wanneer de belasting van een uitgang toeneemt verloopt een eventuele uitgangsverandering ook langzamer. Ook hierin wordt een verschil aangetroffen tussen bipolair en CMOS. Bipolair is laagohmig. Als gevolg daarvan is de vertragingstijd (stijg/daaltijd) bij een uitgangsverandering niet al te groot. Een uitgangstransitie zal niet veel last hebben van de *capacitieve belasting* van een verbindingslijn en van de ingangen van de in te stellen poorten. In bipolaire IC's is de maximale stroom die de uitgang kan leveren bepalend voor de grens voor het aantal ingangen dat de uitgang kan sturen. Fanout voorschriften dienen daarbij nauwkeurig opgevolgd te worden. Een geringe overschrijding van de fanout leidt niet snel tot niet-werkende schakelingen. Om die reden zou een zekere overschrijding kunnen worden toegestaan. Een overschrijding van het nominale vermogen kan echter tot andere effecten leiden, waaronder een *vermindering van de levensduur* van het IC. En dat blijkt pas op de langere termijn!

De hoogohmige CMOS technologie is sterk gevoelig voor een capacitieve belasting. De vertragingstijd bij signaalveranderingen wordt snel groter naarmate de (capacitieve) belasting toeneemt. Als gevolg hiervan neemt de dissipatie tijdens het schakelen toe. In de statische toestand is de fanout van CMOS schakelingen daarentegen zeer groot en is de dissipatie zeer klein. Om tijdens het schakelen minder last te hebben van genoemde verschijnselen hebben recentere CMOS families op elke externe uitgang een *uitgangsbuffer* zitten. Deze buffer past de impedantie van de uitgang aan. Het omschakelen onder belasting gaat daardoor belangrijk sneller. Voor meer informatie zie [Hedenstierna, 1987; Larsson, 1994].

De belasting die een ingang vormt voor een sturend signaal wordt meestal genormaliseerd tot een *fanin van één standaardingang*. Vaak echter stelt een externe ingang van een IC verschillende interne ingangen van het IC in. In dat geval vormt zo'n externe ingang een

grotere belasting voor het sturende IC dan de standaardbelasting. Het is gebruikelijk in die gevallen een extra ingangsaanpassing te plegen in de vorm van een *buffer/versterker*. Meestal bestaat deze uit een invertor.

Conclusies ten aanzien van het compatibel zijn

We komen tot de volgende conclusies.

- Een IC heet compatibel met een ander IC als compatibiliteit op *alle relevante facetten* is geverifieerd. Het verificatietraject loopt daarbij van correcte en robuuste data-overdracht tot en met de afmetingen van de behuizing van het IC. Ook eventuele verschillen in levensduur kunnen belangrijk zijn.
- Bij de verificatie van parameterwaarden dienen de worst case en de best case condities over het hele werkgebied van de schakeling onderzocht te worden. In de praktijk maakt men dan een zogenaamde shmoo plot, waarbij aangegeven wordt in welke gebieden de schakeling wel en in welke de schakeling niet werkt. De gebieden waarin de schakeling niet werkt moeten dan voldoende ver van het gewenste werkgebied liggen.
- De voor de verificatie benodigde informatie is moeilijk compleet te krijgen. Veel staat in application notes, maar zeker niet genoeg.
- In het recente verleden is veel waarde toegekend aan het 'pin compatible' zijn van nieuwe logische families met bestaande logische families. Bij dit streven ging men zelfs zover dat hele series werden gekopieerd, inclusief alle conceptuele fouten uit de eerdere serie. Het is duidelijk dat dit zinloos is, omdat als gevolg van onder andere parameterverschillen een bestaand ontwerp meestal niet zonder meer in een nieuwe familie kan worden gekopieerd. Vaak is het opnieuw ontwerpen eerder gebeurd dan het zoeken naar de (incidenteel optredende) fouten in het gekopieerde ontwerp.
- Het verdient aanbeveling uiterst terughoudend te zijn met de toekenning van het predikaat 'compatibel' en er in elk geval bij te vermelden welke facetten daarbij geverifieerd zijn. (Beter zou zijn: welke niet!)
- Het verdient daarom aanbeveling een ontwerp in één logische familie te realiseren of in één technologie. Door de huidige omvang van ASIC's (Application Specific IC's) zal dit streven niet bezwaarlijk zijn.

De keuze van de juiste logische familie voor een bepaalde toepassing is geen gemakkelijke. Er bestaan op dit moment zeer veel logische families, geoptimaliseerd naar snelheid, vermogensdissipatie, fanout naar andere IC's en ruismarges. Vaak zijn deze criteria strijdig met elkaar en moet er een prioriteitsvolgorde worden vastgesteld. Zo zijn een hoge schakelsnelheid en grote ruismarges strijdig. Vooral wanneer een toepassing op de grens van de mogelijkheden van een bepaalde technologie ligt is een zorgvuldige afweging geboden. Zie ook [Hall, 1991].

5.9 Tijdmodellen

Componenten hebben naast een logische functie ook *fysische eigenschappen*. Een ervan is dat er een zekere tijd nodig is om te reageren op veranderingen. In de nu volgende paragrafen introduceren we een *tijdmodel* voor signalen, verbindingen en componenten. Het doel van het tijdmodel is het *afschermen van het fysische ontwerpniveau*, voor zover dit op het logische ontwerpniveau mogelijk is. Hierdoor kunnen we op het logische niveau op een zodanige wijze ontwerpen dat het ontwerp ook in alle (de meeste?) gevallen realiseerbaar is. Een noodzakelijke voorwaarde om top-down te kunnen ontwerpen.

Het te introduceren tijdmodel is een *eenvoudig* tijdmodel. Het dient om op logisch niveau bepaalde concepten te kunnen introduceren en toe te lichten. Het model ondersteunt dus niet automatisch alle concepten op de lagere ontwerpniveaus. Daartoe moet het tijdmodel in een aantal gevallen uitgebreid worden. Uiteraard mogen de op een hoger niveau getrokken conclusies *niet in strijd zijn* met die van de lagere ontwerpniveaus. Concreet: de grenzen van een geschatte propagatietijd worden met een gedetailleerder tijdmodel wel nauwkeuriger bepaald, maar als dit tot gevolg heeft dat het eerder vastgestelde maximum groter wordt is er een probleem.

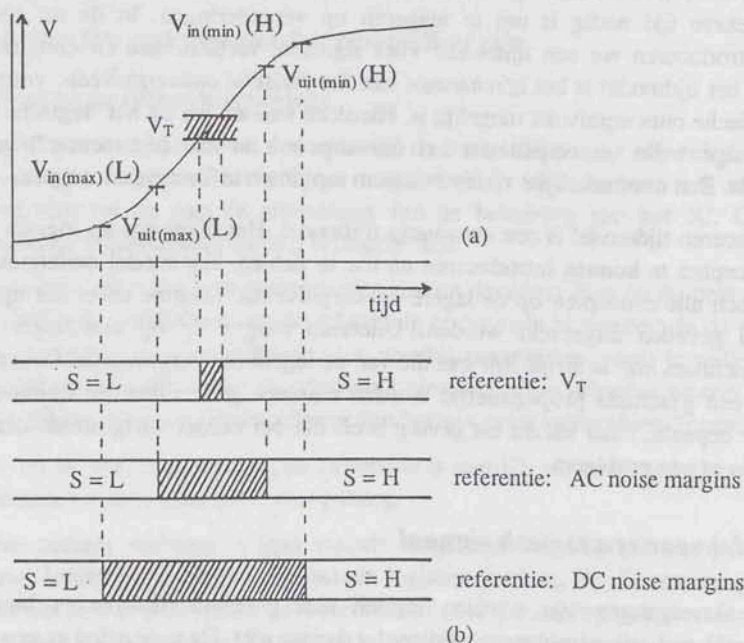
Het tijdmodel voor een logisch signaal

Tijdens signaalovergangen zijn signalen logisch niet gedefinieerd. Slechts binnen de gebieden voor H en L zijn signalen gedefinieerd, erbuiten niet. Daar signalen er een zekere tijd over doen om te veranderen zal het signaalmodel tot uitdrukking moeten brengen wanneer een signaal wel en wanneer een signaal niet gedefinieerd is. Bij het wel of niet gedefinieerd zijn van een signaal moeten we niet alleen kijken naar het logische niveau van het signaal zelf, maar ook naar alle overige aspecten die bij het concept 'logisch signaal' behoren. Met name geldt dit voor de AC en DC noise margins (ruismarges).

In figuur 5.29 zijn alle relevante parameters van een logisch signaal in relatie tot de in te stellen poort aangegeven. *Een logisch signaal is gedefinieerd als het signaal voldoet aan alle kenmerken die een signaal tot een logisch signaal maken.* Voor figuur 5.29 betekent dit dat een signaal geen 'logisch' signaal meer is als het tijdens een $L \rightarrow H$ overgang de grenswaarde $V_{\text{uit(max)}}(L)$ overschrijdt. Immers, op dat moment is de DC noise margin niet meer (geheel) aanwezig. (Nemen we genoegen met een kleinere DC noise margin, dan kan het niveau wat hoger gesteld worden, maar nooit hoger dan $V_{\text{in(max)}}(L)$ van de in te stellen ingang.) Hetzelfde geldt aan de bovenzijde. Het signaal heeft weer een logische waarde als het signaal het niveau $V_{\text{uit(min)}}(H)$ passeert. De DC noise margin is dan weer aanwezig.

De tijdbalken in figuur 5.29.b geven aan wanneer het in figuur 5.29.a beschreven signaal gedefinieerd is en wanneer niet. De flank van het signaal (de stijgtijd in dit geval) en de gekozen referentiepunten bepalen de breedte van het niet-gedefinieerde gebied. De stijgtijd/daaltijd maakt het niet-gedefinieerde gebied mede *afhankelijk van de toepassing*. In de praktijk verschillen de meningen ten aanzien van de referentieniveaus. Sommigen zeggen dat het voldoende is als een signaal uit de buurt van het drempelgebied (threshold) blijft.

Boven de drempel interpreteert een ingang de momentele waarde van het signaal als H en onder de drempel als L.



Figuur 5.29. Het tijdmodel van een logisch signaal.

Bij deze keuze wordt er geen rekening gehouden met de AC en DC noise margins. En die zijn juist bij signaalovergangen, in de *dynamische toestand* van de schakeling, erg belangrijk en wel om de *robuustheid tegen storingen* te kunnen garanderen. Hetzelfde geldt als de niveaus $V_{in(min)}(H)$ en $V_{in(max)}(L)$ als referentie worden aangehouden. Dan zijn de AC noise margins gegarandeerd, maar nog niet de volledige DC noise margins. Het begint er al op te lijken, maar we zijn er nog niet. Zodat we uiteindelijk moeten besluiten $V_{uit(min)}(H)$ en $V_{uit(max)}(L)$ als referentieniveaus voor de timing te nemen!

Uit figuur 5.29 zien we dat, naarmate we meer facetten bij de modelvorming in beschouwing nemen, de gebieden waarin het signaal gedefinieerd is kleiner worden en dat de onzekerheid in de vorm van de lengte van het niet-gedefinieerde interval toeneemt. Nemen we nu vervolgens ook in beschouwing dat de voedingsspanning binnen zekere grenzen kan variëren, dat de temperatuur varieert en dat het proces waarin een IC gemaakt wordt een zekere spreiding kent, dan zal duidelijk zijn dat al deze variaties een effect op de schakelsnelheid kunnen hebben. De beïnvloeding van de flanken kan daarbij zowel positief als negatief uitpakken. Ook de feitelijke logische niveaus kunnen wat verschuiven. Kortom, het niet-gedefinieerde gebied rond een signaalovergang wordt alsmaar groter.

In databoeken van IC's wordt één vast referentieniveau V_{ref} genomen, waaraan de timing parameters refereren. Dit niveau is meestal 0,5 V_{DD} bij CMOS en 1,5 Volt bij TTL logica.

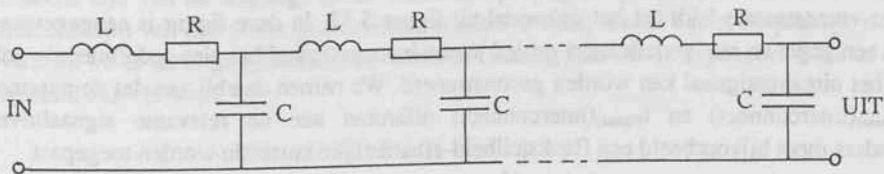
Verder worden parameters onder *typical conditions* gemeten. Het is dan aan degene die het IC toepast om de gespecificeerde parameters te vertalen naar meer realistische waarden. Op zich een te verdedigen handelwijze, omdat de timing ook van de toepassing afhangt en deze kan in een databoek niet voorzien worden. Met name hangt de flanksteilheid sterk af van de belasting van een uitgang. Vervolgens wordt in de databoeken wel nagelaten gedetailleerd te beschrijven wat er allemaal in rekening gebracht moet worden en hoe dit moet!

Vraag

Schmitt-trigger ingangen hebben verschoven drempels voor de signaalovergangen. Heeft dit gevolgen voor het signaalmodel van het signaal dat zo'n ingang instelt, rekening houdend met onder andere de DC en AC noise margins, en zo ja wat voor gevolg? $\square$

Het tijdmodel voor een verbinding

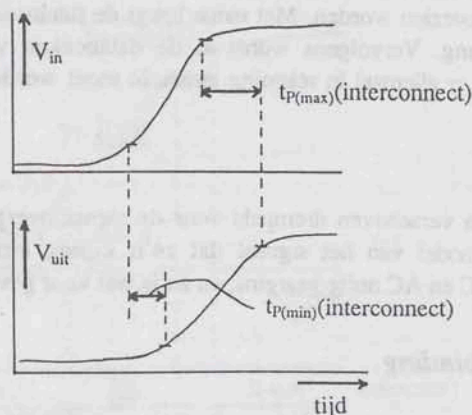
Figuur 5.30 toont een eenvoudig vervangingsschema van een *transmissielijn*. Zowel door het inductieve effect als door het capacitieve effect wordt een signaalovergang vertraagd doorgegeven. Deze vertraging is afhankelijk van de lengte van de transmissielijn, maar ook van de frequentie van het signaal. Het tijdmodel voor een verbinding is daarom afhankelijk van de lengte van de lijn en van de frequentie van het signaal erop. Daarnaast zijn ook de afmetingen van de draden/banen van invloed. Al met al een redelijk complexe afhankelijkheid.



Figuur 5.30. Elektrisch model van een transmissielijn.

Figuur 5.31 geeft aan hoe een verbinding lijn kan omgaan met het door te geven signaal. De aangeboden signaalovergang wordt enigszins *vertraagd* en wat *vervormd* doorgegeven. (De aard van de niet-lineaire vervorming is ook afhankelijk van hoe de uitgang van de lijn is afgesloten. Passen we het eerder opgestelde signaalmodel toe, dan komen we tot de definitie van twee parameters, $t_{P(min)}(interconnect)$ en $t_{P(max)}(interconnect)$. De parameter $t_{P(min)}(interconnect)$ geeft de tijd aan die verloopt tussen het *verlaten van het gedefinieerde gebied* aan de ingangskant van de verbinding lijn enerzijds en het hieraan gekoppelde verlaten van het gedefinieerde gebied door het uitgangssignaal anderzijds en wel onder *zo gunstig mogelijke condities*. Hetzelfde geldt voor $t_{P(max)}(interconnect)$ ten aanzien van het *opnieuw gedefinieerd zijn* van de signaalniveaus aan de ingang en aan de uitgang. Voor deze parameter worden *worst case condities* genomen. Deze parameters voor verbinding lijnen zijn binnen het bekende model van transmissielijnen uit te rekenen of te simuleren.

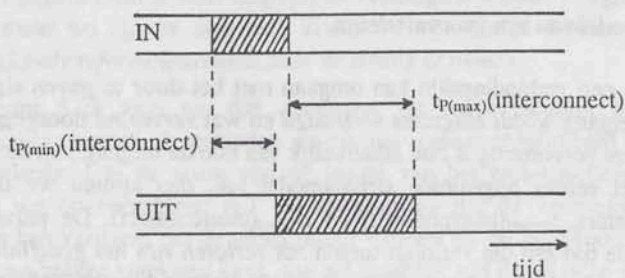
[van der Meijs, 1992]. Gezien de rekentijd doet men dit meestal uitsluitend voor de gebleken tijdkritische paden.



Figuur 5.31. Definitie van de parameters van een transmissielijn.

Er zijn ook andere definities mogelijk. We hebben hier gekozen de *beschikbaarheid van de signaalwaarde* (tot/vanaf wanneer is het signaal logisch gedefinieerd) als criterium te nemen en dit te vertalen naar een worst case model ten aanzien van de duur van het niet-gedefinieerde gebied. Er zijn ook andere delay modellen in gebruik. Zie hiervoor [Abramovici, 1990].

Het vorenstaande leidt tot het tijdmodel uit figuur 5.32. In deze figuur is aangegeven hoe bij een gegeven niet-gedefinieerd gebied in een ingangssignaal het niet-gedefinieerde gebied in het uitgangssignaal kan worden geconstrueerd. We nemen daarbij aan dat de parameters $t_{P(min)}(interconnect)$ en $t_{P(max)}(interconnect)$ refereren aan de relevante signaalniveaus. Anders moet bijvoorbeeld een flanksteilheid-afhankelijke correctie worden toegepast.



Figuur 5.32. Tijdmodel voor een verbinding.

Opmerking

Signalen hebben de neiging na een signaalovergang wat uit te slingeren. Deze *uitslingerverschijnselen* spelen zich niet noodzakelijk binnen de gedefinieerde

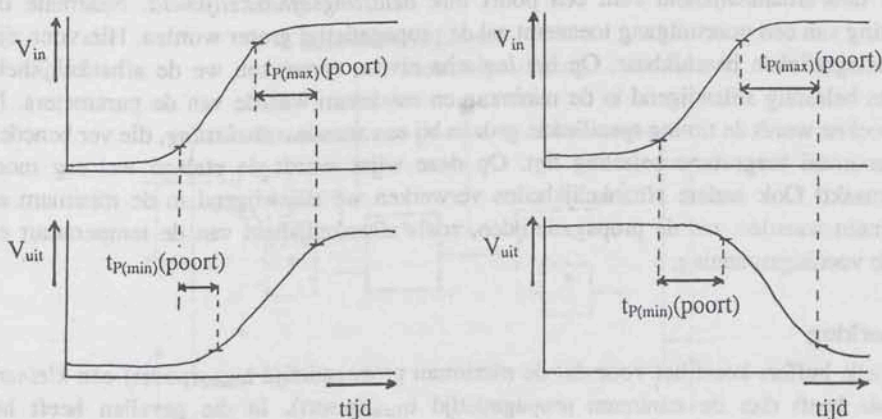
signaalniveaus af. In dat geval telt de propagatietijd tot het tijdstip waarop na de laatste uitslingering het gedefinieerde signaalniveau weer is bereikt. $\square$

In vrijwel alle toepassingen van dit signaalmodeel zullen we op het *logische ontwerpniveau* geen onderscheid maken tussen de verschillende verbindingen in een schakeling. We nemen aan dat bekend is welke verbinding de kortste en welke de langste propagatietijd in de schakeling heeft. Indien we op alle verbindingen met deze twee tijden rekenen en er blijkt genoeg tijd te zijn, dan is betrouwbare data-overdracht mogelijk. Uiteraard zijn er verfijningen mogelijk, maar dan moet alles op layout niveau doorgerekend worden [van der Meys, 1992]. Gezien de benodigde rekentijd beperkt men dit bij voorkeur tot de tijdkritische paden.

Het tijdmodel voor een poort

Signalen hebben een zekere propagatietijd door componenten. Bedroeg deze propagatietijd een tiental jaren geleden nog ca. 10 ns per poort, op dit moment zijn propagatietijden van 0.2 - 0.5 ns goed realiseerbaar. Voor de propagatietijd van signalen door componenten gelden overeenkomstige overwegingen als voor verbindingen. Bepalend is de tijd die een component er na een ingangsverandering over doet om het bedoelde effect op de uitgang te realiseren. Bij het berekenen of meten van de propagatietijden refereren we weer aan de gebruikelijke referentieniveaus op de flanken van de ingangssignalen en de uitgangssignalen.

Figuur 5.33 laat zien hoe de minimum propagatietijd (refereert aan het niet meer gedefinieerd zijn van de uitgang) en de maximum propagatietijd (refereert aan het weer gedefinieerd zijn) van een poort gedefinieerd worden. Daarbij maakt het in de praktijk nog enig verschil uit of de uitgangstransitie in dezelfde richting gaat als de ingangstransitie of er tegengesteld aan verloopt.

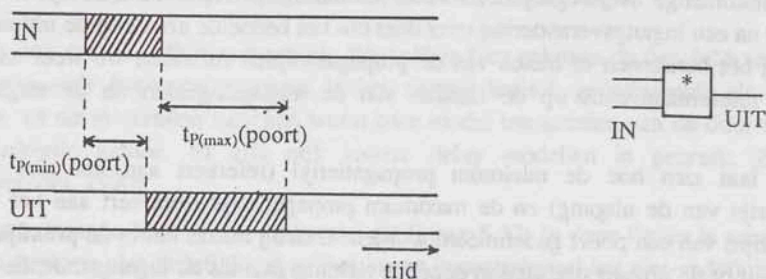


Figuur 5.33. Definitie minimum en maximum propagatietijd van een poort.

Om praktische redenen maken we in deze tekst ook geen onderscheid tussen de propagatietijden van verschillende signaalovergangen ($L \rightarrow H$ en $H \rightarrow L$). We modelleren een poort met 'worst case' parameters, waarbij de *minimum propagatietijd* $t_{P(\min)}(\text{poort})$ het minimum is dat onder de verschillende omstandigheden gevonden wordt (referentie: het verlaten van de gedefinieerde signaalniveaus aan de ingang respectievelijk aan de uitgang). Op overeenkomstige wijze wordt de *maximum propagatietijd* $t_{P(\max)}(\text{poort})$ bepaald. Op deze wijze wordt vermeden dat er op het logische ontwerpniveau een *data-afhankelijkheid* in de timing verificatie sluipt. (Dit neemt niet weg dat dit verderop in het ontwerptraject wel gedaan wordt, maar dan meestal in tijdkritische situaties.)

Voorbeeld

Figuur 5.34 laat zien hoe een signaal modelmatig door een poort propageert. Als het ingangssignaal verandert, dan gaat dat altijd gepaard met een niet gedefinieerd gebied in het ingangssignaal. Met de afgeleide timing parameters van de poort kan nu bepaald worden tot hoe lang de poortuitgang nog gedefinieerd is voor de *oude signaalwaarde* en vanaf wanneer de *nieuwe signaalwaarde* aan de poortuitgang beschikbaar is. □



Figuur 5.34. Signaalpropagatie door een poort.

Naast data-afhankelijkheid kent een poort ook *belastingsafhankelijkheid*. Naarmate de belasting van een poortuitgang toeneemt zal de propagatietijd groter worden. Hiervoor zijn correctiegrafieken beschikbaar. Op het logische niveau verwerken we de afhankelijkheid van de belasting stilzwijgend in de minimum en maximum waarde van de parameters. In databoeken wordt de timing specificatie gedaan bij een *standaardbelasting*, die ver beneden de maximaal toegestane belasting ligt. Op deze wijze wordt de etalage wel erg mooi opgemaakt! Ook andere afhankelijkheden verwerken we stilzwijgend in de minimum en maximum waarden van de propagatietijden, zoals afhankelijkheid van de temperatuur en van de voedingsspanning.

Opmerking

Bij snelle buffers komt het voor dat de maximum propagatietijd $t_{P(\max)}(\text{poort})$ een *kleinere waarde* heeft dan de minimum propagatietijd $t_{P(\min)}(\text{poort})$. In die gevallen heeft het ingangssignaal een beduidend grotere stijg- of daaltijd dan het uitgangssignaal. De snelle buffer *regeneert* het signaal. Dit verschil is een gevolg van de manier van definiëren ten opzichte van verschillende referentieniveaus en van het feit dat de minimum propagatietijd

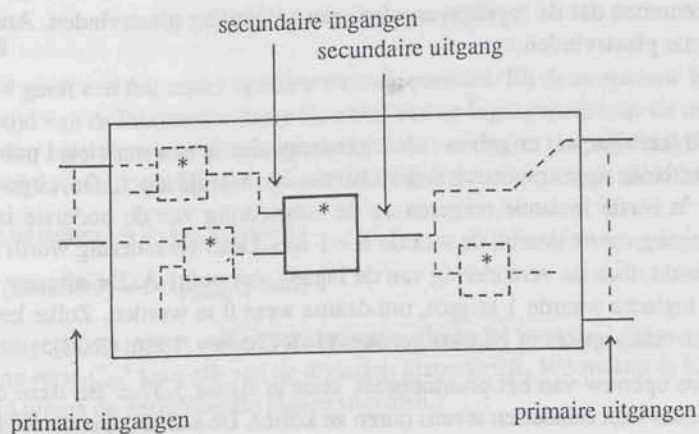
aangeeft tot wanneer de *oude* waarde nog beschikbaar is (*best case* conditie voor de propagatietijd) en de maximum propagatietijd aangeeft vanaf wanneer de *nieuwe* waarde weer beschikbaar is. Dit laatste betekent het opzoeken van de *worst case* condities voor de propagatietijd voor zover het de *nieuwe* signaalwaarde betreft. Bij deze definitie van de propagatietijden zijn we uitgegaan van de beschikbaarheid van de oude respectievelijk de nieuwe signaalwaarde. Derhalve een toepassingsgerichte keuze. $\square$

Bij het afleiden van voorwaarden voor de timing van schakelingen op het logische ontwerpniveau was het tot voor kort gebruikelijk de *propagatietijd van de verbindingen* te verwaarlozen ten opzichte van de *propagatietijd door poorten*. Bij niet al te lange verbindingen gaf dit geen grote fouten. Bij de huidige snelle CMOS logica gaat deze veronderstelling niet meer op. De propagatietijden in de verbindingen zijn zelfs vaak groter dan de propagatietijden door de logica. Zie [Bakoglu, 1990, Ch. 4].

Echter, voor het effect op de signaaloverdracht maakt het bij een minimum-maximum model niet uit wat de oorzaak van de signaalvertraging is. Slechts het totale effect telt. Daarom zullen we bij het afleiden van timing voorwaarden de propagatietijd van de verbindingen meestal stilzwijgend bij de propagatietijd van de poorten tellen. In formules komt dan alleen de propagatietijd van de poorten als parameter voor. Op de lagere ontwerp niveaus, daar waar het gaat om het minimaliseren van de propagatietijd, moeten we uiteraard de twee gevallen afzonderlijk bestuderen.

5.10 Timing van multilevel combinatoriek

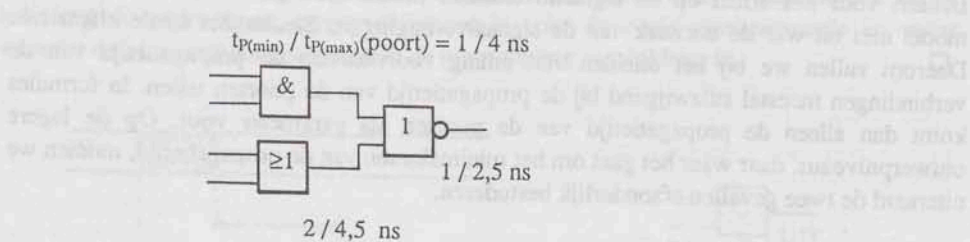
De timing parameters van multilevel combinatoriek leiden we af van de timing parameters van de afzonderlijke poorten. Daarbij gaan we uit van een *best case* en een *worst case* *sommatie* van de betreffende parameters. Verfijningen, die bijvoorbeeld met een statistisch model behaald kunnen worden, laten we buiten beschouwing.



Figuur 5.35. Multilevel combinatoriek.

Multilevel combinatorische schakelingen zijn opgebouwd uit verschillende niveaus van poortschakelingen achter elkaar. Dit heeft tot gevolg dat de paden door zo'n schakeling een verschillende lengte kunnen hebben. Daaraan zijn verschillende propagatietijden gekoppeld. Figuur 5.35 geeft een toelichting op de wijze waarop we de timing parameters voor multilevel combinatoriek berekenen. De ingangssignalen van een poort of een netwerk van poorten hebben een verschillende timing. Als *minimum propagatietijd van de (secundaire) poortuitgang* definiëren we de tijd die we vinden door van alle ingangssignalen van de poort de minimum aankomsttijd van de *niet-gedefinieerde gebieden* te bepalen. Hierbij wordt de minimum propagatietijd van de poort opgeteld. Daaruit vinden we het begin van het niet-gedefinieerde signaalinterval aan de uitgang van de betreffende poort. Hetzelfde doen we voor de *maximale propagatietijd*, maar nu voor het tijdstip waarop de uitgang weer *gedefinieerd* is. We gaan daarbij uit van de *data-onafhankelijke* parameters.

Voorbeeld



Figuur 5.36. Propagatietijden voor een schakeling.

Voor de poorten van de schakeling in figuur 5.36 gelden de op de verschillende niveaus aangegeven propagatietijden. Voor de gehele schakeling geldt:

$$t_{P(\min)}(\text{netwerk}) = 1 + 1 = 2 \text{ ns}$$

$$t_{P(\max)}(\text{netwerk}) = 4,5 + 2,5 = 7 \text{ ns}$$

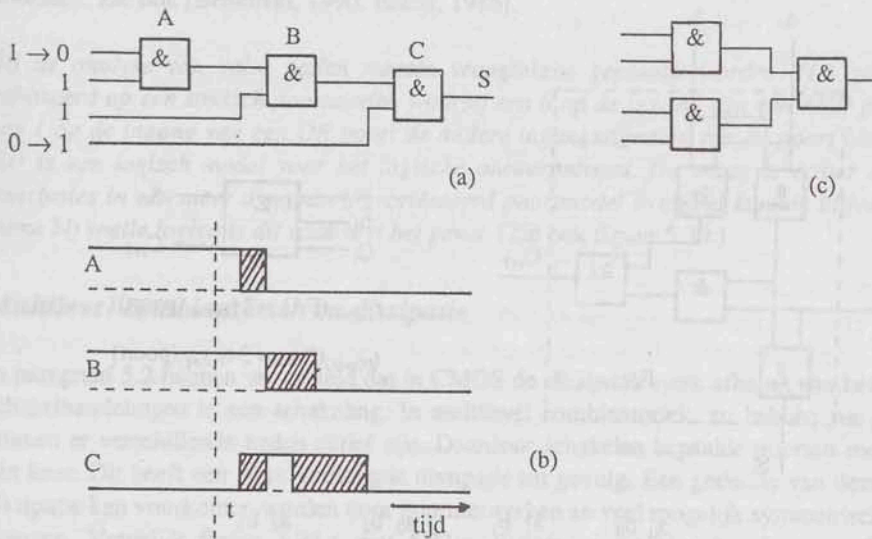
Hierbij is aangenomen dat de ingangsveranderingen gelijktijdig plaatsvinden. Anders moet nog een correctie plaatsvinden. □

Voorbeeld

Figuur 5.37.a/b laat zien wat er gebeurt als ingangssignalen in een multilevel poortnetwerk veranderen. De beide ingangsveranderingen starten op het tijdstip t . De uitgang van de schakeling zal in eerste instantie reageren op de verandering van de onderste ingang van poort C. De uitgang neemt daarbij de waarde $S = 1$ aan. Deze verandering wordt even later ongedaan gemaakt door de verandering van de ingang van poort A. De uitgang van poort C zal even de logische waarde 1 krijgen, om daarna weer 0 te worden. Zulke kortdurende veranderingen worden *spikes* of *glitches* genoemd [McCluskey, 1986, Ch. 3].

Een alternatieve opbouw van het poortnetwerk staat in figuur 5.37.c. Bij deze opbouw is de kans op glitches veel minder en tevens duren ze korter. De achterliggende schakeling zal er dan waarschijnlijk niet op reageren. We merken nadrukkelijk op dat ook ingangsveranderingen op verschillende tijdstippen kunnen komen en daardoor spikes kunnen

veroorzaken. Het 'balanceren' van de paden door een combinatorisch netwerk lost dus niet alle problemen rond spikes op.



Figuur 5.37. Multilevel poortnetwerk.

Het modelleren van modulair opgebouwde schakelingen

Bij het samenstellen van grotere schakelingen wordt vaak gebruik gemaakt van eerder ontworpen modules. In principe worden de timing parameters van de gehele schakeling dan gebaseerd op de timing parameters van de samenstellende modules. Dit gaat op dezelfde wijze als bij poortschakelingen gedaan is. Daarbij komt men onverwachte valkuilen tegen.

Voorbeeld

Figuur 5.38 geeft een full adder opbouw met vijf poorten. Bij deze opbouw is de minimum propagatietijd van de inkomende carry C_{in} en/of van de ingangsdata naar de uitgaande carry C_{uit} gelijk aan die van twee poortniveaus en is de maximum propagatietijd gelijk aan die van drie poortniveaus. Hieruit leiden we de *black box parameters* van de full adder af:

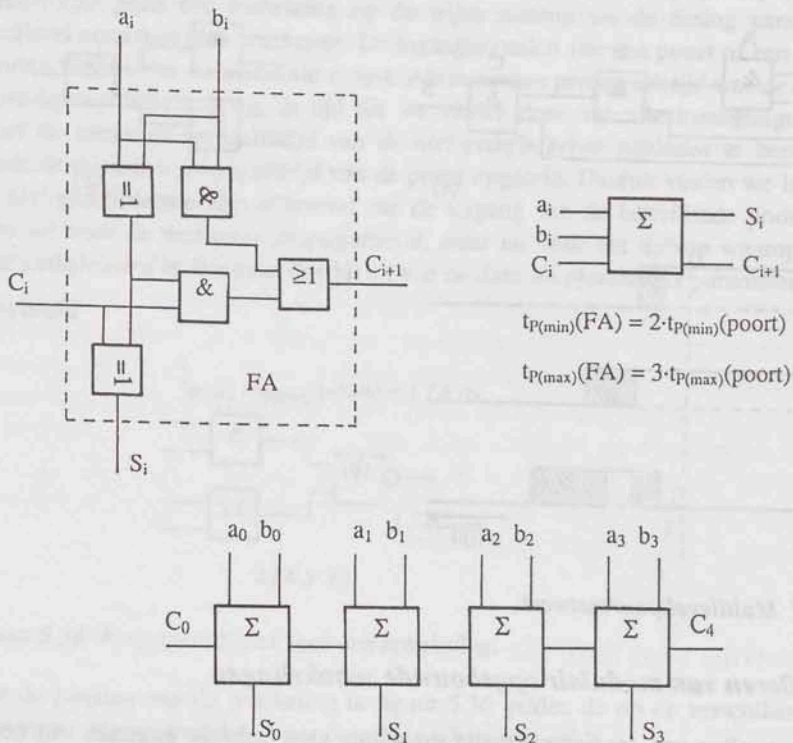
$$t_{P(\min)}(\text{fulladder}) = 2 \cdot t_{P(\min)}(\text{poort}) \quad (5.5.a)$$

$$t_{P(\max)}(\text{fulladder}) = 3 \cdot t_{P(\max)}(\text{poort}) \quad (5.5.b)$$

(Dus niet uitgesplitst in een minimum/maximum tijdmodel naar $t_{P(\min)}(\text{ingang-naar-}C_{uit})$ en $t_{P(\max)}(\text{ingang-naar-}C_{uit})$ voor elk van de ingangen afzonderlijk. We maken in het model geen onderscheid tussen de verschillende ingangssignalen.)

Bij een 4-bit full adder is de propagatietijd van de carry C_0 naar de carry C_4 gelijk aan 12 poortniveaus, uitgaande van drie poortniveaus per sectie. De feitelijke maximale duur van

de carry propagatie is echter die van negen poortniveaus. Ga dit na in de gegeven full adder structuur. (Verondersteld is dat alle databits en C_0 tegelijk aangeboden worden.)



Figuur 5.38. Modulaire opbouw van een 4-bit full adder. □

Het in het vorige voorbeeld beschreven fenomeen illustreert de prijs die betaald moet worden voor een eenvoudig timing model, waarbij geen onderscheid gemaakt wordt naar de propagatietijd van de verschillende ingangssignalen naar de diverse uitgangen. Komt men op basis van dit model tot de conclusie dat er voldoende tijd is voor een data-overdracht, dan is de timing daarmee geverifieerd. Zo niet, dan moet in een later stadium overgegaan worden op een meer gedetailleerd timing model, waarbij onder andere voor elke ingang afzonderlijk de propagatietijden van de signalen erop naar de uitgangen bepaald moeten worden. Vaak wordt daarbij de minimum propagatietijd gevonden als verscheidene of alle ingangen tegelijk veranderen.

Behalve het hiervoor beschreven fenomeen komen in combinatorische schakelingen ook *valse paden* voor. Een vals pad is een pad door een combinatorische schakeling, dat op basis van de geometrische structuur (uitgaande van het poortmodel of een ander black box model voor componenten) kan bestaan, maar dat in feite niet geactiveerd kan worden omdat de voor activering benodigde datacombinaties elkaar uitsluiten. Bij grote combinatorische schakelingen zoals parallelvermenigvuldigers komen veel valse paden

voor. Zou men het definitieve ontwerp baseren op het black box timing model, dan zijn in veel gevallen de geschatte propagatietijden te groot. Het voorbeeld hiervoor beschrijft een heel triviaal geval van een vals pad. In de praktijk is het probleem veel moeilijker te doorzien. Zie ook [Benkoski, 1990; Brand, 1986].

Bij de analyse van valse paden moeten vraagtekens geplaatst worden. Het geheel is gebaseerd op een statisch poortmodel, waarbij een 0 op de ingang van een AND poort of een 1 op de ingang van een OR poort de andere ingangssignalen van de poort blokkeert. Het is een logisch model voor het logische ontwerpniveau. De vraag is echter of deze conclusies in een meer dynamisch georiënteerd poortmodel overeind kunnen blijven. Met name bij snelle logica is dit vaak niet het geval. (Zie ook figuur 5.39.)

Multilevel combinatoriek en dissipatie

In paragraaf 5.2 hebben we gesteld dat in CMOS de *dissipatie* sterk afhangt van het aantal schakelhandelingen in een schakeling. In multilevel combinatoriek, zo hebben we gezien, kunnen er verschillende paden actief zijn. Daardoor schakelen bepaalde poorten meer dan één keer. Dit heeft een sterk verhoogde dissipatie tot gevolg. Een gedeelte van deze extra dissipatie kan voorkomen worden door poortnetwerken zo veel mogelijk symmetrisch op te bouwen. Vergelijk figuur 5.37.a met 5.37.c. Echter, niet alle extra dissipatie kan zo vermeden worden en wel omdat ook de ingangssignalen op een verschillend tijdstip kunnen veranderen. En dat laatste kan niet altijd in de achterliggende schakeling opgevangen worden, met name niet als de tijdstippen waarop de ingangen veranderen verschillend zijn. Het verminderen van de dissipatie is een onderwerp dat recent veel aandacht krijgt. Met de toenemende omvang van schakelingen in een IC begint dissipatie meer en meer de beperkende factor te worden [Leijten, 1995; Murgai, 1995]. Daarbij is steeds meer apparatuur portable. Ook hiervoor is het beperken van het energieverbruik noodzakelijk.

5.11 De toepassing van (tijd)modellen

Modellen spelen een essentiële rol bij het ontwerpen. Als men niet in staat is op een bepaald ontwerpniveau met modellen van de fenomenen van de lagere niveaus te werken is een *top-down benadering* van het ontwerpproces onmogelijk. Er moeten dan in een te vroeg stadium te veel details meegenomen worden, hetgeen de efficiëntie van het ontwerpproces niet ten goede komt. Een model moet daarom een zo eenvoudig en abstract mogelijk beeld geven van datgene wat op de lagere ontwerp niveaus nog moet worden uitgewerkt.

Het toepassen van modellen is slechts zinvol als het model relevant is voor het fenomeen dat het beschrijft. Tevens moeten modellen eenvoudig zijn. Nu is eenvoudig een relatief begrip. Met de hand wordt iets al gauw ondoenlijk. Computers en geavanceerde software pakketten maken veel mogelijk. Bij ons timing model zullen we vooralsnog het model afbakenen tot op een niveau dat voor de uitleg van alle relevante concepten op het logische ontwerpniveau nodig is. Het voert voor deze tekst te ver om alle invloeden op de timing

kwantitatief in kaart te brengen. Dit is meer de taak van degenen die het proces moeten karakteriseren waarin het IC gemaakt wordt [Jaeger, 1993].

Op het logisch ontwerpniveau werken we meestal met een *minimum* en een *maximum waarde* van de *timing parameters*. Hieraan ligt een goede schatting van alle relevante invloeden op de timing parameters ten grondslag. Voor het timing model moet onderzocht worden wat de invloed is van de temperatuur, de voedingsspanning, het proces (MOS/Bipolair of ander proces) en de spreiding van de parameters, eventuele data-afhankelijkheid in de vorm van de richting van een flank en de van het aantal schakelende ingangen afhankelijke flanksteilheid, alsmede de belasting van de uitgang door andere componenten. Nu is het niet eenvoudig uit deze veelheid van afhankelijkheden de minimum en maximum waarde van een bepaalde parameter te bepalen, te meer daar genoemde invloeden elkaar onderling ook beïnvloeden. Neem de voedingsspanning. Als deze toeneemt, dan worden de stromen door de schakeling naar evenredigheid groter en hiermee ook de dissipatie. Dit leidt tot een temperatuurverhoging. Deze heeft op zijn beurt weer invloed op het geleidingsvermogen van de transistoren en dus op de stromen in de schakeling. Soms kan bij simulatoren slechts met de minimum waarde van *alle* parameters gesimuleerd worden of met de maximum waarde. Het zal duidelijk zijn dat dit niet noodzakelijk tot de uiterste grenzen leidt.

Een belangrijk aspect bij het bepalen van de timing parameters is het toekomstige *werkgebied* van de schakeling. Het maakt veel uit of logica deel gaat uitmaken van een apparaat dat onder normale omgevingscondities gebruikt gaat worden, of dat het onder veel extremere omgevingscondities moet kunnen werken zoals meetapparatuur langs een snelweg. Onder normale omgevingscondities (typical conditions) kan met minder strenge eisen gewerkt worden, hetgeen in de praktijk betekent dat het niet-gedefinieerde gebied bij met name signaalovergangen kleiner wordt. (Berucht is de consequente toepassing van deze conclusie in apparaten met geforceerde ventilatie. Als deze uitvalt, dan gaan veel apparaten vrij snel stuk omdat bij de dimensionering te veel rekening is gehouden met de beschikbaarheid van geforceerde ventilatie. Een beperkte tijd zonder koeling zou moeten kunnen.)

Metten is weten

Wil dit bekende gezegde waar zijn, dan moet er op een zodanige wijze gewerkt worden dat inderdaad gemeten wordt wat men wil meten. Neem als voorbeeld de propagatietijd van een poort. In databoeken en in de documentatie van standaard cel bibliotheken wordt de propagatietijd vaak gespecificeerd als de reactietijd tussen een verandering op één ingang van de poort en de daarbij behorende uitgangsreactie. De andere ingangen zijn dan vast ingesteld. Hierbij wordt een meetfout gemaakt. In een OR poort bijvoorbeeld zitten in het nMOS deel de transistoren parallel. Een *parallelschakeling* van geleidende transistoren heeft een lagere impedantie dan die van elke transistor afzonderlijk. Om de minimum propagatietijd van een OR poort te meten moeten alle ingangen tegelijk schakelen en niet één ingang afzonderlijk. Tevens geldt dat de minimum propagatietijd van een CMOS OR poort gemeten moet worden bij de $H \rightarrow L$ ingangstransitie. Iets dergelijks geldt voor een

serieschakeling van transistoren. Daar is ook sprake van een worst case schakelconditie, omdat de transistoren in een serieschakeling één voor één gaan geleiden. Zie [Shoji, 1988, pp. 130-152].

Kennis van al deze afhankelijkheden mag eigenlijk geen rol spelen op het logische ontwerpniveau. Toch is het mede de verantwoordelijkheid van de logisch ontwerper dat het produkt er op tijd komt. En juist dat op tijd zijn hangt enerzijds af van het feit of het ontwerpproces top - down gericht en goed gestructureerd is en anderzijds van het gebruik van relevante gegevens en parameters. Enig inzicht in wat er speelt op de lagere ontwerp-niveaus is dus essentieel.

Een best case/worse case model of een statistisch model?

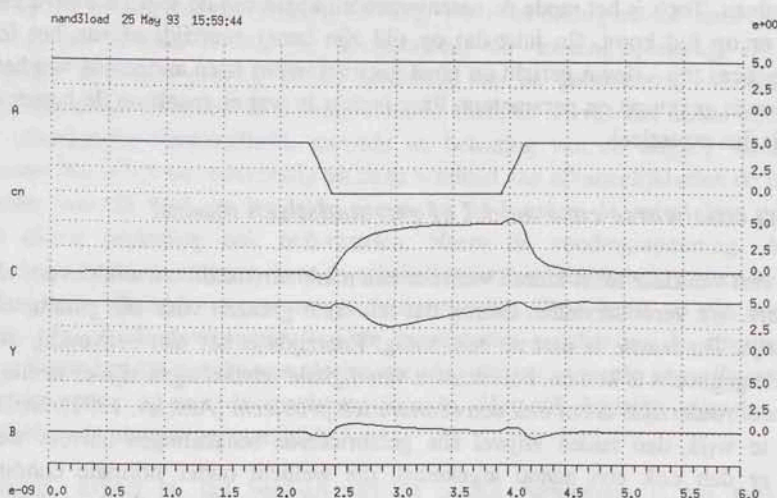
Voor de rest van deze tekst kiezen we voor een minimum/maximum model voor alle timing parameters. We veronderstellen daarbij dat relevante grenzen voor alle parameterwaarden bekend zijn. Dit laatste is niet zo eenvoudig. Enerzijds is het niet eenvoudig om aan de benodigde gegevens te komen. Fabrikanten van digitale schakelingen zijn er in elk geval erg zuinig mee. Anderzijds is het ook een *economisch probleem* [Ambler, 1992]. Stellen we de grenzen te wijd, dan zullen vrijwel alle gefabriceerde schakelingen correct werken. Er worden er dan ook een aantal afgekeurd, die wellicht onder normale condities goed functioneren. Of er is een groot verlies aan prestaties van de schakeling. Een statistisch onderbouwd timing model biedt in deze gevallen uitkomst. Met een statistisch onderbouwd timing model kan bij elke marge een faalkans voor de schakeling berekend worden. Een statistisch model gaat uit van de verdeling van de propagatietijden over een bepaald aantal gemeten exemplaren. Op deze wijze kan de kans bepaald worden dat bijvoorbeeld de propagatietijd van een serieschakeling van een aantal poorten boven een bepaalde grens uitkomt. Een statistisch model gaat uit van de afzonderlijke gemiddelden voor de propagatietijd per poort, sommeert deze en berekent de nieuwe standaardafwijking voor het nieuwe gemiddelde. We komen dan lager uit dan bij de optelling van de minimum waarden en van de maximum waarden. Zo'n statistisch model kan daarom voordelen bieden. Enige voorzichtigheid bij de interpretatie van de resultaten ervan is echter geboden. De statistiek veronderstelt onafhankelijkheid van parameters/variabelen en dat zijn propagatietijden beslist niet.

Modellering van het dynamische gedrag van poorten

In de praktijk komt het vaak voor dat de gegevens van een model geïnterpreteerd worden buiten de grenzen van het model. Dit kan tot verkeerde conclusies leiden. Neem als voorbeeld het logische gedrag van een AND poort. Volgens de regels van de schakelalgebra is de uitgang van een AND poort 0 als tenminste één van de ingangen 0 is. Betekent dit dat, als er een *signaalovergang* op een van de andere ingangen staat, de uitgang dan ook 0 blijft?

Lange tijd is men bij het ontwerpen hier van uitgegaan. Zo wordt een AND vaak toegepast om een signaal tijdelijk te onderdrukken (de default waarde is dan '0'). In CMOS past men

hiervoor pass transistoren of transmission gates toe. Bij de klassieke TTL technologie was dit toegestaan. In zeer snelle logica komt het voor dat snelle signaalovergangen doorspreken naar de uitgang van de poort. Zie figuur 5.39. De AND poort onderdrukt het signaal dan uitsluitend in de statische toestand.



Figuur 5.39. Uitgangsreactie op zeer snelle ingangstransities (simulatieresultaat).

Men kan op twee manieren met dit gegeven omgaan. Wat nonchalant zou men kunnen stellen dat de uitgangspuls te kort duurt om de achterliggende logica te activeren. Het is immers een bekend verschijnsel dat ingangspulsen een bepaalde minimum duur moeten hebben, anders reageert een poort er niet op. Deze conclusie, die zeker gold bij standaard TTL, mag echter niet zonder meer vertaald worden naar elke andere technologie. Maar dit wordt vaak wel gedaan, zonder nader onderzoek. Over deze handelwijze kunnen we kort zijn: het is fout. Denk niet alleen aan de risico's bij een overgang op een andere technologie, maar ook en vooral aan de DC en AC noise margins. Dan is de conclusie duidelijk.

Een ander voorbeeld van hoe het dynamische gedrag kan afwijken van het model is wanneer IC's van verschillende logische families door elkaar heen gebruikt worden. Door onder andere impedantieverschillen kloppen de gespecificeerde propagatietijden niet meer.

5.12 Defecten en foutmodellering

In hoofdstuk 4 hebben we als basis voor het testen van logische schakelingen het stuck-at-0/1 foutmodel geïntroduceerd [Armstrong, 1966]. Dit foutmodel is een *logisch* foutmodel. De onderliggende technologie speelt echter een belangrijke rol bij het testen van schakelingen. Het onderstaande dient als toelichting op de relatie tussen fysische defecten enerzijds en de bijbehorende foutmodellen anderzijds. Aan de hand van enkele voorbeelden

laten we zien dat het stuck-at foutmodel niet voldoende is om een combinatorische schakeling uitputtend te testen. Ook besteden we enige aandacht aan de noodzaak van design-for-testability bij logische schakelingen.

Een defect is een afwijking van de door de ontwerper bedoelde fysische structuur van de schakeling. Op het boardniveau zijn slechte contacten van IC pennen met de koperbaantjes een regelmatig voorkomend defect. Er ontbreekt dan een stukje verbinding. We noemen dit een *open defect*. Een sluiting (*short* of *bridging*) tussen twee sporen is een defect dat uit extra geleidend materiaal bestaat. Dergelijke fouten komen ook in een IC voor. De gevolgen van een defect kunnen velerlei zijn. Het volgende voorbeeld geeft een toelichting.

Voorbeeld

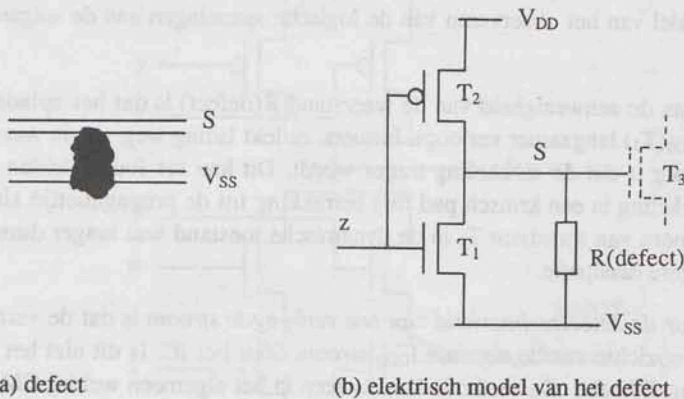
Figuur 5.40 beschrijft een CMOS inverter met een extra weerstand tussen de uitgang van de inverter en de spanning V_{SS} . Deze weerstand wordt veroorzaakt door een stukje materiaal (in dit geval extra polysilicon of metaal), dat daar op de een of andere wijze tijdens de processing is terechtgekomen. Afhankelijk van de grootte van het defect en de plaats ervan zal de weerstand van het defect verschillend zijn. We nemen aan dat de weerstandswaarde kan variëren van

$$R(\text{defect}) \ll R_{\text{doorlaat}}(\text{transistor})$$

tot

$$R(\text{defect}) \gg R_{\text{doorlaat}}(\text{transistor})$$

en gaan na hoe dit defect de werking van deze inverter kan beïnvloeden.



Figuur 5.40. Inverter met geleidend defect.

Als het ingangssignaal z hoog (H) is, dan heeft het defect geen invloed op het logisch niveau van de uitgangsspanning S van de inverter. Immers, transistor T_1 geleidt en als gevolg van het defect zal het laagniveau L van S alleen maar beter kunnen worden. Ook loopt er geen stroom van V_{DD} naar V_{SS} omdat transistor T_2 gesperd is. De uitgangsspanning, de stroom van V_{DD} naar V_{SS} en de dissipatie in de statische toestand zijn normaal.

Geheel anders ligt het als de ingangsspanning z laag (L) is. Dan geleidt transistor T_2 , waardoor er een spanningsdeling optreedt tussen $R_{\text{doorlaat}}(T_2)$ en $R(\text{defect})$. Afhankelijk van de waarde van $R(\text{defect})$ zal de uitgangsspanning in het H gebied kunnen liggen,

$$R(\text{defect}) \gg R_{\text{doorlaat}}(T_2)$$

maar ook in het L gebied,

$$R(\text{defect}) \ll R_{\text{doorlaat}}(T_2)$$

In het eerste geval kan de aanwezigheid van het defect niet aan de uitgangsspanning worden gedetecteerd. In het tweede geval kan dit wel.

Wanneer $R(\text{defect}) \approx R_{\text{doorlaat}}(T_2)$, dan kan de uitgangsspanning zowel H als L zijn en wel afhankelijk van de exacte ligging van de omslagdrempel. Het kan nu voorkomen dat de uitgangsspanning door de volgende component correct herkend wordt als H (bij z is laag). Er zijn echter geen of *onvoldoende ruismarges*. De schakeling is waarschijnlijk niet robuust. Dit laatste kan slechts geconstateerd worden door zeer veel metingen te doen, onder verschillende omstandigheden, teneinde daaruit een indruk te krijgen over de foutkans. Al met al een tijdrovende en kostbare procedure. Bij productietesten doet men dit daarom niet. Een dergelijk defect, dat niet consequent tot het optreden van een fout leidt, noemt men ook wel een *niet-catastrofaal defect* of *soft defect*.

Er is echter een andere manier om dit defect te vinden. Zouden we de stroom door de inverter voldoende nauwkeurig kunnen meten, dan zien we dat bij een laag ingangsniveau op z er in de statische toestand een stroom van V_{DD} naar V_{SS} loopt. Deze stroom, I_{DDQ} genoemd, loopt er normaal in CMOS niet. Het defect is hiermee in principe detecteerbaar, alleen niet door middel van het observeren van de logische spanningen aan de uitgang van de inverter.

Een tweede effect van de aanwezigheid van de weerstand $R(\text{defect})$ is dat het opladen van de gate capaciteit $V_{GS}(T_3)$ langzamer verloopt. Immers, er lekt lading weg via de weerstand $R(\text{defect})$. Het gevolg is dat de schakeling trager wordt. Dit kan tot fouten leiden als de betreffende deelschakeling in een kritisch pad met betrekking tot de propagatietijd zit. Ook zal de omschakelstroom van transistor T_3 in de dynamische toestand wat langer duren. Dit leidt tot een wat hogere dissipatie.

Een voorwaarde voor de detecteerbaarheid van een verhoogde stroom is dat de verhoging relatief groot is ten opzichte van de normale I_{DDQ} stroom door het IC. Is dit niet het geval, dan vervalt deze mogelijkheid. De methode kan daarom in het algemeen wel bij CMOS en niet bij nMOS schakelingen worden toegepast. En ook niet bij bipolaire schakelingen. (Waarom niet?) □

Uit dit voorbeeld volgen enkele belangrijke conclusies.

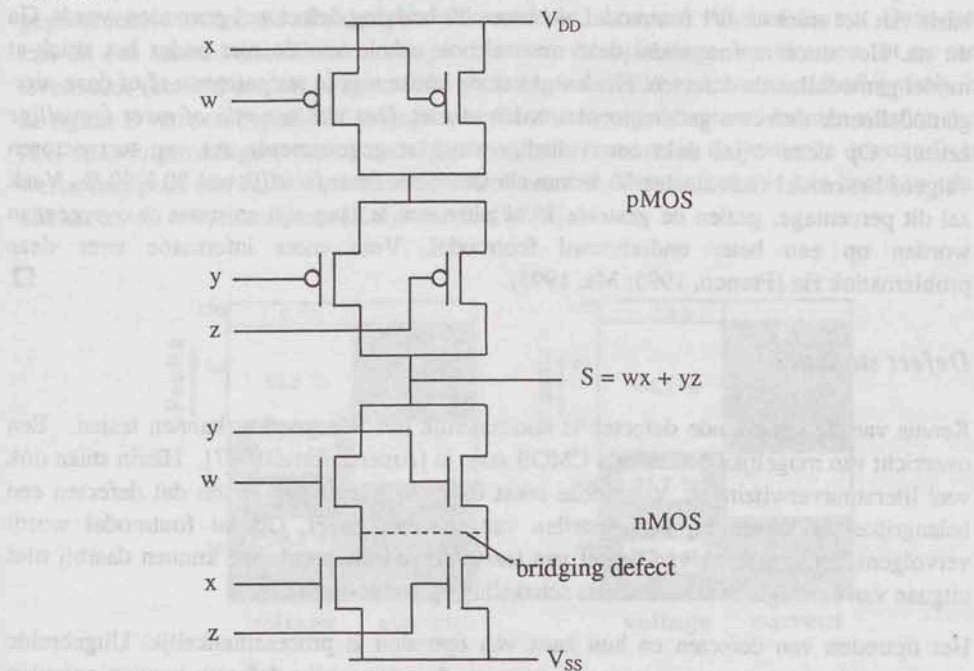
- Functioneel meten (via logische spanningen) dekt niet alle fouten.
- Een defect kan vaak op meer dan één manier gedetecteerd worden.

- Soms is design-for-testability (DFT) een noodzaak. (nMOS $\rightarrow$ CMOS; bijvoorbeeld om een afwijkende I_{DDQ} van een component te kunnen meten mag de I_{DDQ} van de rest van het IC niet te groot zijn.)
- Alle eigenschappen van een IC moeten geverifieerd worden, inclusief dissipatie en robuustheid.
- Robuustheid is nauwelijks op economische wijze verifieerbaar. Fouten die de robuustheid van een IC beïnvloeden moeten op een andere manier geverifieerd worden.

Het testen van IC's, zo volgt uit het vorenstaande, kan niet beperkt blijven tot het meten van logische spanningen. Om een goed en dekkend testvoorschrift op te stellen moet er ingegaan worden op de technologie en op de layout van de schakeling. Daarnaast moet de kans van optreden van elk defect bekend zijn. De basis voor een goede test moet op het technologieniveau gelegd worden.

Het volgende voorbeeld toont aan dat het stuck-at foutmodel niet noodzakelijk alle logisch detecteerbare fouten dekt. Bij testen op basis van het stuck-at-0/1 model worden in principe alleen *logische spanningen* gemeten.

Voorbeeld



Figuur 5.41. Schakeling met een bridging defect.

Figuur 5.41 beschrijft een schakeling met een *bridging defect*, dit is een kortsluiting tussen twee verbindingen op het IC. Als gevolg hiervan verandert de logische formule van het nMOS deel van de schakeling. De logische functie ervan wordt

$$S_{\text{nMOS}}^* = (w + y)(x + z) = wx + yz + \underline{wz + xy} \quad (5.6)$$

waarvan het onderstreepte gedeelte onbedoeld is toegevoegd door het defect. Er treden twee ingangscombinaties op, $w\bar{x}yz$ en $\bar{w}xy\bar{z}$, waarvoor zowel het nMOS deel als het pMOS deel geleidt. Ga dit na. Als nu de weerstand van de pMOS transistoren hoger is dan die van de nMOS transistoren, dan zal de uitgangsspanning bij deze ingangscombinaties door de achterliggende schakeling als het logische niveau L geïnterpreteerd worden. Het bedoelde uitgangsniveau is H. Daarnaast zijn er mogelijk problemen ten aanzien van de DC en AC noise margins.

Gaan we vervolgens met behulp van de twee Karnaughdiagrammen voor de logische functies van het nMOS en het pMOS gedeelte na welke testvectoren toegepast kunnen worden om alle stuck-at-0/1 fouten in $w/t/m/z$ en S te testen, dan blijkt dat er een set van testvectoren mogelijk is, waar de ingangscombinaties $\bar{w}xy\bar{z}$ en $w\bar{x}y\bar{z}$ niet toe behoren. Ga dat na! *Het stuck-at-0/1 foutmodel test dus niet automatisch alle bridging defects.*

Opmerking

Het blijkt in dit voorbeeld echter mogelijk te zijn een set van testvectoren te kiezen, op basis van het stuck-at-0/1 foutmodel, waarmee dit bridging defect wel gevonden wordt. Ga dit na. Het stuck-at foutmodel dekt meestal ook enkele van de niet onder het stuck-at model gemodelleerde defecten. Het hangt van de keuze van de testpatronen af of deze niet-gemodelleerde defecten gedetecteerd worden of niet. *Dus van een min of meer toevallige keuze!* Op deze wijze dekt een volledige stuck-at gegenereerde set van testvectoren volgens het model vaak slechts 30 % van alle defecten, maar *feitelijk* wel 80 à 90 %. Vaak zal dit percentage, gezien de gestelde kwaliteitseisen, te laag zijn en moet er overgegaan worden op een beter onderbouwd foutmodel. Voor meer informatie over deze problematiek zie [Franco, 1995; Ma, 1995]. $\square$

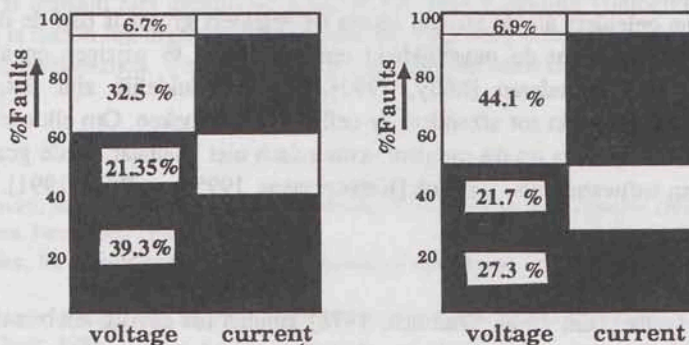
Defect statistics

Kennis van de optredende defecten is noodzakelijk om IC's goed te kunnen testen. Een overzicht van mogelijke defecten in CMOS staat in [Amerasekera, 1987]. Hierin staan ook veel literatuurverwijzingen. Voor deze tekst is het van belang te weten dat defecten een belangrijke rol spelen bij het opstellen van een foutmodel. Op dit foutmodel wordt vervolgens het genereren van de set van testvectoren gebaseerd. We kunnen daarbij **niet** uitgaan van een logisch model van de schakeling op het poortniveau.

Het optreden van defecten en hun kans van optreden is procesafhankelijk. Uitgebreide proces monitoring moet een antwoord geven op de vraag welke defecten kunnen optreden met de bijbehorende kans van optreden van elk defect. Op basis hiervan kan een realistisch foutmodel voor een schakeling opgesteld worden. Bij de fabricage van IC's worden uitgebreide teststructuren toegepast om defecten op te sporen en hun kans van optreden te meten. We gaan er bij de verdere beschouwingen van uit dat deze *defect statistics* bekend zijn.

Om een geproduceerd IC afdoende te kunnen testen moet bekend zijn welke defecten er in de diverse structuren kunnen optreden en wat hun gevolgen voor de werking van het IC zijn. De werking betreft hier niet alleen de logische functie, maar ook alle andere eigenschappen van het IC zoals dissipatie, propagatietijden, levensduur, etc. Hiertoe moeten alle mogelijke defecten op de layout structuur gestrooid worden, om vervolgens met een gedetailleerde simulator te kunnen nagaan wat het effect van elk defect op de werking van het IC is. Dit proces van foutsimulatie is tijdrovend en daarom baseerde men het testen gewoonlijk op stuck-at 0/1 gebaseerde foutmodellen. Met alle consequenties van dien voor de fault coverage!

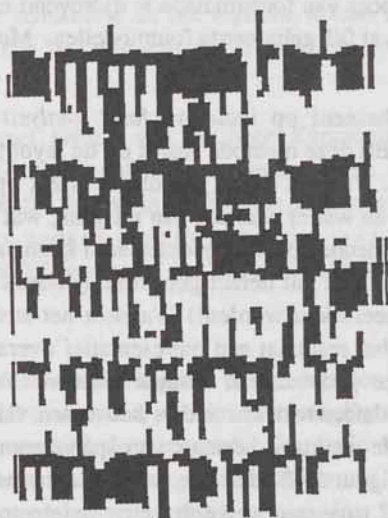
Foutsimulatie wordt vaak gebaseerd op *inductive fault analysis* (IFA) [Dekker, 1988; Ferguson, 1988; Shen, 1985]. Bij deze methode wordt op de layout van het IC random met realistische defecten gestrooid. Van elk defect wordt het effect op de werking nagegaan. Aldus ontstaat een overzicht van wat er allemaal kan misgaan, wat de kans op een bepaald defect is en waaraan de verschillende defecten gedetecteerd kunnen worden. Het *effect van een defect* wordt de *fout (fault)* van dat defect genoemd. (Bedenk dat een defect vaak op meer dan één manier gedetecteerd kan worden!) Wanneer het strooien van defecten lang genoeg geduurd heeft, dan is het resultaat een representatief overzicht van 'alle' mogelijke defecten en weten we hoe ze gedetecteerd kunnen worden. Vervolgens kan een test gegenereerd worden, die alle defecten dekt middels het meten van hun fouten. Deze test kan uit een aantal verschillende metingen bestaan, van spanningen en stromen tot en met levensduur (burn-in testen). Figuur 5.42 geeft een voorbeeld van wat men daarbij vindt. In de figuur is van een bepaald IC nagegaan welke defecten gedetecteerd kunnen worden met H/L spanningsmetingen, welke met I_{DDQ} metingen (stroom in de statische toestand) en welke met geen van beide methoden gevonden worden. Deze informatie kan benut worden om het testen te optimaliseren [Kuijstermans, 1995].



Figuur 5.42. Detecteerbaarheid van defects.

Naarmate computers sneller worden is het mogelijk op de layout van een IC van elk type defect uit te rekenen hoeveel procent van het oppervlak gevoelig is voor dat bepaalde defect. Dit oppervlak heet de *critical area* van dat defect. Zie figuur 5.43. Samen met de uit process monitoring bekende kans op elk defect kan nu de kans op een bepaalde fout

(fault) berekend worden. (Bedenk dat een bepaalde fout zich ook als gevolg van andere defecten kan manifesteren!) Op deze wijze wordt het testen van een geproduceerd IC beter beheersbaar [Pineda de Gyvez, 1993]. Vervolgens kan een economische afweging gemaakt worden tot hoever gegaan moet worden bij het uitvoeren van een test. De praktijk leert dat de eisen met betrekking tot de *fault coverage* steeds hoger komen te liggen.



Figuur 5.43. Voorbeeld van een critical area overzicht [Pineda de Gyvez, 1993].

Tevens kan nu eventueel overgegaan worden tot *layout optimalisatie*. Een defect in de vorm van extra materiaal en met een bepaalde doorsnede heeft in principe geen effect op twee naburige geleiders als de afstand tussen de geleiders groter is dan de doorsnede van dat defect. Vaak bestaat de mogelijkheid om een layout te wijzigen en aldus de *fout-gevoeligheid* te verminderen [Maly, 1990]. Het zal duidelijk zijn dat deze layout optimalisatie zich beperkt tot afzonderlijke cellen in bibliotheken. Om elk ontwerp aldus in zijn geheel te bekijken is op dit moment economisch niet haalbaar, mede gezien het (nog) ontbreken van software tools. Zie ook [Kuijstermans, 1995; van Wijk, 1991].

Sequentiële fouten

Stuck-open faults [Jain, 1983; Wadsack; 1978] kunnen tot gevolg hebben dat er tijdelijk ladingen in een CMOS *combinatorische* schakeling opgesloten worden, waardoor er onder specifieke omstandigheden (tijdelijke) geheugenwerking kan optreden. De schakeling krijgt dan een sequentieel karakter. Dit kan in een combinatorische schakeling nooit de bedoeling zijn. Om dit type fouten te kunnen testen zijn speciale *two-pattern testvectors* nodig. Het achtereenvolgens aanbieden van zo'n two-pattern testvector vraagt extra voorzieningen in een schakeling [Leenstra, 1993]. Alle te testen combinatorische schakelingen moeten op de mogelijkheid van sequentiële fouten onderzocht worden.

In [Sachdev, 1995] wordt melding gemaakt van een historische fout bij het testen van RAM geheugens. De adresdecoder (combinatoriek) van deze geheugens wordt getest door deze af te beelden op de geheugenmatrix [Dekker, 1988]. De decoder wordt impliciet getest via de geheugenmatrix en niet afzonderlijk. Het in de literatuur beschreven bewijs hiervoor is opgesteld voor nMOS RAM's. Deze hadden geen last van de hier beschreven geheugenwerking in de combinatoriek van de decoder. In CMOS schakelingen daarentegen heeft de decoder bij bepaalde defecten wel last van geheugenwerking. De decoder moet dus apart gemodelleerd worden. In praktische gevallen leidt dit tot een verhoging van het aantal benodigde testpatronen met enkele procenten.

Conclusies

- Uit het vorenstaande wordt duidelijk dat het testen van logische schakelingen dient te gebeuren op basis van een defectgebaseerd foutmodel. Een logisch foutmodel, zoals het stuck-at-0/1 foutmodel, dekt lang niet alle defecten. Anderzijds worden bepaalde fouten, waaronder sequentiële fouten, niet gedetecteerd tenzij de testpatronen in een specifieke volgorde worden aangeboden.
- Het opzetten van een defectgebaseerd foutmodel kost veel computertijd. Tevens vereist het een grondig inzicht in de technologie. Dit zijn redenen waarom men in de praktijk nog vaak terugvalt op eenvoudigere foutmodellen. Naarmate er meer software beschikbaar komt voor de analyse van een layout vindt een verschuiving plaats naar defectgebaseerde testpatroongeneratie. De winst in termen van fault coverage en de hiermee samenhangende kwaliteit van een produkt is groot.
- Tevens speelt bij testen de economische haalbaarheid van een produkt een grote rol. Een perfect maar te duur IC wordt niet verkocht.

We besluiten hiermee de discussie over de relatie, die er is tussen defecten en testen. Onze uiteenzetting is bepaald niet uitputtend geweest. Laat het voorlopig voldoende zijn dat de relatie gelegd is tussen het logische ontwerp van de schakeling enerzijds en het testbaar zijn van het ontwerp anderzijds. Voor meer informatie zie [Abramovici, 1990; Rasjuman, 1992; Turino, 1990].

Literatuur

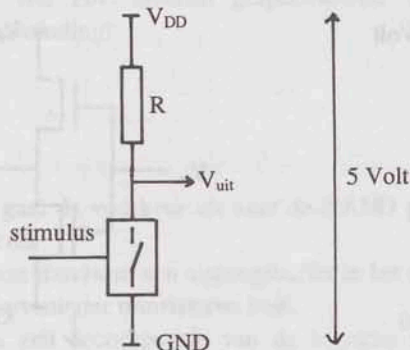
1. M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Testing and Testable Design*, Computer Science Press, New York, 1990.
2. A. P. Ambler, M. Abadir and S. Sastry, *Economics of design and Test*, Ellis Horwood, New York, 1992.
3. E. A. Amerasekera and D. S. Campbell, *Failure Mechanisms in Semiconductor Devices*, John Wiley & Sons, New York, 1987.
4. D. Armstrong, *On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Nets*, IEEE Trans. on Electronic Computers, Vol. EC-15, 1966, pp. 66 - 93.
5. H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley Publ. Comp., Reading, Mass., 1990.
6. F. P. M. Beenker, R. G. Bennetts, A. P. Thijssen, *Testability Concepts for Digital ICs*, Kluwer Academic Publishers, Dordrecht, 1995.
7. J. Benkoski, E. vanden Meersch, L. J. M. Claesen, and H. de Man, *Timing Verification Using Statistically Sensitizable Paths*, IEEE Trans. on CAD, Vol. 9, 1990, pp. 1073 - 1084.

8. D. Brand and S. Iyengar, *Timing Analysis Using Functional Relationships*, IEEE ICCAD 1986 Dig. of Technical Papers, Santa Clara, CA., Nov. 1986, pp. 126 - 130.
9. Z. V. Bundalo and B. L. Dokic, *Three-State CMOS Schmitt-Triggers*, Int. Journal on Electronics, Vol. 66, 1989, pp. 81 - 92.
10. I. Catt, *Crosstalk (Noise) in Digital Systems*, IEEE Trans. on Electronic Computers, Vol. EC-16, 1967, pp. 743 - 763.
11. R. Dekker, F. Beenker and L. Thijssen, *Fault Modeling and Test Algorithm Development for Static Random Access Memories*, Proc. IEEE Int. Test Conf, Washington, 1988, pp. 343 - 352.
12. S. W. Director and W. Maly, *A Statistical Approach to VLSI*, North Holland, 1994.
13. P. Franco, W. D. Farwell, R. L. Stokes, E. J. McCluskey, *An Experimental Chip to Evaluate Test Techniques Chip and Experiment Design*, Proc. ITC Int. Test conf., 1995, pp. 653 - 662.
14. F. J. Ferguson, J. P. Shen, *Extraction and Simulation of Realistic CMOS Faults using Inductive Fault Analysis*, Proc. Int. Test Conf, Washington, 1988, pp. 475 - 484.
15. R. Funk, *Interfacing HC/HCT QMOS Logic with other Families and Various Types of Loads*, RCA Application Note ICAN-7325, in RCA High-Speed CMOS Logic ICs, SSD-290C, 1986.
16. R. Funk, *Power Supply Distribution and Decoupling for QMOS High-Speed Logic ICs*, RCA Application Note ICAN-7329, in RCA High-Speed CMOS Logic ICs, SSD-290C, 1986.
17. J. J. Goedbloed, *Elektromagnetische Compatibiliteit: Analyse en Onderdrukking van Stoorproblemen*, Kluwer, Dordrecht, 1991.
18. W. Hall, *Avoid Confusion in Choosing Digital Logic*, Elektronik Design, 1991, pp. 63 - 76.
19. K. Hart and A. Slob, *Integrated Injection Logic, a New Approach to LSI*, IEEE J. of Solid State Circuits, Vol. SC-7, 1972, pp. 346 - 351.
20. Havasy and M. Kutzin, *Interfacing COS/MOS with other Logic Families*, RCA Application Note ICAN-6602, in RCA High-Speed CMOS Logic ICs, SSD-290C, 1986.
21. N. Hedenstierna and K. O. Jeppson, *CMOS Circuit Speed and Buffer Optimization*, IEEE Trans. on CAD, Vol. CAD-6, 1987, pp. 270 - 281.
22. C. F. Hill, *Noise Margins and Noise Immunity in Logic Circuits*, Microelectronics, 1986, pp. 16 - 21.
23. D. B. Gustavson and J. Theus, *Wire-OR Logic on Transmission Lines*, IEEE Micro, June 1983, pp. 51 - 55.
24. R. C. Jaeger, *Introduction to Microelectronic Fabrication*, Addison-Wesley Publ. Comp., Reading, Mass., 1993.
25. S. K. Jain, and V. D. Agrawal, *Test Generation for MOS Circuits Using D Algorithm*, Proc. 20th Design Automation Conference, 1983, pp. 65 - 70.
26. H. W. Johnson and M. Graham, *High-Speed Digital design, a Handbook of Black Magic*, Prentice-Hall, Englewood Cliffs, N.Y., 1993.
27. F. C. M. Kuijstermans, A. P. Thijssen, M. Sachdev, *Defect Oriented Test Methodology for Complex Mixed-Signal Circuits*, Proc. IEEE 1995 European Design and Test Conf., Paris 1995, pp. 18 - 23.
28. P. Larsson and C. Svensson, *Impact of Clock Slope on True Single Phase Clocked CMOS Circuits*, IEEE J. of SSC., Vol. 29, 1994, pp. 723 - 726.
29. J. A. J. Leijten, J. van Meerbergen, and J. A. G. Jess, *Analysis and Reduction of Glitches in Synchronous Networks*, Proc. ETC European Design and Test Conf., Paris, 1995, pp. 398 - 403.
30. J. Leenstra, *Hierarchical Test Development and Design-for-Testability for (A)synchronous ASICs*, PhD Thesis, Eindhoven, 1993.
31. S. C. Ma, P. Franco and E. J. McCluskey, *An Experimental Chip to Evaluate Test Techniques, Experimental Results*, Proc. ITC Int. Test Conf., 1995, pp. 663 - 672.
32. W. Maly, *Computer-Aided Design for VLSI Circuit Manufacturability*, Proc. IEEE, Vol. 78, Febr. 1990, pp. 356 - 392.
33. J. Mavor, M. A. Jack and P. B. Denyer, *Introduction to MOS LSI Design*, Addison-Wesley, Reading, Mass., 1983.
34. E. J. McCluskey, *Logic Design Principles*, Prentice-Hall Inc, Englewood Cliffs, N.J., 1986.
35. N. P. van der Meys, *Accurate and Efficient Layout Extraction*, Thesis Delft Univ. of Technology, Delft 1992.
36. R. S. Müller and T. L. Kamins, *Device Electronics for Integrated Circuits*, John Wiley and Sons, 1977.
37. R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, *Decomposition of Logic Functions for Minimum Transition Activity*, Proc. ETC Design and Test Conference, Paris, 1995, pp. 404 - 410.
38. National Semiconductor, Application Note 640, Dec. 1989.

39. H. W. Ott, *Noise Reduction Techniques in Electronic Systems*, John Wiley, New York, 1976.
40. Philips Components and Materials, *Designer's Guide High-Speed CMOS*, 1986.
41. Philips Semiconductors, *Concise Catalogue*, 1995.
42. J. Pineda de Gyvez, *Integrated Circuit Defect-Sensitivity: Theory and Computational Models*, Kluwer Academic Publishers, Boston, 1993.
43. R. Rasjuman, *Digital Hardware Testing, Transistor-Level Fault Modeling and Testing*, Artech House, Boston, 1992.
44. M. J. Riezenman, *Wanlass's CMOS Circuit*, IEEE Spectrum, May 1991, p. 44.
45. M. Sachdev, *Test and Testability Techniques for open Defects in RAM Address Decoders*, to be published.
46. D. Shear, *EDN's Advanced CMOS Logic Ground-Bounce Tests*, EDN, March 2, 1989.
47. J. P. Shen, W. Maly and F. J. Ferguson, *Inductive Fault Analysis of MOS Integrated Circuits*, IEEE Design and Test, December 1985, pp. 13 - 26.
48. M. Shoji, *CMOS Digital Circuit Technology*, Prentice-Hall Int., Englewood Cliffs, N.J. 1988.
49. A. P. Thijssen, H. A. Vink, C. H. Eversdijk, *Digitale Techniek Deel 1*, Delftse Universitaire Pers, 1994.
50. J. Turino, *Design to Test*, van Nostrand Reinhold, New York, 1990.
51. H. J. M. Veendrick, *Geïntegreerde MOS-schakelingen: Een inleiding tot VLSI en ASIC's*, Delta Press, Amerongen, 1990.
52. F. M. Wanlass and C. T. Sah, *Nanowatt Logic Using Field-Effect Metal-Oxide Semiconductor Triodes*, Proc. Solid State Circuits Conf., Febr. 1963, pp. 32 - 33.
53. N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, a Systems Perspective*, Addison-Wesley, Englewood Cliffs, N.J., 1993.
54. T. Williams, *EMC for Product Designers: Meeting the European Directive*, Butterworth-Heinemann, 1992.
55. C. Wissenburgh, *Inleiding in de Elektronica, een Ontwerpgerichte Benadering*, Delftse Universitaire Pers, Delft 1995.
56. L. van Wijk, *Defect Oriented Testing, a Method to Verify the Accuracy of Fault Models*, Masters Thesis, TU Delft, Fac. ET, 1991.

Opgaven hoofdstuk 5

5.1.



De schakelaar heeft R_{ON}/R_{OFF} weerstanden van $10/100 \Omega$. De grootste logische slag (het verschil tussen $V_{uit}(H)$ en $V_{uit}(L)$) is bij een zo gunstig mogelijk gekozen weerstand R circa

- a. 2,41 Volt.
- b. 2,48 Volt.
- c. 2,60 Volt.
- d. 4,00 Volt. (Zie opmerking z.o.z.)

(Er hoeft geen rekening gehouden te worden met de belasting van de uitgang door ingangen van de erdoor ingestelde componenten.)

5.2. Zie voor de figuur opgave 5.1.

De schakelaar in de figuur heeft een bepaalde verhouding van de R_{ON}/R_{OFF} weerstanden. Bij welke waarde van de weerstand R is de logische slag (het verschil tussen $V_{uit(H)}$ en $V_{uit(L)}$) maximaal, rekening houdend met een belasting R_L van de uitgang?

a. $R = R_L \cdot \sqrt{\frac{R_{ON} \cdot R_{OFF}}{(R_{ON} + R_L)(R_{OFF} + R_L)}}$.

b. $R = R_L \cdot \sqrt{\frac{R_{ON} + R_{OFF}}{(R_{ON} + R_L)(R_{OFF} + R_L)}}$.

c. $R = R_L \cdot \sqrt{R_{ON} + R_{OFF}}$.

d. R heeft een andere waarde.

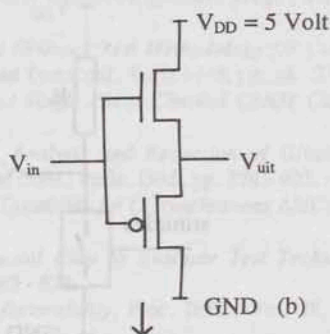
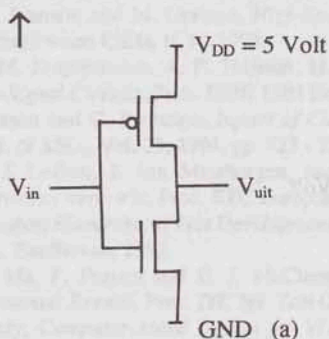
5.3. Van de transistoren in de figuur (a) hieronder is de R_{ON}/R_{OFF} weerstand bekend. De transistoren zijn voor wat betreft de weerstandswaarden identiek. De maximale logische slag (het verschil tussen $V_{uit(H)}$ en $V_{uit(L)}$) die bereikt kan worden, afhankelijk van de waarde van de belastingsweerstand R_L , is:

a. $\frac{R_{OFF} - R_{ON}}{R_{OFF} + R_{ON}} \cdot 5 \text{ Volt}$ bij $R_L \rightarrow \infty$.

b. $\frac{R_{OFF} - R_{ON}}{R_{OFF} + R_{ON}} \cdot 5 \text{ Volt}$ bij $R_L = \frac{1}{2}(R_{OFF} + R_{ON})$.

c. 5 Volt bij $R_L \rightarrow \infty$.

d. Geen van de gegeven waarden is correct.



5.4. Over de gegeven MOS transistorschakelingen in de figuur hierboven worden enkele beweringen gedaan.

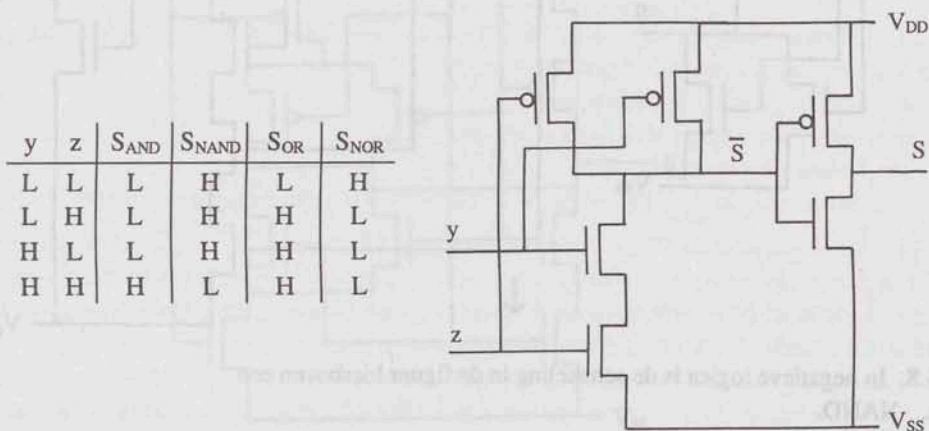
- Schakeling (a) is een CMOS invertor.
- In schakeling (a) is de stroom in de statische toestand ($V_{in} \in H/L$) groter dan de stroom in de dynamische toestand.

- Binnen de gangbare afspraken voor CMOS schakelingen kan ook schakeling (b) zonder meer als inverter worden toegepast.
- Schakeling (b) is de meest toegepaste schakeling voor een CMOS inverter.

Van deze afspraken is/zijn er

- een juist.
- twee juist.
- drie juist.
- vier juist.

5.5. Het volgende schema beschrijft een poortschakeling in CMOS.



Tevens zijn er vier H/L tabellen gespecificeerd. Welke H/L tabel behoort bij de gespecificeerde schakeling?

- S_{AND} .
- S_{NAND} .
- S_{OR} .
- S_{NOR} .

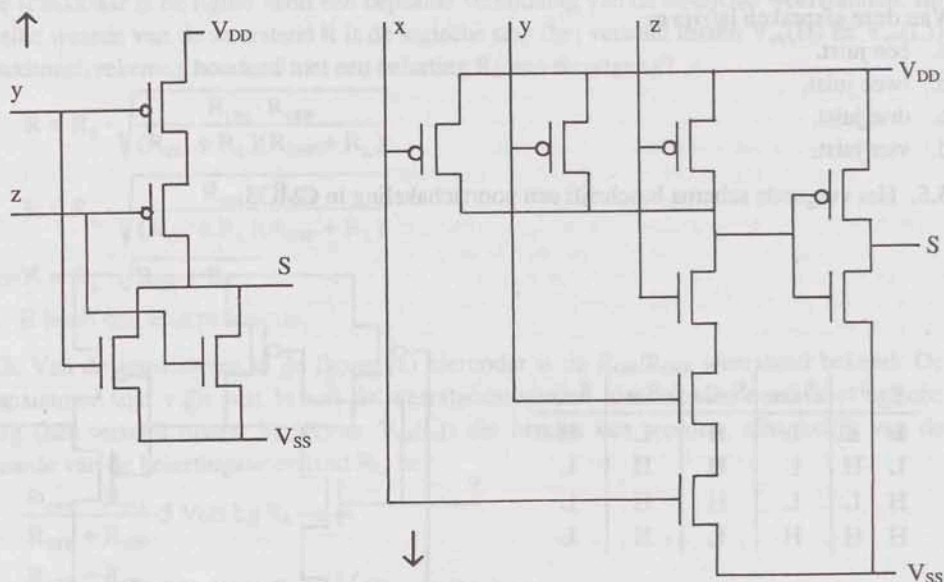
5.6. In CMOS gaat de voorkeur uit naar de NAND poort boven de AND poort en wel vanwege het feit dat

- de NAND poort standaard een uitgangsbuffer in het schema heeft zitten.
- de NAND poort minder transistoren kost.
- met NAND's een decompositie van de logische functie in een multilevel realisatie gemakkelijker gaat.
- Geen van de reeds gegeven antwoorden is goed. De voorkeur gaat in het algemeen uit naar de AND poort, vanwege de hogere schakelsnelheid.

5.7. Het transistorschema in de figuur hierna beschrijft een poortschakeling in CMOS. Afhankelijk van de gemaakte afspraak om positieve respectievelijk negatieve logica toe te passen is deze schakeling een

- NAND of een AND poort.

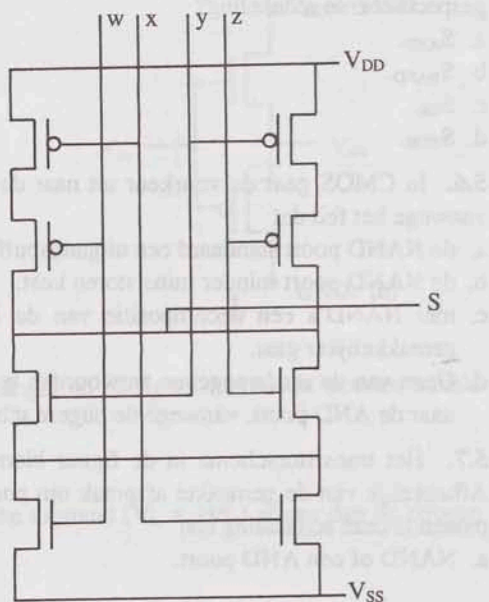
- b. NOR of een NAND poort.
 c. AND of een OR poort.
 d. OR of een NOR poort.



5.8. In negatieve logica is de schakeling in de figuur hierboven een

- a. NAND.
 b. NOR.
 c. AND.
 d. OR.

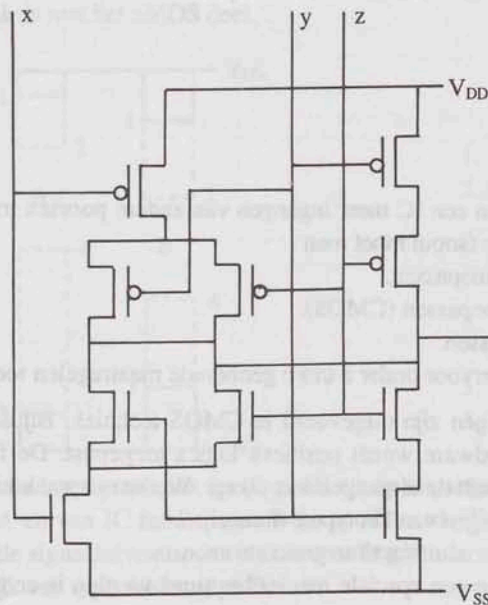
5.9. Zie de figuur hiernaast.



De logische functie (in positieve logica) van de getekende schakeling is

- a. $S = (w + x)(y + z).$
- b. $S = wx + yz.$
- c. $S = \overline{(w + x)(y + z)}.$
- d. $S = wx + yz.$

5.10.



De logische functie (in positieve logica) van de getekende schakeling is

- a. $S = \overline{xy + xz + yz}.$
- b. $S = xy + xz + yz.$
- c. $S = (x + y)(x + z)(y + z).$
- d. S heeft een andere formule.

5.11. In het Karnaughdiagram is de logische functie S gespecificeerd.

		z				S
		1	1	1	1	
y	1	1	0	0	0	
	0	1	1	1	1	
		x				

Inverse variabelen zijn niet beschikbaar. Realiseer de logische functie S in full CMOS (positieve logica). Deze schakeling kost daarin minimaal

- a. zes transistoren.
- b. acht transistoren.
- c. 12 transistoren.
- d. meer dan 12 transistoren.

5.12. Over pass transistoren en transmission gates worden enkele beweringen gedaan.

- Een pass transistor laat een logisch niveau beter door dan een transmission gate.
- De toepassing van transmission gates bespaart transistoren, vergeleken met pass transistoren.
- Met pass transistoren kan gemakkelijk de functie “selector” worden gerealiseerd.

Hiervan is/zijn er

- a. geen juist.
- b. een juist.
- c. twee juist.
- d. drie juist.

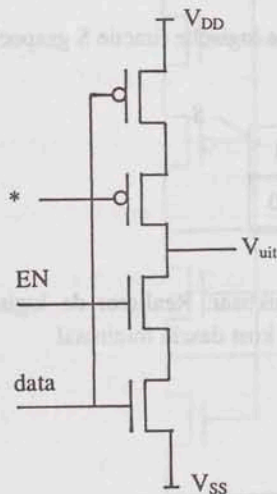
5.13. Indien een uitgang van een IC meer ingangen van andere poorten moet instellen dan de gespecificeerde maximale fanout moet men

- a. een extra uitgangsbuffer toepassen.
- b. een open-drain uitgang toepassen (CMOS).
- c. een 3-state uitgang toepassen.
- d. naar keuze een van de hiervoor onder a t/m c genoemde maatregelen toepassen.

5.14. Twee poortschakelingen zijn uitgevoerd in CMOS techniek. Bij de afbeelding van hun logische functie op hardware wordt positieve logica toegepast. De functies S_0 en S_1 zijn verschillend. Men verbindt de uitgangen met elkaar. Welke uitspraak hierover is correct indien de te verbinden uitgangen van het type 3-state zijn?

- a. Er is geen gevaar voor kortsluiting of te grote stromen.
- b. Indien de schakelingen op een speciale manier bestuurd worden is er geen gevaar voor kortsluiting.
- c. Uitgangen van dit type mogen nooit met elkaar verbonden worden, vanwege het gevaar van kortsluiting.
- d. De met elkaar verbonden uitgangen leveren in positieve logica de “wired-OR” functie van S_0 en S_1 op.

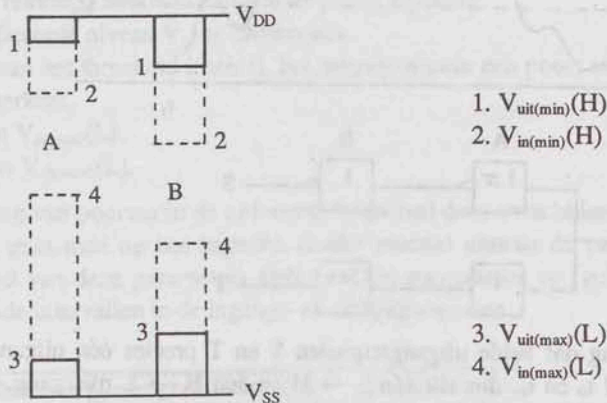
5.15.



Om van de getekende schakeling een invertor/negator met 3-state uitgang te maken moet op het punt * het volgende signaal worden aangesloten:

- EN.
- $\overline{\text{EN}}$.
- EN-data.
- Geen van de antwoorden onder a t/m c is juist omdat het pMOS deel dan niet complementair is met het nMOS deel.

5.16.



In de figuur zijn de ingangsniveaus (gestippeld) en de uitgangsniveaus (getrokken lijn) van IC familie type A en van IC familie type B gespecificeerd. Wanneer we alleen kijken naar de gespecificeerde signaalniveaus, dan is data-overdracht

- alleen mogelijk van type A naar type B.
- alleen mogelijk van type B naar type A.
- mogelijk van type A naar type B en ook omgekeerd.
- tussen IC's van het type A en het type B niet mogelijk.

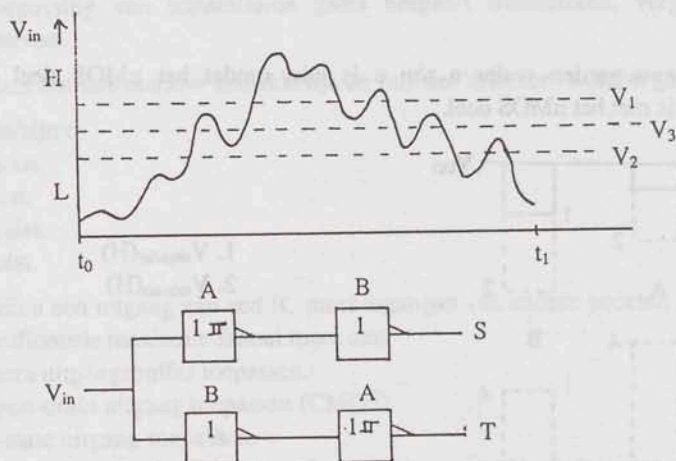
5.17. Voor betrouwbare data-overdracht tussen verschillende types IC's is het nodig dat

- voldaan wordt aan de gespecificeerde logische niveaus;
- zekere (DC en AC) noise margins overblijven.

We noemen twee verschillende IC's van het type A en het type B compatibel (ten aanzien van de onderzochte aspecten) als de logische niveaus kloppen en de overblijvende DC noise margins niet kleiner zijn dan de kleinste marge, die optreedt bij data-overdracht tussen IC's van hetzelfde type onderling. Bij de specificatie van de logische niveaus volgens de figuur van opgave 5.16. is betrouwbare data-overdracht, wanneer we uitsluitend letten op de geformuleerde eisen ten aanzien van de logische niveaus en de gespecificeerde DC noise margins, mogelijk van

- type A naar type B.
- type B naar type A.
- type A naar type B en omgekeerd type B naar type A.
- Er is geen betrouwbare data-overdracht mogelijk omdat in beide richtingen één of meer van de DC noise margins te klein wordt.

5.18. Gegeven zijn twee types invertoren. Type A heeft een Schmitt-trigger ingang met drempelspanningen V_1 en V_2 . Type B heeft een normale ingang met drempelspanning V_3 . In het interval tussen t_0 en t_1 heeft het ingangssignaal V_{in} onderstaand verloop.



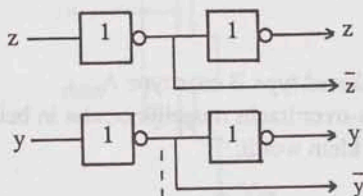
Het is de bedoeling dat beide uitgangssignalen S en T precies één uitgangspuls afgeven tijdens het interval t_0 en t_1 , dus elk één $L \rightarrow H$ en één $H \rightarrow L$ overgang. Ga na wat het feitelijke verloop van S en van T is. Dan geldt:

- S is fout; T is fout.
- S is fout; T is goed.
- S is goed; T is fout.
- S is goed; T is goed.

5.19. De uitspraken a tot en met d betreffen de drempelspanning van een poort en het Hoog respectievelijk Laag niveau van een op een ingang ervan aangeboden signaal. Een van deze uitspraken is fout. Welke?

- De gevoeligheid van poortingangen voor storingen kan verlaagd worden door de Hoog $\rightarrow$ Laag drempel en de Laag $\rightarrow$ Hoog drempel elk bij een hogere spanning te leggen.
- Als de flank van een signaal door omstandigheden minder steil wordt, dan heeft dit tot gevolg dat een door dit signaal bestuurd poort later op een ingangsverandering reageert.
- Een vergroting van de marge tussen het Hoog en het Laag niveau enerzijds en de drempelspanning anderzijds heeft een vergroting van de propagatietijd van het signaal door de poort tot gevolg.
- De ligging van de drempelspanning varieert met de temperatuur. Dit heeft gevolgen voor de feitelijke propagatietijd van de poort.

5.20.



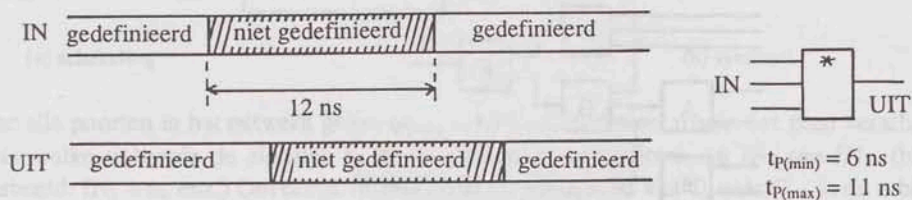
Bij code-omzetters en andere ingangen van IC's vindt men vaak de getekende ingangsconstructie. Men past deze toe om

- het inwendige van de code-omzetter te beschermen tegen externe overspanningen.
- genegeerde signaalwaarden op te wekken
- onder meer om de fanin aan te passen.
- het gebruik van Schmitt-trigger ingangen mogelijk te maken.

5.21. In de praktijk zijn de aanbevolen referentieniveaus voor de timing parameters van een logische poort, rekening houdend met alle relevante aspecten,

- het gespecificeerde niveau V_{ref} in databoeken.
- de grenzen van het threshold interval, het interval waarin een poort een $H \rightarrow L$ of $L \rightarrow H$ transitie herkent.
- $V_{in(min)}(H)$ en $V_{in(max)}(L)$.
- $V_{uit(min)}(H)$ en $V_{uit(max)}(L)$.

5.22. Het gedrag van poorten in de tijd wordt beïnvloed door verschillende parameters. Bij het ontwerpen gaat men op het logische niveau meestal uit van de parameters $t_{p(min)}$ en $t_{p(max)}$. Op grond van deze parameters stelt men het poortmodel op, gebaseerd op wel of niet gedefinieerde intervallen in de ingangs- en uitgangssignalen.



Bij een niet-gedefinieerd interval van 12 ns in hetingangssignaal IN van de poort heeft het niet-gedefinieerde interval aan de uitgang UIT van de poort de volgende duur:

- 12 ns.
- 17 ns.
- 18 ns.
- 23 ns.

5.23. De parameter $t_{p(min)}(\text{poort})$ is als volgt gedefinieerd.

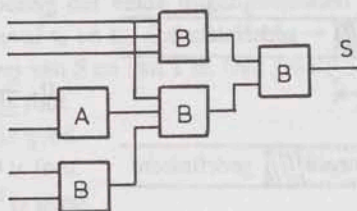
- De parameter $t_{p(min)}(\text{poort})$ legt de minimum propagatietijd van een poort vast.
- De parameter $t_{p(min)}(\text{poort})$ legt de tijd vast tussen het verlaten van het gedefinieerde niveau van hetingangssignaal enerzijds en het niet meer gedefinieerd zijn van het uitgangssignaal anderzijds en dat gemeten onder de meest gunstige (= snelle) schakelcondities.
- Idem, maar dan voor het weer gedefinieerd zijn van hetingangssignaal en het uitgangssignaal.
- Geen van deze omschrijvingen is juist. Het is de minimum van de tijden die onder b of c gevonden worden.

5.24. Het drempelgebied van een poortingang (het threshold interval) geeft aan tussen welke spanningsniveaus het niveau ligt waarop de poort daadwerkelijk op een $H \rightarrow L$ of

een $L \rightarrow H$ overgang reageert. Het threshold interval introduceert enige onzekerheid ten aanzien van het tijdstip, waarop de poort op een ingangsverandering zal gaan reageren. Hierover worden vier beweringen gedaan, waarvan er een juist is. Welke?

- De bijdrage aan de niet-gedefinieerde gebieden in het signaalmodel van de poort, voor zover geïntroduceerd door het threshold interval, is afhankelijk van de flanksteilheid van het sturende signaal. De flanksteilheid wordt daartoe gemeten tussen 10 % en 90 % van het verschil tussen de H en L niveaus.
- De bijdrage aan de niet-gedefinieerde gebieden in het signaalmodel van de poort, voor zover geïntroduceerd door het threshold interval, is afhankelijk van de flanksteilheid van het sturende signaal. De flanksteilheid moet daartoe gemeten worden tussen de aangegeven grenzen van het threshold interval.
- Bij de gegeven grenzen van het threshold interval heeft de bijdrage aan de niet-gedefinieerde gebieden in het signaalmodel van de poort een vaste waarde.
- Geen van de vorige antwoorden is juist. Het threshold interval heeft in principe geen invloed op de duur van de niet-gedefinieerde gebieden in het signaalmodel van de poort. Wel heeft het threshold interval invloed op de grootte van de AC/DC noise margins.

5.25. De gegeven schakeling is opgebouwd uit poorten van het type A en van het type B.



De timing parameters zijn:

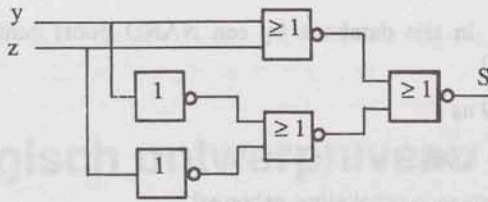
type A: $t_{p(\min)}(\text{poort}) = 10 \text{ ns}$; $t_{p(\max)}(\text{poort}) = 20 \text{ ns}$.

type B: $t_{p(\min)}(\text{poort}) = 5 \text{ ns}$; $t_{p(\max)}(\text{poort}) = 15 \text{ ns}$.

De ingangssignalen zijn stabiel tot het tijdstip t_0 en kunnen daarna gedurende 30 ns veranderen. Hierna hebben ze weer een gedefinieerde logische waarde. (Het niet-gedefinieerde gebied in het ingangssignaal loopt dus van t_0 tot $t_0 + 30 \text{ ns}$.) Het niet-gedefinieerde gebied in het uitgangssignaal loopt van

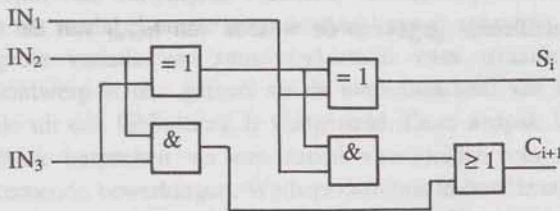
- t_0 tot $t_0 + 60 \text{ ns}$.
- t_0 tot $t_0 + 90 \text{ ns}$.
- $t_0 + 10$ tot $t_0 + 90 \text{ ns}$.
- Het niet-gedefinieerde interval is anders dan onder a t/m c is aangegeven.

5.26. Van de gegeven schakeling is de propagatietijd van alle poorten 10 ns. Op het tijdstip t_0 hebben de ingangssignalen y en z de waarde 0. Op het tijdstip t_1 veranderen y en z tegelijk van 0 in 1. Welk van onderstaande uitspraken heeft betrekking op het gedrag van het uitgangssignaal S rond en vlak na t_1 ?

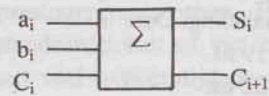


- S is en blijft op de waarde 0.
- S is en blijft op de waarde 1.
- S vertoont na t_1 een 0 - 1 - 0 spike.
- S vertoont na t_1 een 1 - 0 - 1 spike.

5.27. Gegeven is de volgende realisatie van een sectie van een n-bit full adder.



(a) schakeling



(b) symbool

Voor alle poorten in het netwerk geldt: $t_{p(max)} = 15$ ns. Functioneel maakt het geen verschil uit in welke volgorde de signalen a_i , b_i en C_i worden aangesloten op IN_1 t/m IN_3 . (bij voorbeeld: $IN_1 = b_i$, enz.) Om echter de maximale propagatietijd van C_i naar C_{i+1} in de n-bit full adder zo kort mogelijk te houden moet men een aansluiting van de signalen nemen als

IN_1 IN_2 IN_3 . (alle a_i 's en b_i 's veranderen tegelijk)

- a_i b_i C_i .
- C_i a_i b_i .
- a_i C_i b_i .
- De volgorde van de aansluitingen maakt niets uit.

5.28. Spikes (kortdurende overgangen) in combinatorische schakelingen kunnen ontstaan als gevolg van looptijdverschillen in de combinatoriek. Een van de volgende beweringen is niet juist. Welke?

- Bij een goed uitgebalanceerde combinatorische schakeling treden aan de uitgang nooit spikes op.
- Spikes kunnen in CMOS tot een verhoogde dissipatie leiden.
- Voor wat de timing betreft zijn spikes niet hinderlijk, mits ze voorbij zijn voor de maximale propagatietijd van de schakeling verstreken is.
- Spikes kunnen zo kort duren dat de volgende poort er niet op reageert. Poorten discrimineren van nature korte spikes, als de vermogensinhoud van de spike te weinig is om de poort te activeren.

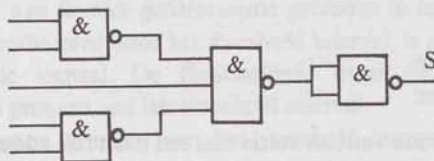
5.29. Bij het bepalen van de propagatietijd van een combinatorische schakeling baseren we ons meestal op het globale tijdmodel, waarbij we per poort rekening houden met $t_{p(min)}$ en

$t_{P(max)}$. Een fabrikant vermeldt in zijn databoek bij een NAND poort echter een meer gedetailleerde specificatie en wel

$$t_{PHL(min)} = 7 \text{ ns} \quad t_{PHL(max)} = 9 \text{ ns}$$

$$t_{PLH(min)} = 12 \text{ ns} \quad t_{PLH(max)} = 15 \text{ ns}$$

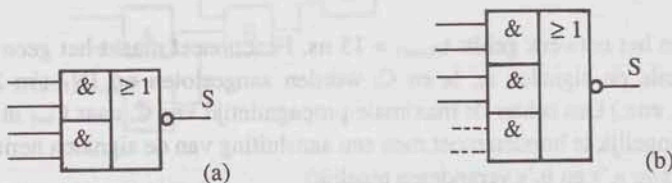
Met deze NAND's wordt onderstaande schakeling gebouwd.



Bepaal met behulp van de gedetailleerde gegevens de waarde van $t_{P(min)}$ van de totale schakeling. Deze is

- 19 ns.
- 22 ns.
- 36 ns.
- 42 ns.

5.30. In figuur a is het ontwerpschema van een te realiseren logische schakeling gegeven.



Figuur b beschrijft een verkrijgbare standaardbouwsteen, waarop het ontwerp van figuur a past. Daarbij is een AND poort van figuur b overbodig. Welk van onderstaande antwoorden is op deze situatie van toepassing?

- De ingangen van de overbodige AND poort behoeven niet te worden aangesloten.
- Op alle ingangen van de overbodige AND poort moeten enen gezet worden
- Op tenminste een ingang van de overbodige AND poort moet een logische 0 gezet worden; op de andere ingang naar keuze een logische 0 of 1.
- Op een ingang van de overbodige AND poort moet een logische 0 gezet worden; op de andere ingang mag een willekeurig signaal gezet worden met een waarde tussen 0 en 5 Volt.

Stelling

Digitale techniek is allesbehalve statisch.

6

Het logisch ontwerpniveau

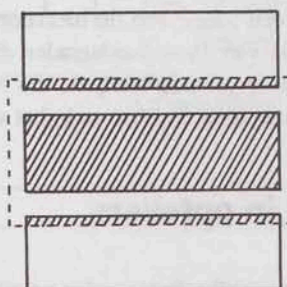
6.1 Standaardcomponenten

Logische schakelingen worden niet altijd en ook niet tot in elk detail tot op het laagste ontwerpniveau ontworpen. Naarmate een (deel)probleem of bewerking vaker optreedt wordt het rendabel om er extra aandacht aan te schenken. In het recente verleden is aldus een grote variatie van *standaardcircuits* voor speciale toepassingen ontworpen. Een (deel)ontwerp is dan gereed als de toepasbaarheid van een standaardcircuit of van een module uit een bibliotheek is vastgesteld. Deze aanpak bespaart veel ontwerptijd. In dit hoofdstuk bespreken we een aantal voorbeelden van standaardoplossingen voor veel voorkomende bewerkingen. We beperken ons in hoofdzaak tot het logisch ontwerpniveau. Onderstaand schema geeft de plaats van dit hoofdstuk in het ontwerptraject weer.

structureel ontwerpniveau

logisch ontwerpniveau

fysisch ontwerpniveau



scope hoofdstuk

Het aantal *vrijheidsgraden* om tot een definitief ontwerp te komen is als regel groot. De ontwerper moet uit de vele mogelijkheden een oplossing kiezen die het best voldoet aan de gestelde eisen en randvoorwaarden. Het onderzoek van alle mogelijkheden vraagt veel tijd, hetgeen in de kostprijs van het eindprodukt tot uitdrukking komt. In de praktijk staat men dan ook vaak voor het dilemma vroegtijdig te moeten beslissen, op basis van een snelle evaluatie van de mogelijkheden. Veel oplossingen blijken dan in een later stadium suboptimaal te zijn. Het is duidelijk dat de toepassing van standaardcircuits (bibliotheek) hierop een gunstige invloed heeft, zeker bij prototypes of kleine series van een IC. (Bij grote series domineren de produktiekosten van het IC over de ontwerpkosten. Daar liggen de criteria voor optimalisatie dus anders.)

De functie van een standaardcircuit moet goed gedefinieerd zijn. De toepassing ervan moet de ontwerper zo weinig mogelijk beperkingen opleggen. Vaak treft men daarom aanpassingen aan om het circuit meer universeel toepasbaar te maken. Voorbeelden zijn

ingangsbuffers om de fanin te standaardiseren. Ook zijn er voorzieningen nodig om de toepasbaarheid van componenten in een groter geheel te vergemakkelijken. Hierbij denken we bijvoorbeeld aan 3-state of open-drain uitgangen. Tevens is er een technologie-afhankelijke voorkeur voor bepaalde logische operatoren.

Een probleem bij het gebruik van standaardcircuits is dat er voor een bepaalde toepassing vaak te veel 'functionaliteit' in de componenten zit. Er zijn dan extra instelsignalen nodig om de gewenste deelfunctie(s) te selecteren. Dit vergt een grotere besturing en/of extra oppervlak. Men heeft dit probleem in eerste instantie opgelost door verschillende varianten van standaardcircuits uit te brengen of bibliotheken met standaardfuncties uit te breiden met de nodige varianten. Daarmee kan de overhead gereduceerd worden. Aldus is een enorme collectie componenten ontstaan.

De huidige generatie van silicon compilers (software om schakelingen af te beelden op hardware) is dermate krachtig dat de ontwerper er in eerste instantie mee kan volstaan het ontwerp in software functioneel te beschrijven. De hardware beschrijvingstaal VHDL vervult hierbij een belangrijke rol. In de gegenereerde lay-out van de hardware vinden we de afzonderlijke deelfuncties vaak niet meer terug. Ook componenten met een 'fysische' functie zoals buffers kunnen naar behoefte automatisch worden toegevoegd en gedimensioneerd. Slechts bij 'high-performance' schakelingen moet er nog in detail tot op de lagere ontwerp-niveaus worden ontworpen.

In het volgende nemen we enkele voorbeelden door van het ontwerp van schakelingen voor standaardfuncties. Daarbij zullen een aantal van de hiervoor genoemde aspecten de revue passeren, waaronder een overzicht van de vrijheidsgraden die men bij een ontwerp heeft. De meer structurele aspecten van de probleemaanpak, die veelal direct boven het logisch ontwerp-niveau gesitueerd moeten worden, lichten we in het volgende hoofdstuk toe.

6.2 Carrypropagatie in optellers

In paragraaf 4.2 is afgeleid dat een opteller kan worden opgebouwd uit een rij van identieke cellen, die we full adders genoemd hebben. Elke full adder genereert een sombit en een carrybit. Een optelling is gereed als alle carrybits in de sombits verwerkt zijn. De logische werking van één full adder is beschreven met de logische formules

$$S_i = \bar{a}_i \bar{b}_i C_i + \bar{a}_i b_i \bar{C}_i + a_i \bar{b}_i \bar{C}_i + a_i b_i C_i \quad (6.1)$$

voor het sombit en

$$C_{i+1} = a_i b_i + (a_i + b_i) \cdot C_i \quad (6.2)$$

voor het carrybit.

Carry generate en carry propagate hulpfuncties

In formule (6.2) geeft de term $a_i b_i$ aan dat de betreffende full adder een uitgaande carry $C_{i+1} = 1$ genereert op basis van de aangeboden databits a_i en b_i en wel onafhankelijk van

de waarde van de inkomende carry C_i . De term $(a_i + b_i)$ geeft aan dat een inkomende carry C_i wordt doorgegeven aan de uitgaande carry C_{i+1} . Formule (6.2) wordt daarom wel geschreven als

$$C_{i+1} = G_i + P_i \cdot C_i \quad (6.3)$$

↑ carry propagate functie

↑ carry generate functie

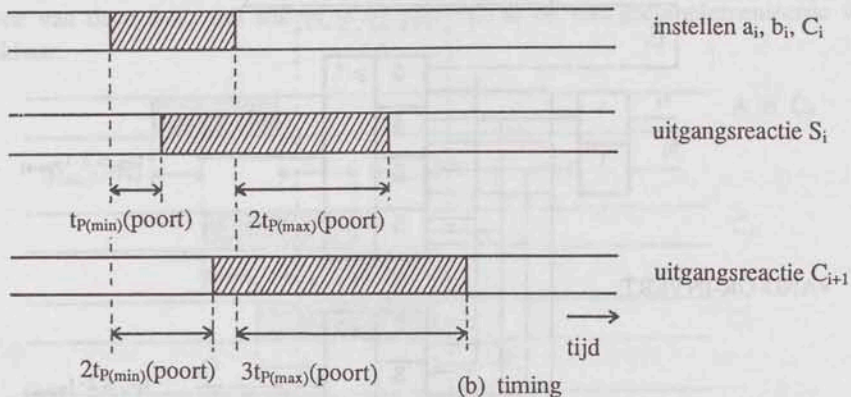
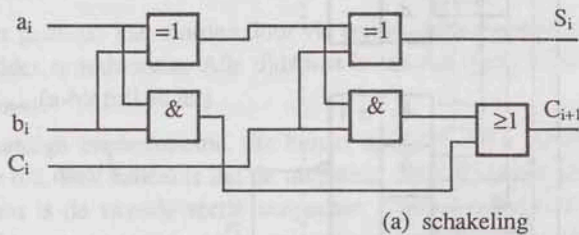
waarbij twee hulpfuncties, de *carry generate functie* G_i

$$G_i = a_i b_i \quad (6.4)$$

en de *carry propagate functie* P_i

$$P_i = a_i + b_i \quad (6.5)$$

zijn geïntroduceerd. Deze *hulpfuncties* passen we later in dit hoofdstuk vaak toe. (Merk op dat voor P_i ook de formule $P_i = a_i \oplus b_i$ genomen mag worden. Ga na waarom!)

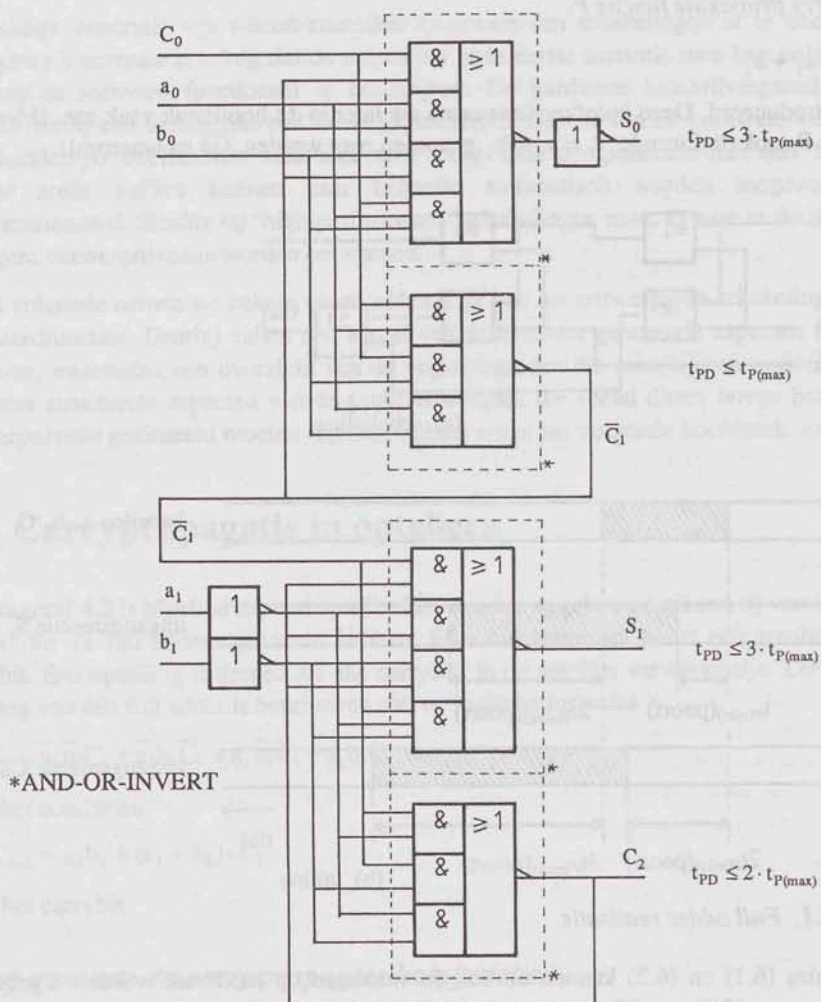


Figuur 6.1. Full adder realisatie.

De formules (6.1) en (6.2) kunnen op velerlei manieren op hardware worden afgebeeld. Een ervan, op AND en OR poorten, is in figuur 4.3 toegelicht. De realisatie met een *minimum aantal poorten* in figuur 6.1 volgt uit

$$\begin{aligned}
 S_i &= \bar{a}_i \bar{b}_i C_i + \bar{a}_i b_i \bar{C}_i + a_i \bar{b}_i \bar{C}_i + a_i b_i C_i \\
 &= (\bar{a}_i \bar{b}_i + a_i b_i) \cdot C_i + (\bar{a}_i b_i + a_i \bar{b}_i) \cdot \bar{C}_i \\
 &= (\bar{a}_i \oplus \bar{b}_i) \cdot C_i + (a_i \oplus b_i) \cdot \bar{C}_i \\
 &= (a_i \oplus b_i) \oplus C_i
 \end{aligned} \tag{6.6}$$

$$\begin{aligned}
 C_{i+1} &= a_i b_i + a_i C_i + b_i C_i \\
 &= a_i b_i + a_i \bar{b}_i C_i + \bar{a}_i b_i C_i \\
 &= a_i b_i + (a_i \bar{b}_i + \bar{a}_i b_i) \cdot C_i \\
 &= a_i b_i + (a_i \oplus b_i) \cdot C_i
 \end{aligned} \tag{6.7}$$



Figuur 6.2. 2-Bit binaire full adder SN7482.

Deze formules resulteren in de schakeling van figuur 6.1.a. Merk op dat in deze schakeling twee keer een *half adder* (paragraaf 4.2.) voorkomt. Het timing model van de schakeling staat in figuur 6.1.b.

De timing eigenschappen van een 4-bit full adder op basis van de schakeling in figuur 6.1 zijn in paragraaf 5.11 verkend. Het voor de duur van de optelling bepalende signaal C_{i+1} wordt het meest vertraagd. Realisaties op basis van verschillende types poorten worden besproken in [Lai, 1979; Liu, 1974; Sukarai, 1983].

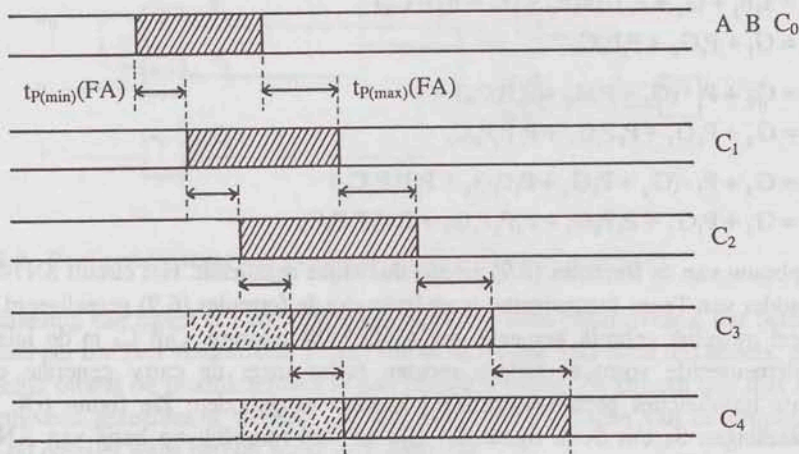
Optimalisatie van het full adder ontwerp

De beperkende factor in een optellerontwerp is de *carrypropagatie*. In een n -bit full adder doorloopt in het slechtste geval een in de eerste sectie gegenereerde carry de hele rij van full adders, om in de hoogstwaardige sectie te eindigen in het somsignaal S_{n-1} of in de uitgaande carry C_n . De *maximale propagatietijd* van de carry in een n -bit full adder is n keer de propagatietijd van een 1-bit full adder. (Zie ook het gestelde in paragraaf 5.11 ten aanzien van valse paden!) Dit betekent dat de propagatietijd $t_{p(max)}(n\text{-bit full adder})$,

$$t_{p(max)}(n\text{-bit full adder}) = n \cdot t_{p(max)}(1\text{-bit full adder}) \quad (6.8)$$

kleiner gemaakt kan worden door via een aangepast ontwerp de propagatietijd van de 1-bit full adder te reduceren. Alle tijdswinst in het full adder ontwerp komt n -voudig ten goede aan $t_{p(max)}(n\text{-bit full adder})$.

Een aardige implementatie, die berust op AND-NOR (AND-OR-NOT) poorten, staat in figuur 6.2. Een nadeel is dat de uitgaande carry C_1 gecomplementeerd gegenereerd wordt. Daarom is de tweede sectie aangepast. Deze is gebaseerd op $\bar{a}_1$, $\bar{b}_1$ en $\bar{C}_1$. De met het complementeren van a_1 en b_1 gemoeide propagatietijd is niet hinderlijk omdat het andereingangssignaal van de tweede sectie, de carry $\bar{C}_1$, ook reeds één poortniveau vertraagd is. Na twee van deze 2-bit full adders is de carry C_4 in de niet-gecomplementeerde vorm beschikbaar.



Figuur 6.3. Ripple carry timing.

Figuur 6.3 specificeert de timing van de ripple carry. Merk op dat in C_3 en C_4 twee niet-gedefinieerde gebieden in de signalen elkaar overlappen, die van de a_i 's en b_i 's enerzijds en die van de reeds vertraagde carry C_2 en C_3 anderzijds.

Andere voorbeelden van technologie-aanpassingen om de werking van een 1-bit full adder ontwerp te versnellen zijn besproken in [Tjon A Loi, 1992; Wei, 1990]. In [Wei] staan ook veel referenties. De mogelijkheden om een n -bit full adder te versnellen op basis van een slim full adder ontwerp en/of een aangepaste lay-out zijn weldra uitgeput. *Hoe snel we het full adder ontwerp ook weten te realiseren, de propagatietijd blijft lineair afhankelijk van het aantal secties van de opteller.* Verdere versnellingen vereisen een andere aanpak.

6.3 Carrypropagatie in 4-bit full adders

In feite heeft de ontwerper zich bij een transportopbouw volgens figuur 6.2 niet weten los te maken van de wijze waarop de optelling op papier gebeurt. Een andere benadering wordt mogelijk zodra men inziet dat optellen een *combinatorisch proces* is. Zodra de bits van de twee operanden hun definitieve waarde hebben aangenomen is alle informatie beschikbaar die nodig is om de sombits en het uitgaande carrybit te berekenen. Met name kan de carry C_4 rechtstreeks in de bits van de operanden en de inkomende carry C_0 worden uitgedrukt. De som-van-produktenvorm van de formule voor C_4 wordt daarbij erg omvangrijk, met veel en ook grote termen. Daarom worden de formules voor C_i vaak uitgedrukt in de eerder geïntroduceerde *carry generate functie* G_i en de *carry propagate functie* P_i . Deze hulpfuncties, die apart gerealiseerd moeten worden, maken het mogelijk de hierin uitgedrukte formule voor C_4 af te beelden op een poortschakeling in twee niveaus:

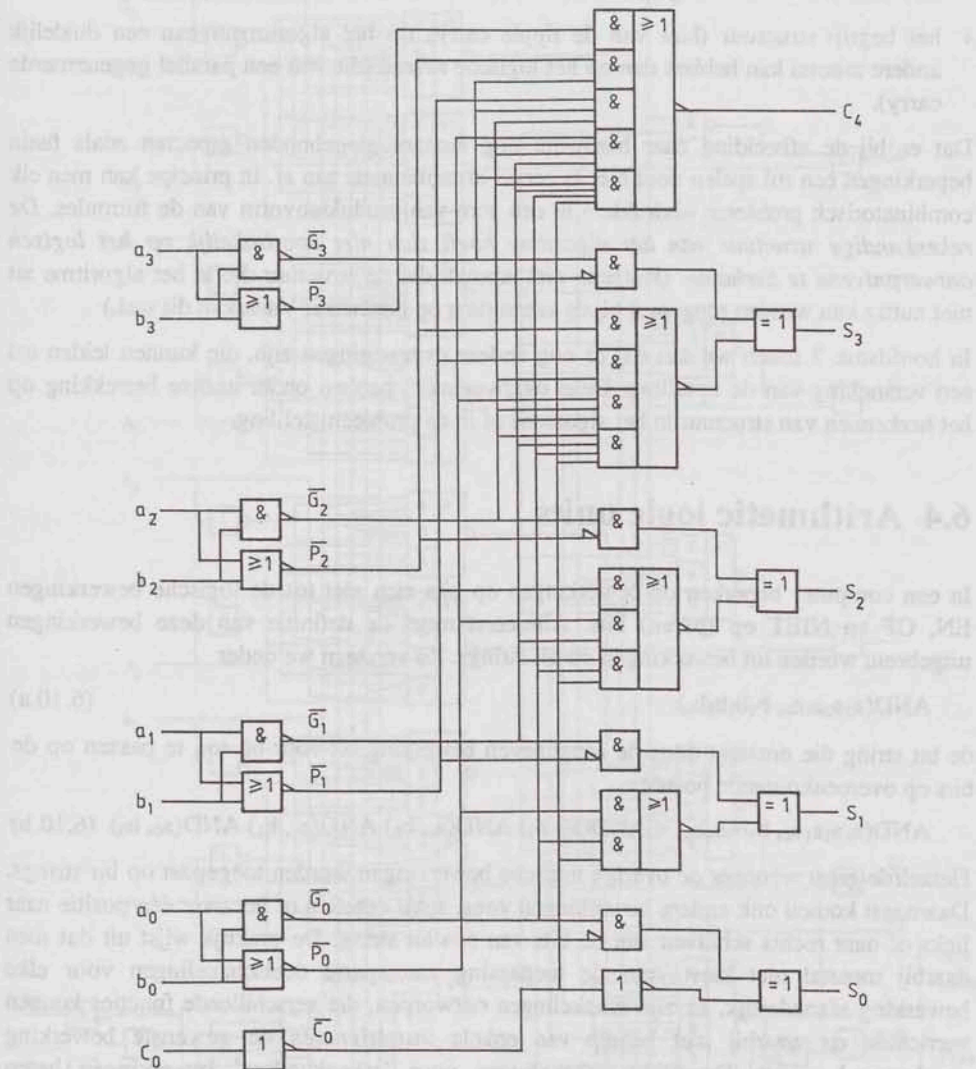
$$\begin{aligned} C_1 &= a_0 b_0 + (a_0 + b_0) \cdot C_0 \\ &= G_0 + P_0 C_0 \end{aligned} \quad (6.9.a)$$

$$\begin{aligned} C_2 &= a_1 b_1 + (a_1 + b_1) \cdot C_1 \\ &= a_1 b_1 + (a_1 + b_1) \cdot [a_0 b_0 + (a_0 + b_0) \cdot C_0] \\ &= G_1 + P_1 G_0 + P_1 P_0 C_0 \end{aligned} \quad (6.9.b)$$

$$\begin{aligned} C_3 &= G_2 + P_2 \cdot (G_1 + P_1 G_0 + P_1 P_0 C_0) \\ &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0 \end{aligned} \quad (6.9.c)$$

$$\begin{aligned} C_4 &= G_3 + P_3 \cdot (G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0) \\ &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0 \end{aligned} \quad (6.9.d)$$

In de opbouw van de formules (6.9) zit een duidelijke regelmaat. Het circuit SN7483A (4-bit full adder van Texas Instruments) is op basis van de formules (6.9) gerealiseerd. Daarbij is zo veel mogelijk gebruik gemaakt van AND-NOR poorten. Om C_4 in de juiste, niet-gecomplementeerde vorm te krijgen moeten onder meer de carry generate en carry propagate hulpfuncties gecomplementeerd worden aangeboden. Zie figuur 6.4. Ook de somschakelingen S_0 t/m S_3 in figuur 6.4 zijn zo veel mogelijk op basis van AND-NOR poorten gerealiseerd. We werken deze omzetting niet verder uit.



Figuur 6.4. Realisatieschema van een 4-bit opteller met snel transport.

In de schakeling van figuur 6.4 doorloopt het transport twee poortniveaus, dat betekent $\frac{1}{2}$ poortniveau per bit. Een verdere uitbreiding van deze aanpak naar n -bit full adders, $n > 4$, is niet mogelijk omdat de produkttermen te omvangrijk worden. Ze kunnen dan niet meer in één poortniveau gerealiseerd worden. De hier beschreven reductie van de propagatietijd van de 4-bit opteller werd bereikt zodra men inzag dat

- optellen een combinatorisch probleem is;

- elk combinatorisch probleem in principe in de som-van-produktenvorm van de logische formules kan worden uitgedrukt;
- het begrip structuur (hier van de ripple carry) op het algoritmeniveau een duidelijk andere inhoud kan hebben dan op het logische niveau (die van een parallel gegenereerde carry).

Dat er bij de afbeelding naar hardware nog technologiegebonden aspecten zoals fanin beperkingen een rol spelen doet hier in eerste instantie niets aan af. In principe kan men elk combinatorisch probleem uitdrukken in een som-van-produktenvorm van de formules. *De rekenkundige structuur van het algoritme hoeft zich niet noodzakelijk op het logisch ontwerpniveau te herhalen.* (Hetgeen niet inhoudt dat de structuur die in het algoritme zit niet nuttig kan worden toegepast bij de afbeelding op hardware. Vaak kan dit wel.)

In hoofdstuk 7 tonen we aan dat er nog andere overwegingen zijn, die kunnen leiden tot een versnelling van de optelling. Deze overwegingen hebben onder andere betrekking op het herkennen van structuur in het algoritme of in de probleemstelling.

6.4 Arithmetic logic units

In een computer beperken de bewerkingen op bits zich niet tot de logische bewerkingen EN, OF en NIET op (paren) bits. Allereerst moet de definitie van deze bewerkingen uitgebreid worden tot bewerkingen op bit strings. Zo verstaan we onder

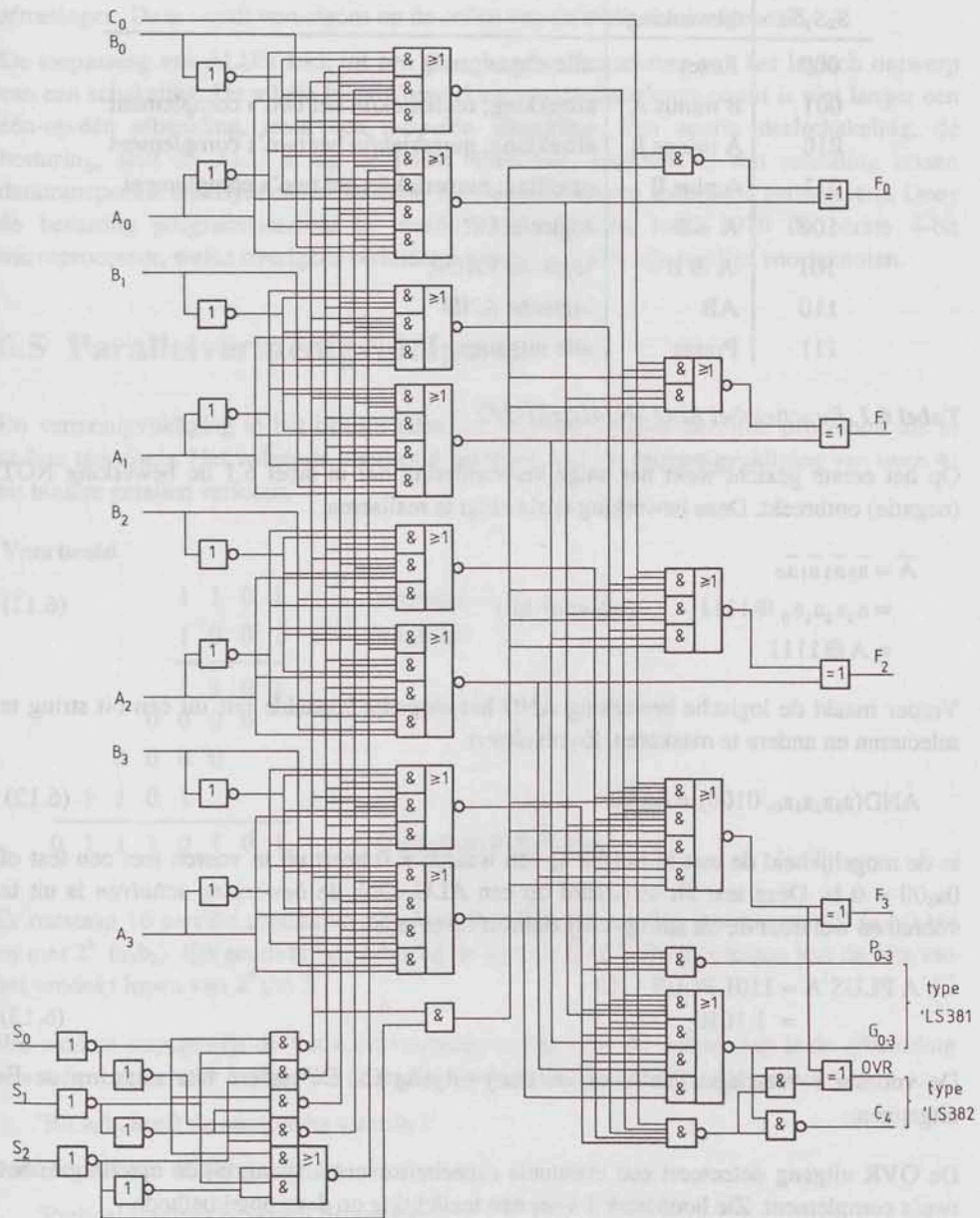
$$\text{AND}(a_3a_2a_1a_0, b_3b_2b_1b_0) \quad (6.10.a)$$

de bit string die ontstaat door de aangegeven bewerking bit-voor-bit toe te passen op de bits op overeenkomende posities:

$$\text{AND}(a_3a_2a_1a_0, b_3b_2b_1b_0) = \text{AND}(a_3, b_3) \text{ AND}(a_2, b_2) \text{ AND}(a_1, b_1) \text{ AND}(a_0, b_0) \quad (6.10.b)$$

Hetzelfde geldt wanneer de overige logische bewerkingen worden toegepast op bit strings. Daarnaast komen ook andere bewerkingen voor, zoals optellen of het over één positie naar links of naar rechts schuiven van de bits van een bit string. De praktijk wijst uit dat men daarbij meestal niet kiest voor de toepassing van aparte deelschakelingen voor elke bewerking afzonderlijk. Er zijn schakelingen ontworpen, die verschillende functies kunnen verrichten en waarbij met behulp van enkele *instelsignalen* de gewenste bewerking geselecteerd wordt. Dergelijke schakelingen voor 'rekenkundige' bewerkingen heten *ALU's* (Arithmetic Logic Units). Zij vormen het hart van een processor. Rond de ALU zit tevens een aantal registers voor tijdelijke opslag van data. Deze registers laten we hier buiten beschouwing. Zie ook hoofdstuk 16.

Figuur 6.5 beschrijft het logische schema van de ALU SN74LS381/382. Deze 4-bit ALU heeft twee 4-bit data-ingangen, waarop de bit strings $a_3a_2a_1a_0$ en $b_3b_2b_1b_0$ worden aangeboden. Daarnaast is er een carry-ingang voor een eventuele inkomende carry C_0 . Verder zijn er drie *instelingen*, waarmee van de acht mogelijke bewerkingen er één kan worden geselecteerd.



Figuur 6.5. Poortschema ALU SN74LS381/382 (Texas Instruments).

Tabel 6.1 is de functietabel van de ALU. Links staat de ingangscombinatie en rechts staat de bijbehorende bewerking.

Instelling $S_2S_1S_0$	Ingestelde bewerking	Resultaat
000	Reset	alle uitgangen 0
001	B minus A	afrekking; numeriek/in het two's complement
010	A minus B	afrekking, numeriek/in het two's complement
011	A plus B	optelling; numeriek/in het two's complement
100	A + B	logische OR
101	A $\oplus$ B	logische EXOR
110	AB	logische AND
111	Preset	alle uitgangen 1

Tabel 6.1. Functietabel ALU SN74LS381/382.

Op het eerste gezicht wekt het enige verwondering dat in tabel 6.1 de bewerking NOT (negatie) ontbreekt. Deze bewerking is als volgt te realiseren.

$$\begin{aligned}
 \overline{A} &= \overline{a_3}\overline{a_2}\overline{a_1}\overline{a_0} \\
 &= a_3a_2a_1a_0 \oplus 1111 \quad (\text{bit-voor-bit}) \\
 &= A \oplus 1111
 \end{aligned} \tag{6.11}$$

Verder maakt de logische bewerking AND het mogelijk bepaalde bits uit een bit string te selecteren en andere te maskeren. Zo resulteert

$$\text{AND}(a_3a_2a_1a_0, 0100) = 0a_200 \tag{6.12}$$

in de mogelijkheid de test of het bit a_2 een waarde $\neq 0$ heeft uit te voeren met een test of $0a_200 \neq 0$ is. Deze test zit standaard op een ALU. Ook de *bewerking schuiven* is uit te voeren en wel door de bit string A bij zichzelf op te tellen:

$$\begin{aligned}
 A \text{ PLUS } A &= 1101 \text{ PLUS } 1101 \\
 &= 1 \ 1010
 \end{aligned} \tag{6.13}$$

De voorste 1 verschijnt hierbij op de carry-uitgang C_4 . De andere bits staan op de F-uitgangen.

De OVR uitgang detecteert een eventuele capaciteitsoverschrijding bij de optelling in het two's complement. Zie hoofdstuk 1 voor een toelichting op deze optelmethode.

Het vorenstaande geeft een kort overzicht van de mogelijkheden van ALU's. De toepasbaarheid ervan is veel groter dan die van 4-bit full adders. ALU's zijn er in verschillende uitvoeringen, met drie of vier instelsignalen. Een cascadeschakeling tot 8-bit of 16-bit woordbreedte is zonder meer mogelijk. Daarbij moeten o.a. de instelsignalen met elkaar verbonden worden. In een bibliotheek wordt een ALU meestal geparameteriseerd beschreven. Met een van de parameters kan het aantal secties worden ingesteld. Na

expansie ontstaat er dan een logische beschrijving van een schakeling van de gewenste afmetingen. Deze wordt vervolgens op de cellen van de bibliotheek afgebeeld.

De toepassing van ALU's leidt tot een geheel andere benadering van het logisch ontwerp van een schakeling. De relatie functie/bewerking $\leftrightarrow$ standaardcomponent is niet langer een één-op-één afbeelding, maar een n-op-één afbeelding. Een aparte deelschakeling, de besturing, stelt de ALU in op de juiste bewerking. Daarmee is een scheiding tussen datatransporten enerzijds en het instellen van de bewerkingen anderzijds gerealiseerd. Door de besturing programmeerbaar te maken ontstond aldus rond 1970 de eerste 4-bit microprocessor, welke overigens beduidend minder kon dan zijn huidige soortgenoten.

6.5 Parallelvermenigvuldigers

De vermenigvuldiging in het binaire talstelsel verloopt volgens dezelfde procedure als in andere talstelsels. Het volgende voorbeeld geeft aan hoe de vermenigvuldiging van twee 4-bit binaire getallen verloopt.

Voorbeeld

$$\begin{array}{r}
 1101 \quad (a_3a_2a_1a_0) \\
 1001 \quad (b_3b_2b_1b_0) \quad \times \\
 \hline
 1101 \\
 0000 \\
 0000 \\
 1101 \quad + \\
 \hline
 01110101 \quad (P_7P_6P_5P_4P_3P_2P_1P_0)
 \end{array}$$

Er ontstaan 16 partiële produkten $a_i b_j$. Van deze bits varieert het gewicht van 2^0 ($a_0 b_0$) tot en met 2^6 ($a_3 b_3$). Elk produkt heeft daarbij de waarde 0 of 1. De gewichten van de bits van het produkt lopen van 2^0 t/m 2^7 . $\square$

We werken stapsgewijs de 4 x 4 bit vermenigvuldiger uit. De eerste stap is de *afbeelding van rekenkunde naar logica*. We doen dit met de voor de hand liggende proposities

'Bit a_i/b_j heeft de numerieke waarde 1'

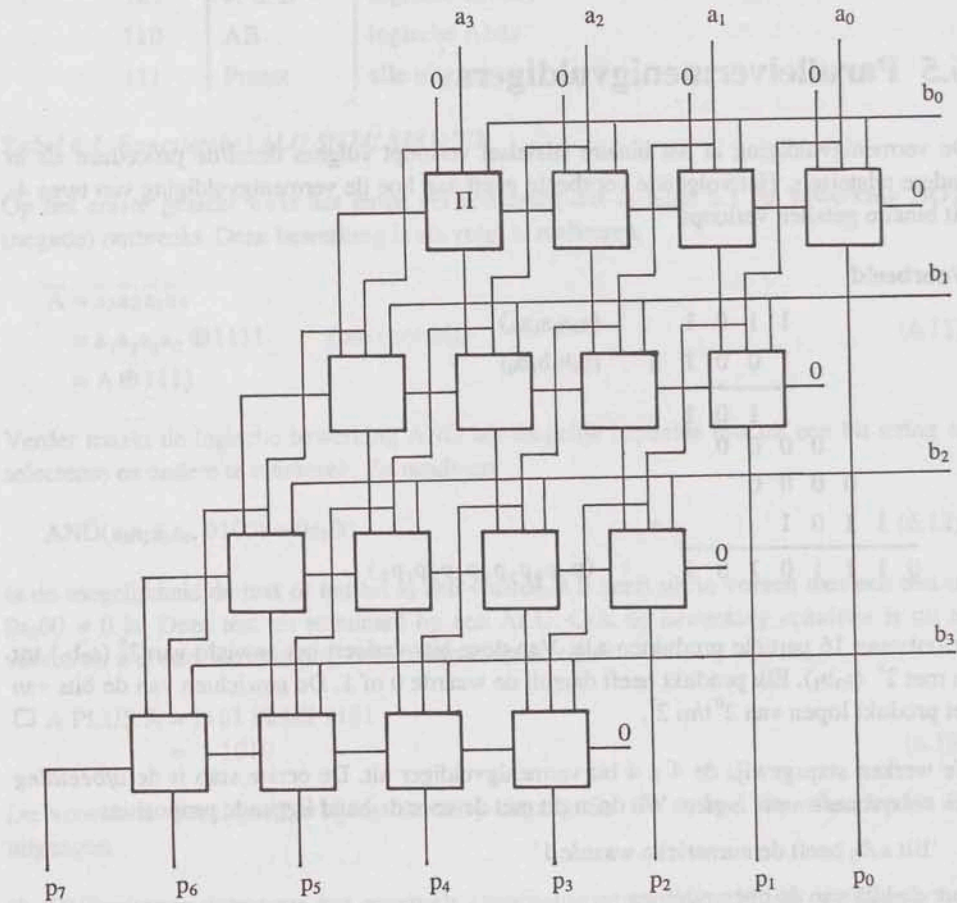
voor de bits van de operanden en

'Partieel produkt $a_i b_j$ heeft de waarde 1'

voor de bits van het partiële produkt. Zo ook voor de sombits. Al doende ontstaat de *waarheidstabel* 6.2. Deze tabel geldt voor elk partiële produkt. Tabel 6.2 is identiek aan de waarheidstabel van de logische AND. Met deze conclusie is een schakeling gevonden om de partiële produkten te maken.

a_i	b_j	$a_i b_j$		a_i	b_j	$a_i b_j$
F	F	F	$F \leftrightarrow 0$	0	0	0
F	T	F		0	1	0
T	F	F	$T \leftrightarrow 1$	1	0	0
T	T	T		1	1	1

Tabel 6.2. Waarheidstabel van het partiële produkt.



Figuur 6.6. 4 x 4 bit parallelvermenigvuldiger.

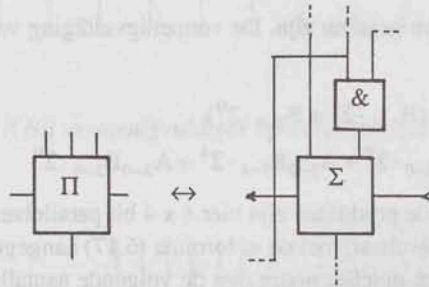
Figuur 6.6 beschrijft de structuur van een 4 x 4 bit parallelvermenigvuldiger op basis van een array van identieke cellen. In elke laag wordt een partiële produkt $b_j A$,

$$b_j A = b_j a_3 b_j a_2 b_j a_1 b_j a_0$$

bepaald en bij het vorige tussenresultaat opgeteld. Het vermenigvuldigalgoritme luidt:

$$\begin{aligned}
 X_0 &= 0 \\
 X_1 &= b_0 A \cdot 2^0 + X_0 \\
 X_2 &= b_1 A \cdot 2^1 + X_1 \\
 X_3 &= b_2 A \cdot 2^2 + X_2 \\
 X_4 &= b_3 A \cdot 2^3 + X_3 \\
 &= b_3 A \cdot 2^3 + b_2 A \cdot 2^2 + b_1 A \cdot 2^1 + b_0 A \cdot 2^0 \\
 &= P_7 P_6 \cdots P_1 P_0 \text{ BIN}
 \end{aligned} \tag{6.14}$$

Figuur 6.7 beschrijft de inhoud van één cel ervan. Elke cel bestaat uit één AND voor de bepaling van het partiële product en uit één full adder om het partiële product op te tellen bij het vorige tussenresultaat. Een op deze wijze opgebouwde parallelvermenigvuldiger voor 8 x 8 bits bestaat uit 256 van deze cellen.



Figuur 6.7. Cel van een parallelvermenigvuldiger.

Een aantal cellen kan vereenvoudigd worden. Zo is er op de eerste rij nog geen partieel produkt. In deze cellen is daarom de full adder overbodig. Verder heeft de rechter cel van elke rij geen inkomende carry. Hier volstaat een half adder. Dergelijke vereenvoudigingen passen in een 'application specific' omgeving. Voor een software implementatie zijn ze vaak niet relevant.

Opmerking

In de tekeningen van dit boek houden we volgens een eerdere afspraak zo veel mogelijk de links-naar-rechts richting van de data-overdracht aan. In de figuren 6.6 en verder zijn we hiervan afgeweken. De reden is dat de rekenkundige 'volgorde' van de cijfers op papier meestal anders is. Om geen verwarring te scheppen kiezen we daarom soms voor een andere tekenwijze. □

6.6 Modulair opgebouwde vermenigvuldigers

Grote vermenigvuldigers worden vaak samengesteld uit kleinere. Als voorbeeld nemen we de opbouw van een 8 x 8 bit vermenigvuldiger uit vier 4 x 4 bit vermenigvuldigers. (Waarbij wordt opgemerkt dat 8 x 8 bit vermenigvuldigers gewoonlijk ook uit één array bestaan).

Twee binair gecodeerde getallen A en B van elk acht bits mogen we ook schrijven als

$$\begin{aligned} A &= a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0_{\text{BIN}} \\ &= a_7 a_6 a_5 a_4_{\text{BIN}} \cdot 2^4 + a_3 a_2 a_1 a_0_{\text{BIN}} \cdot 2^0 \\ &= A_{7-4} \cdot 2^4 + A_{3-0} \cdot 2^0 \end{aligned} \quad (6.15)$$

en

$$B = B_{7-4} \cdot 2^4 + B_{3-0} \cdot 2^0 \quad (6.16)$$

waarin A_{7-4} t/m B_{3-0} 4-bit binaire getallen zijn. De vermenigvuldiging van A en B komt nu neer op

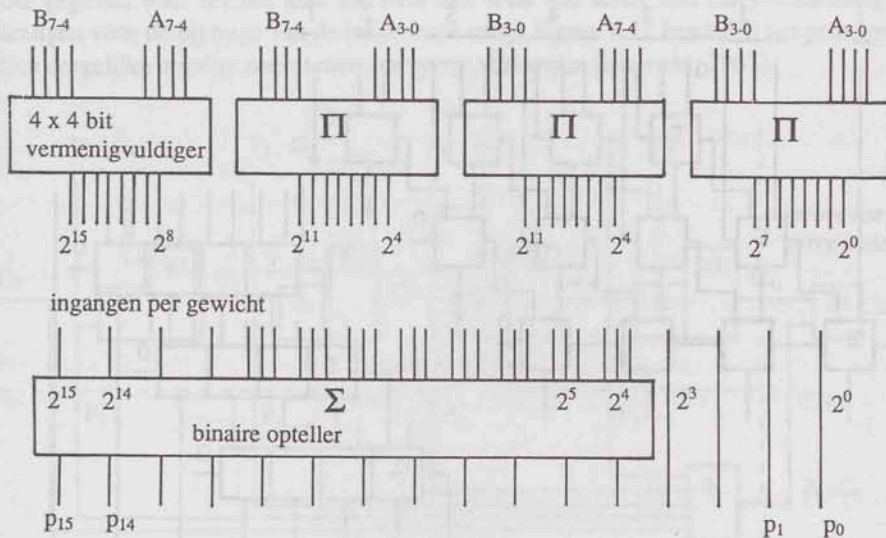
$$\begin{aligned} A \cdot B &= (A_{7-4} \cdot 2^4 + A_{3-0} \cdot 2^0) \cdot (B_{7-4} \cdot 2^4 + B_{3-0} \cdot 2^0) \\ &= A_{7-4} B_{7-4} \cdot 2^8 + A_{7-4} B_{3-0} \cdot 2^4 + A_{3-0} B_{7-4} \cdot 2^4 + A_{3-0} B_{3-0} \cdot 2^0 \end{aligned} \quad (6.17)$$

Voor de vorming van de vier partiële produkten zijn vier 4 x 4 bit parallelvermenigvuldigers nodig. Deze leveren elk een 8-bit resultaat, met de in formule (6.17) aangegeven gewichten. Voor deze partiële produkten is een opteller nodig met de volgende aantallen ingangen per bit:

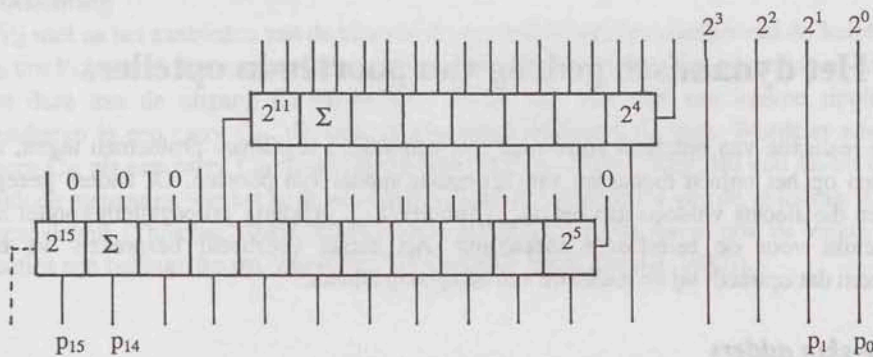
gewicht	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
aantal ingangen	1	1	1	1	3	3	3	3	3	3	3	3	1	1	1	1

Dit resulteert in een 8 x 8 bit vermenigvuldiger met de opbouw volgens figuur 6.8.

De opteller, die nodig is om de partiële produkten op te tellen, kan worden opgebouwd volgens figuur 6.9. In de opteller van figuur 6.9 is gebruik gemaakt van een *woordgeoriënteerde opteller* [Oberman, 1979]. Hierin worden woorden met een oplopend gewicht aangeboden aan 4- of n-bit full adders. Volgens een andere benadering wordt er 'per gewicht' gekeken hoeveel bits er voor dat gewicht opgeteld moeten worden, rekening houdend met de inkomende carries en hun eventuele propagatietijd. Dan kan er per gewicht een 'opteller op maat' gemaakt worden, die alle bits zo snel mogelijk optelt. Daarbij is van belang de uitgaande carries zo snel mogelijk te genereren. Dit noemt men een *bitgeoriënteerde opteller*.



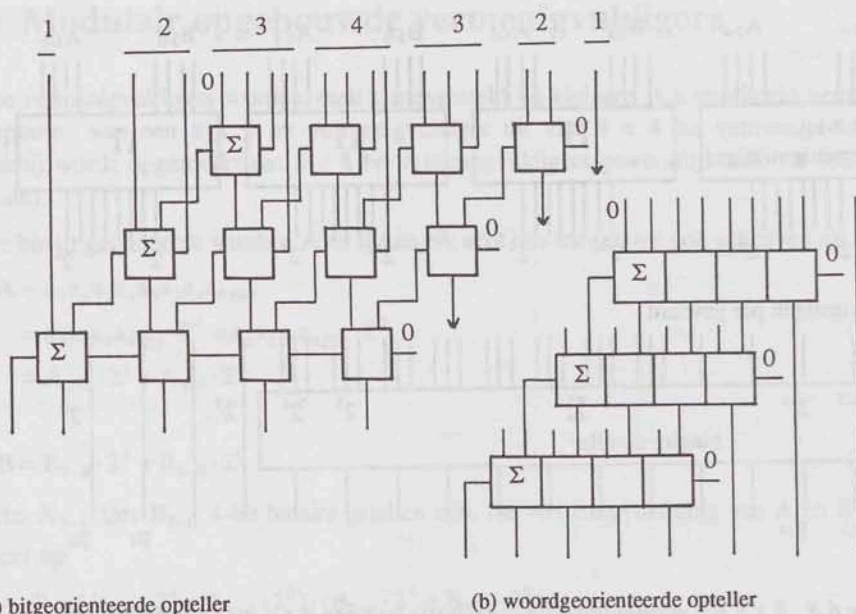
Figuur 6.8. 8 x 8 Bit vermenigvuldiger op basis van een 4 x 4 bit vermenigvuldiger.



Figuur 6.9. Opteller voor een modulair opgebouwde 8 x 8 bit vermenigvuldiger.

Figuur 6.10 laat een bitgeoriënteerde opteller zien voor een 4 x 4 bit parallelvermenigvuldiger. Ga na welke winst in propagatietijd hiermee behaald wordt, vergeleken met het maximale carry pad in een woordgeoriënteerde opteller die bestaat uit drie 4-bit full adders.

We concluderen dat er verschillende manieren zijn om tegen een opteller aan te kijken. Een opteller is dus niet automatisch een uit n -bit full adders opgebouwde structuur. Welke structuur in een bepaalde toepassing tot de beste resultaten leidt (uitgedrukt in snelheid van de optelling of in een kleiner oppervlak bijvoorbeeld) is vaak niet vooraf te zeggen. Nader onderzoek op technologieniveau moet uitsluitsel geven op deze vraag. Daarbij speelt de interconnect een steeds grotere rol.



Figuur 6.10. Bitgeoriënteerde opteller (carry-save adder).

6.7 Het dynamisch gedrag van poorten in optellers

Bij de realisatie van optellers komt men met een zekere regelmaat problemen tegen, die berusten op het onjuist toepassen van het timing model van poorten. Of anders gezegd, poorten die slechts voldoen aan het $t_{P(\min)}(\text{poort})/t_{P(\max)}(\text{poort})$ propagatietijdmodel zijn ongeschikt voor de betreffende toepassing. Als eerste voorbeeld bespreken we een probleem dat optreedt bij de realisatie van carry-skip adders.

Carry-skip adders

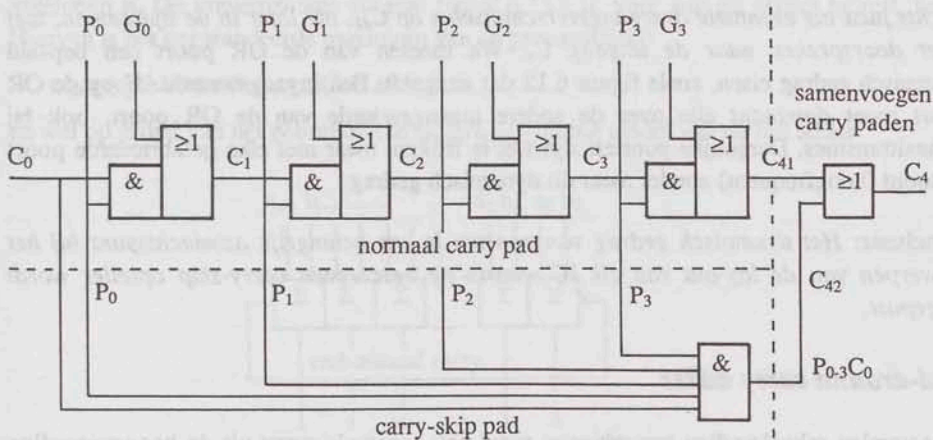
Volgens formule (6.9.d) kan de uitgaande carry C_4 van een 4-bit full adder worden uitgedrukt als een functie van de verschillende carry generate en carry propagate hulpfuncties, alsmede in de inkomende carry C_0 :

$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

In deze formule geven de eerste vier produkttermen aan wanneer er in de betreffende 4-bit sectie een uitgaande carry C_4 gegenereerd wordt. Deze bijdrage aan de carry C_4 is *onafhankelijk van de inkomende carry* C_0 . Daarnaast is er ook een mogelijke bijdrage van de inkomende carry C_0 aan de uitgaande carry C_4 en wel als $P_3P_2P_1P_0 = 1$ is. Deze laatste term is *onafhankelijk van een interne carry* in de 4-bit sectie zelf. Immers:

$$P_3P_2P_1P_0 = (a_3 + b_3)(a_2 + b_2)(a_1 + b_1)(a_0 + b_0) \quad (6.18)$$

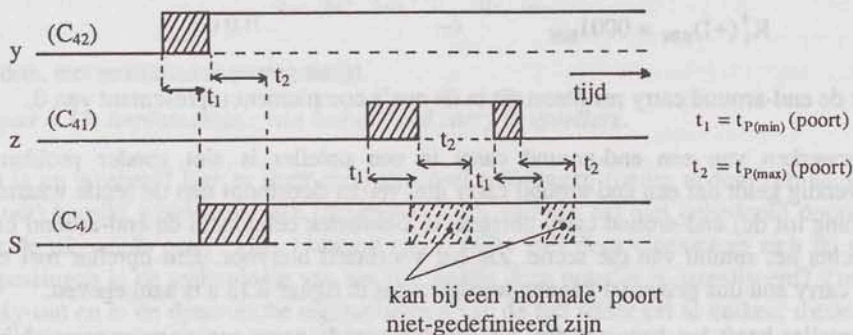
Dit gegeven leidt tot het idee om over een n -bit full adder een carry-versnelling aan te brengen voor de bijdrage van de inkomende carry. Figuur 6.11 beschrijft het principe ervan. Een dergelijke opteller noemt men een *carry-skip adder* [Majerski, 1967].



Figuur 6.11. Carry-skip adder.

Toelichting

Vrij snel na het aanbieden van de bits van de operanden zijn de waarden van de hulpfuncties P_0 t/m P_3 bekend. Samen met C_0 kan dit een carry C_{42} opleveren. Een poortniveau later zien we deze aan de uitgang C_4 verschijnen. Is $C_0 = 0$, dan kan een interne ripple carry resulteren in een carry C_{41} , die ook tot een uitgaande carry C_4 leidt. Wordt er zowel een carry C_{41} als een carry C_{42} gegenereerd, dan wint de snelste carry. Bij een $4n$ -bit opteller leidt de toepassing van het geschetste principe tot een versnelling van de optelling. Zie voor meer details [Thijssen, 1995]. (Bij optimale carry-skip adders speelt ook de lengte van de secties een belangrijke rol. Zie [Chan, 1989; IBM, 1985; Turrini, 1989]). □



Figuur 6.12. Tijdsmodel voor OR poort met dominante ingang.

De werking van de carry-skip adder berust op het feit dat een eventuele carry C_{42} via de OR poort alvast doorgelaten wordt en aldus de carry C_4 bepaalt. Er behoeft dan niet meer op de andere component van de carry C_4 gewacht te worden. *Deze veronderstelling is slechts juist als eventuele overgangsverschuiven op C_{41} , die later in de tijd komen, niet meer doorspreken naar de uitgang C_4 .* We moeten van de OR poort een bepaald dynamisch gedrag eisen, zoals figuur 6.12 dat aangeeft. Een ingangswaarde '1' op de OR poort moet *dominant* zijn over de andere ingangswaarde van de OR poort, ook bij signaaltransities. Dergelijke poorten zijn wel te maken, maar niet elke gefabriceerde poort vertoont (hoogfrequent) zonder meer dit dynamisch gedrag.

Conclusie: Het dynamisch gedrag van poorten is een belangrijk aandachtspunt bij het ontwerpen van de lay-out van elk IC waarin de beschreven carry-skip opteller wordt toegepast.

End-around carry adder

Bij sommige rekenkundige bewerkingen moet een eventuele carry uit de hoogstwaardige positie (gewicht 2^n) weer op de laagstwaardige positie bijgeteld worden (met gewicht 2^0). Dit komt onder andere voor bij de *one's complement optelling*. In het one's complement rekent men 'modulo $2^n - 1$ '. Het weglaten van een carry met gewicht 2^n zou dus 1 te veel weglaten [Thijssen, 1995].

Voorbeeld

In het volgende voorbeeld is er sprake van een end-around carry.

$$\begin{array}{rcll}
 +3 & R_1^4(+3)_{\text{BIN}} = 0011_{\text{BIN}} & \rightarrow & 0011 \\
 -2 + & R_1^4(-2)_{\text{BIN}} = 1101_{\text{BIN}} + & \rightarrow & \underline{1101} + \\
 +1 & [R_1^4(+3) + R_1^4(-2)]_{\text{BIN}} & \rightarrow & \begin{array}{r} 10000 \\ \rightarrow 1+ \end{array} \\
 & R_1^4(+1)_{\text{BIN}} = 0001_{\text{BIN}} & \leftarrow & 0001
 \end{array}$$

Zonder de end-around carry resulteert dit in de one's complement representant van 0. $\square$

Het verwerken van een end-around carry in een opteller is niet zonder problemen. Rekenkundig geldt dat een end-around carry niet verder doorloopt dan de sectie waarin de (aanleiding tot de) end-around carry ontstaan is. Numeriek beïnvloedt de end-around carry dan slechts het sombit van die sectie. Zie het voorbeeld hiervoor. Een opteller met end-around carry zou dus gebouwd kunnen worden zoals in figuur 6.13.a is aangegeven.

Deze opteller heeft het bezwaar dat, als een inkomende carry een overgangsverschuif induceert dat doorloopt naar de uitgang van de betreffende full adder cel, er in principe oneindig lang een overgangsverschuif in de opteller kan rondzingen. In de praktijk sterft

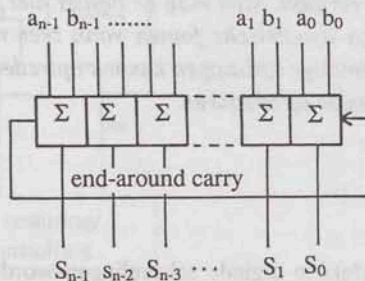
zo'n overgangsverschijnsel weldra uit, maar er kan niet zonder meer gegarandeerd worden dat de optelling in

$$t_{P(\max)}(\text{opteller}) \leq n \cdot t_{P(\max)}(\text{full adder}) \quad (6.19)$$

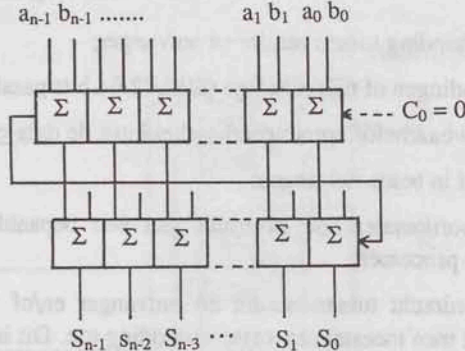
afgelopen is. De implementatie volgens figuur 6.13.b is, voor wat dit aspect betreft, beter. Hiervan is het gegarandeerde maximum van de propagatietijd

$$t_{P(\max)}(\text{opteller}) \leq 2n \cdot t_{P(\max)}(\text{full adder}) \quad (6.20)$$

en wel op grond van het gebruikelijke $t_{P(\min)}/t_{P(\max)}$ timing model van de full adder.



(a) opteller met end-around carry / met niet-gedefinieerde propagatietijd



(b) idem, met gedefinieerde propagatietijd.

Figuur 6.13. Implementatie van end-around carry in optellers.

Wat is nu wijsheid? Kan er door een aangepast celontwerp (onder andere een aangepaste lay-out) bereikt worden dat een inkomende carry van een cel niet onbedoeld doorspreekt naar de uitgaande carry? Hoe gedraagt een opteller met deze eigenschap zich bij geringe aanpassingen in de technologie van het IC, waarin deze opteller is gerealiseerd? Zonder in de lay-out en in de dynamische eigenschappen van de full adder cel te duiken, mede bezien de omgeving van de cel op het silicium, kan er op deze vragen meestal geen 100 % betrouwbaar antwoord gegeven worden [Shedlesky, 1977]. We geven slechts als onze mening dat in hoogfrequent schakelingen de hier geschetste problematiek vaker optreedt

dan menig ontwerper zich bewust is. Zoals het voorbeeld hiervoor aantoont zijn dergelijke problemen soms oplosbaar door het inzetten van extra logische componenten. (Voer voor minimaliseerders!)

De hier besproken problematiek, we hebben dit reeds eerder opgemerkt, komt in veel digitale schakelingen voor. Voor de ontwerper van logische schakelingen betekent dit dat op zijn minst al die plaatsen waar extra eisen aan het timing model gesteld zijn gelokaliseerd moeten worden. Vervolgens moeten deze eisen expliciet geformuleerd worden. De oplossing kan vervolgens bestaan uit aanpassingen van het logisch ontwerp, zoals bij de end-around carry, of het probleem moet doorgegeven worden aan degene die het logisch ontwerp naar silicium vertaalt. Men mag er beslist niet van uitgaan dat 'her' wel goed zal gaan. Bovendien zijn dynamische fouten vaak zeer moeilijk op te sporen omdat zij zeer zelden of op onregelmatige tijdstippen kunnen optreden (hazards). Er is dan een groot aantal metingen nodig om ze op te sporen.

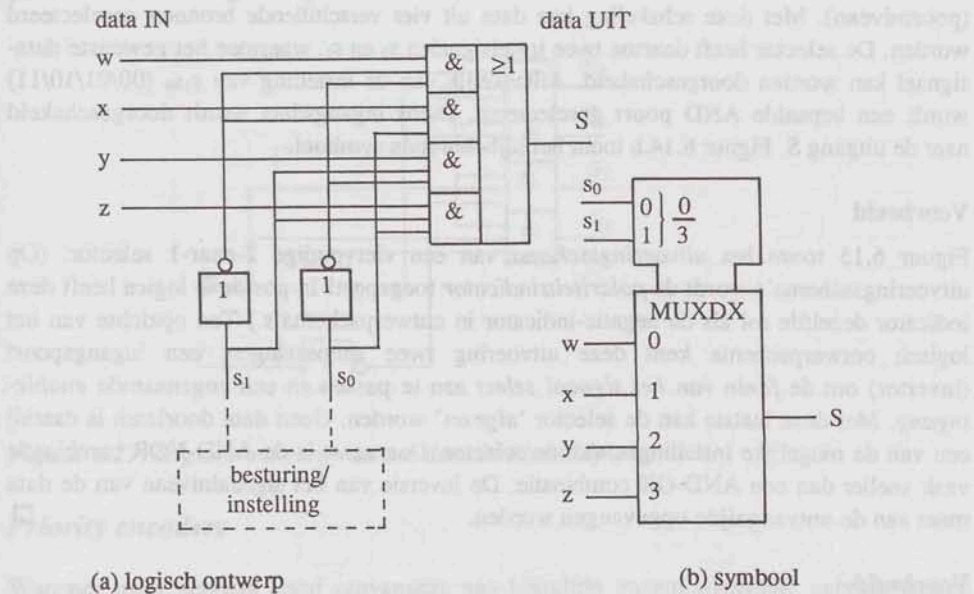
6.8 Data-overdracht

De keuze van de manier waarop data in digitale schakelingen wordt overgedragen hangt van veel aspecten af, waaronder

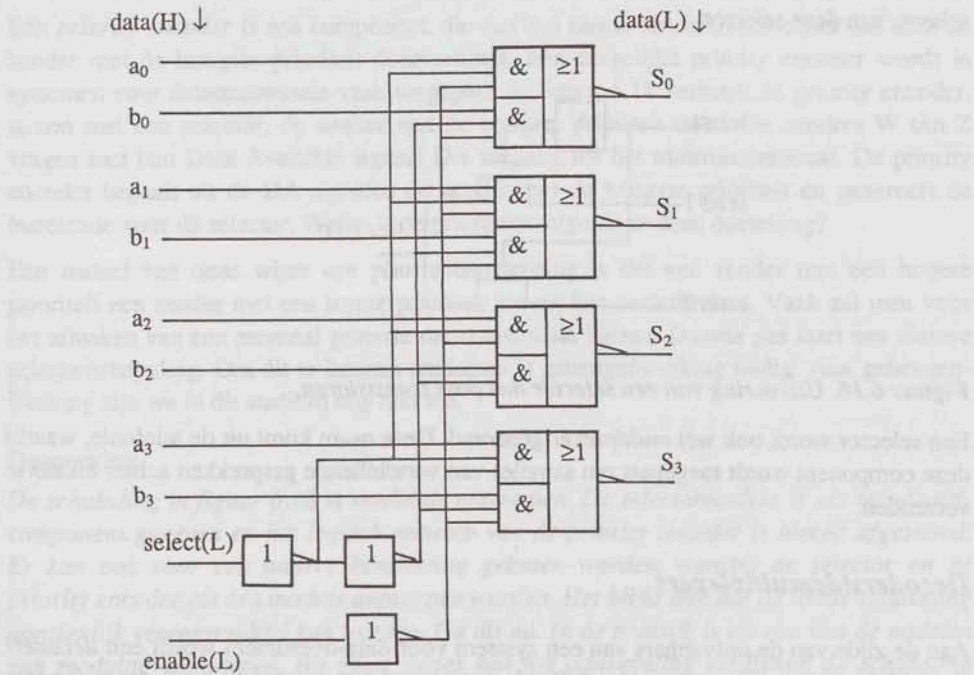
- de hoeveelheid data per tijdseenheid tussen bron/zender en bestemming/ontvanger (eenheid: bits/s);
- de lengte van de verbinding tussen zender en ontvanger;
- enkelvoudige verbindingen of meervoudige (8/16/32/64 bits parallel);
- de gewenste betrouwbaarheid/reproduceerbaarheid van de data-overdracht;
- dataverkeer in een of in beide richtingen;
- bij gedeelde transportkanalen: de prioriteit van een bepaalde data-overdracht ten opzichte van andere processen.

Bij intensieve data-overdracht tussen zender en ontvanger en/of tussen dicht bij elkaar geplaatste modules past men meestal een vaste verbinding toe. Dit is bijvoorbeeld het geval tussen de poorten van combinatorische logica. Bij meer incidentele data-overdracht en bij (relatief) lange verbindingen kiest men vaak voor een selectiemechanisme bij de zenders en/of bij de ontvangers. Op deze wijze kan het transportkanaal door verschillende gebruikers gedeeld worden. Dit kan oppervlak in een IC of anderszins in een kabel besparen. (Werp eens een blik in oude computers van studieverzamelingen of in een techniek-museum!) Een nadeel van deze opzet is dat er problemen kunnen ontstaan als verschillende gebruikers tegelijk toegang vragen tot de gezamenlijke transportweg. Dan moet er op de een of andere wijze een prioriteit bepaald worden tussen de verschillende zenders en/of ontvangers. In deze paragraaf gaan we in op enkele hardware componenten, die toegepast worden in systemen voor data-overdracht. Andere aspecten, zoals de keuze voor synchrone of asynchrone data-overdracht en busprotocollen, komen later aan bod.

Selectoren/multiplexers



Figuur 6.14. Enkelvoudige 4-naar-1 selector



Figuur 6.15. Uitvoering van een viervoudige 2-naar-1 selector.

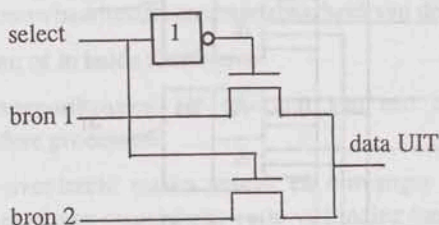
Een selector is een component, waarmee een keuze gemaakt kan worden uit data van verschillende bronnen. Figuur 6.14 beschrijft het logisch ontwerp van een 4-naar-1 selector (poortniveau). Met deze schakeling kan data uit vier verschillende bronnen geselecteerd worden. De selector heeft daartoe twee instelsignalen s_0 en s_1 , waarmee het gewenste data-sigitaal kan worden doorgeschakeld. Afhankelijk van de instelling van s_1s_0 (00/01/10/11) wordt een bepaalde AND poort geselecteerd. Diens ingangsdata wordt doorgeschakeld naar de uitgang S. Figuur 6.14.b toont het bijbehorende symbool.

Voorbeeld

Figuur 6.15 toont het uitvoeringsschema van een viervoudige 2-naar-1 selector. (Op uitvoeringsschema's wordt de *polariteitsindicator* toegepast! In positieve logica heeft deze indicator dezelfde rol als de negatie-indicator in ontwerpschema's.) Ten opzichte van het logisch ontwerpschema kent deze uitvoering twee aanpassingen, een ingangspoort (invertor) om de *fanin* van het signaal *select* aan te passen en een zogenaamde *enable-ingang*. Met deze laatste kan de selector 'afgezet' worden. Geen data doorlaten is daarbij een van de mogelijke instellingen van de selector. Daarnaast is de AND-NOR combinatie vaak sneller dan een AND-OR combinatie. De inversie van het signaalniveau van de data moet aan de ontvangzijde opgevangen worden. □

Voorbeeld

Figuur 6.16 beschrijft een vaak in CMOS toegepaste constructie, waarbij met pass transistoren (of met transmission gates) een selector wordt gerealiseerd. Tekent het poortschema van deze selector.



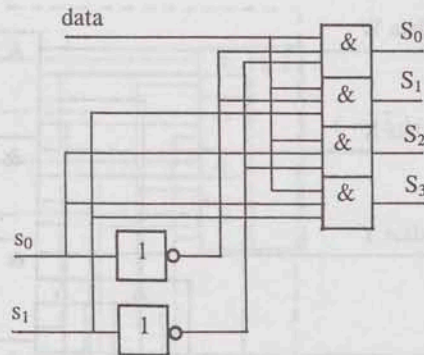
Figuur 6.16. Uitvoering van een selector met pass transistoren. □

Een selector wordt ook wel *multiplexer* genoemd. Deze naam komt uit de telefonie, waarin deze component wordt toegepast om samples van verschillende gesprekken achter elkaar te verzenden.

Decoders/demultiplexers

Aan de zijde van de ontvangers van een systeem voor data-overdracht wordt een *decoder/demultiplexer* toegepast als er verschillende ontvangers zijn en men niet wil dat de andere ontvangers kunnen meeluisteren. Figuur 6.17 beschrijft het logisch ontwerp van wat men

een 2-line-to-4-line decoder/demultiplexer noemt. Met deze schakeling kan men de gewenste bestemming aanwijzen.



Figuur 6.17. Logisch ontwerp van een decoder/demultiplexer.

Priority encoders

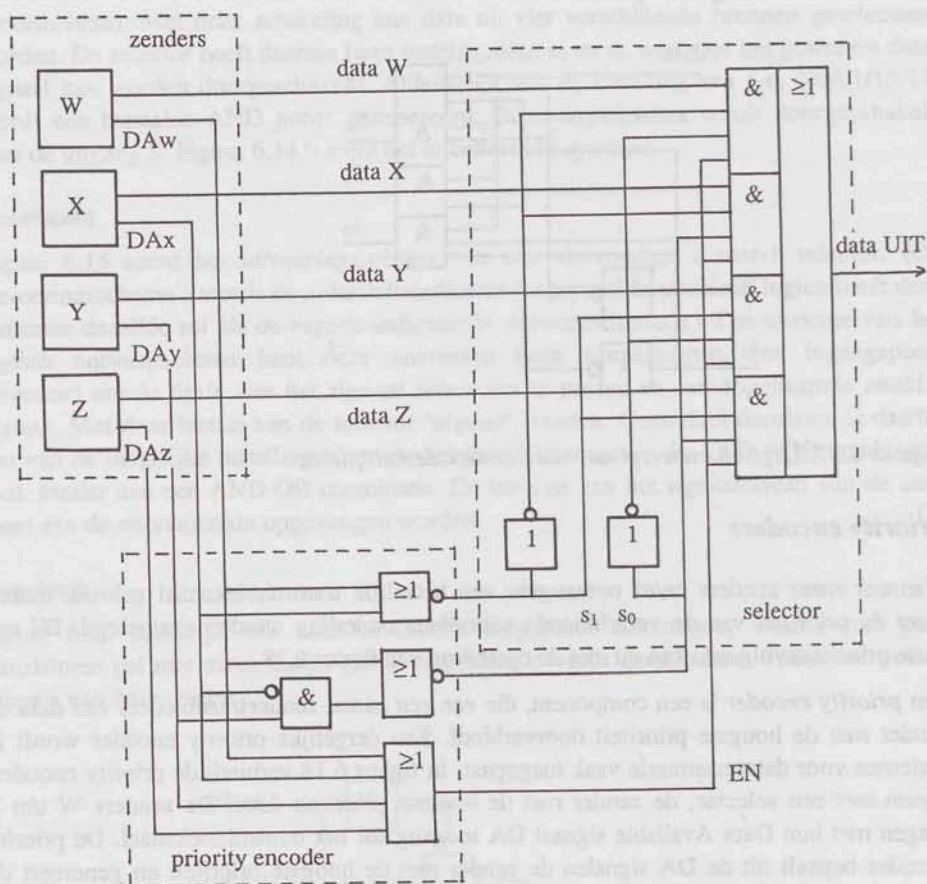
Wanneer meer zenders en/of ontvangers van hetzelfde transmissiekanaal gebruik maken moet de prioriteit van de verschillende gebruikers onderling worden vastgelegd. Bij een vaste prioriteitsvolgorde kan dit met de opstelling van figuur 6.18.

Een *priority encoder* is een component, die van een aantal zenders/aanbieders van data de zender met de hoogste prioriteit doorverbindt. Een dergelijke priority encoder wordt in systemen voor datatransmissie vaak toegepast. In figuur 6.18 verbindt de priority encoder, samen met een selector, de zender met de hoogste prioriteit door. De zenders W t/m Z vragen met hun Data Available signaal DA toegang tot het transmissiekanaal. De priority encoder bepaalt uit de DA signalen de zender met de hoogste prioriteit en genereert de instelcode voor de selector. Welke is de prioriteitsvolgorde in deze opstelling?

Een nadeel van deze wijze van prioriteitstoewijzing is dat een zender met een hogere prioriteit een zender met een lagere prioriteit abrupt kan onderbreken. Vaak zal men voor het afmaken van een eenmaal gestarte datatransmissie kiezen. Daarna pas start een nieuwe prioriteitsbepaling. Om dit te kunnen realiseren is geheugenwerking nodig. Aan geheugenwerking zijn we in dit stadium nog niet toe.

Opmerking

De schakeling in figuur 6.18 is modulair ontworpen. De selectormodule is als standaard-component gegeven en het logisch ontwerp van de priority encoder is hierop afgestemd. Er kan ook voor een andere benadering gekozen worden, waarbij de selector en de priority encoder als één module ontworpen worden. Het blijkt dan dat de totale schakeling aanzienlijk vereenvoudigd kan worden. Ga dit na. In de praktijk is dit een van de nadelen van modulair ontwerpen. Bij grote series kan het aanbeveling verdienen de schakeling over de grenzen van de functionele modules heen te optimaliseren. □



Figuur 6.18. Opstelling met priority encoder.

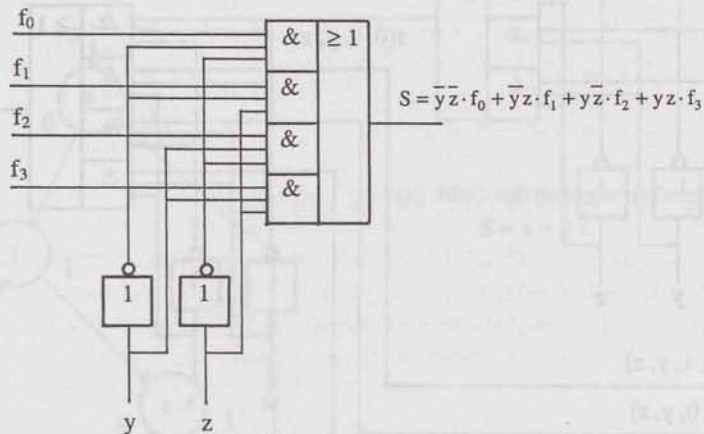
Hiermee hebben we enkele veel toegepaste componenten (logisch ontwerpniveau) voor de opbouw van systemen voor data-overdracht besproken. Er bestaat een veelheid van uitvoeringen, die elk op de hier besproken ontwerpen lijken, maar die geoptimaliseerd zijn naar oppervlak (uitvoering met pass transistoren van een selector bijvoorbeeld) of naar schakelsnelheid (op basis van AND-OR-INVERT poorten). Daarnaast bestaan er componenten voor speciale toepassingen, zoals een 1-uit-n decoder om te verhinderen dat bij een 3-state bus meer dan een zender actief is. Zie hiervoor de databoeken.

6.9 Decompositie van logische functies met selectoren

Logische functies kunnen direct op selectoren/multiplexers worden afgebeeld. Figuur 6.19 geeft een voorbeeld van hoe dit gaat bij een willekeurige functie in twee variabelen.

Afhankelijk van de waarde van de ingangsvariabelen y en z wordt bij elke minterm de bijbehorende functiewaarde f_i geselecteerd.

Voorbeeld



Figuur 6.19. Afbeelding van een functie in de mintermvorm op een selector. □

Het principe van figuur 6.19 kan worden uitgebreid tot functies in n variabelen. Daarbij loopt men op zeker moment tegen praktische beperkingen aan en wel voor wat betreft het toegestane aantal ingangen per poort. De *Shannondecompositie* van logische functies (paragraaf 2.8.) kan een uitweg bieden door de functie eerst in 'hapklare' brokken te decomponeren, om deze vervolgens af te beelden op selectoren.

Voorbeeld

Op grond van de Shannondecompositie kunnen we een functie in vier variabelen schrijven als

$$\begin{aligned} S &= f(w, x, y, z) \\ &= \bar{w} \cdot f(0, x, y, z) + w \cdot f(1, x, y, z) \\ &= \bar{w}\bar{x} \cdot f(0, 0, y, z) + \bar{w}x \cdot f(0, 1, y, z) + w\bar{x} \cdot f(1, 0, y, z) + wx \cdot f(1, 1, y, z) \end{aligned} \quad (6.21)$$

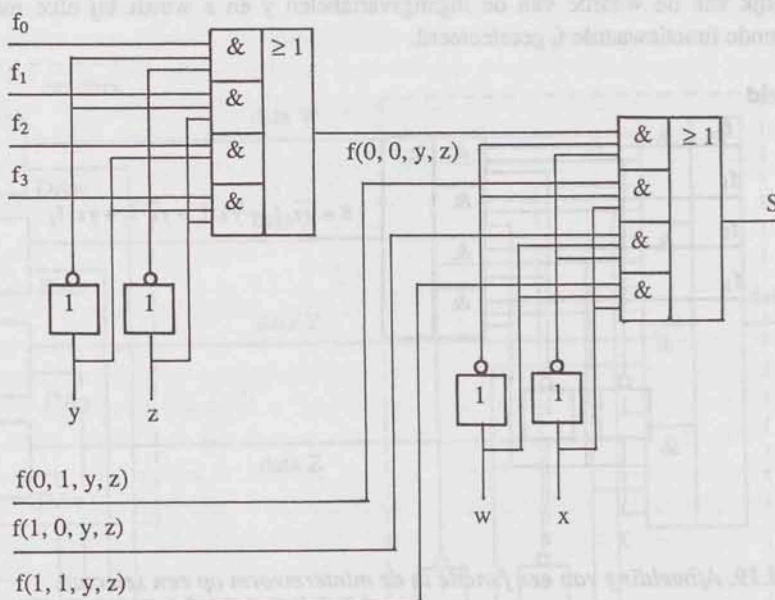
De subfuncties of residuen zijn hierbij functies in twee variabelen, die elk voor zich met de 4-input selector volgens figuur 6.19 kunnen worden gerealiseerd. Daarnaast kan ook de decompositie naar de variabelen w en x met eenzelfde selector gerealiseerd worden. Zie figuur 6.20. □

Zonder bewijs geven we de volgende stelling.

Stelling 6.1

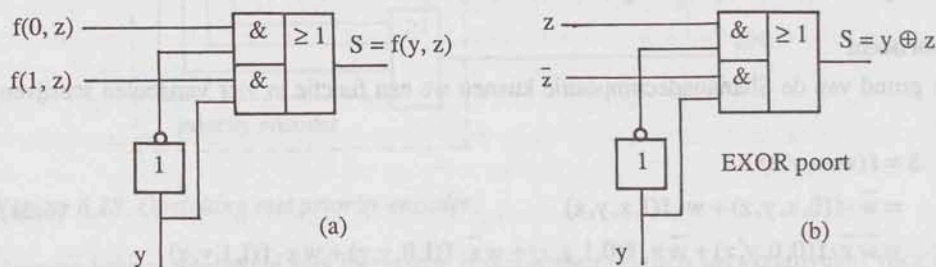
Een willekeurige logische functie S in n variabelen kan op een 2^{n-1} -input selector worden afgebeeld.

Het bewijs is triviaal. Zie ook het volgende voorbeeld. □



Figuur 6.20. Toepassing van de Shannondecompositie naar twee variabelen.

Voorbeeld



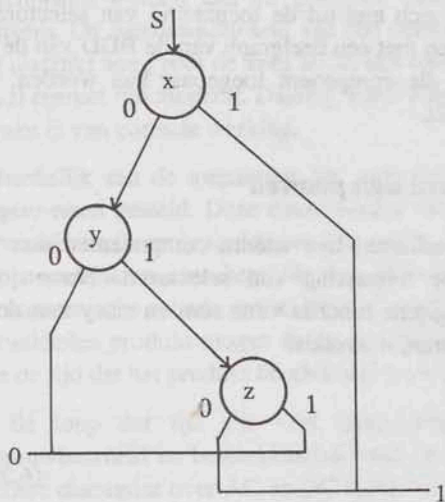
Figuur 6.21. Afbeelding van EXOR functie op een selector.

Figuur 6.21 geeft aan hoe de EXOR functie in twee variabelen op een 2-input selector kan worden afgebeeld. De selector maakt een decompositie naar de variabele y . De resulterende subfuncties zijn functies in één variabele en behoren dus tot de verzameling $\{0, 1, \bar{z}, z\}$. □

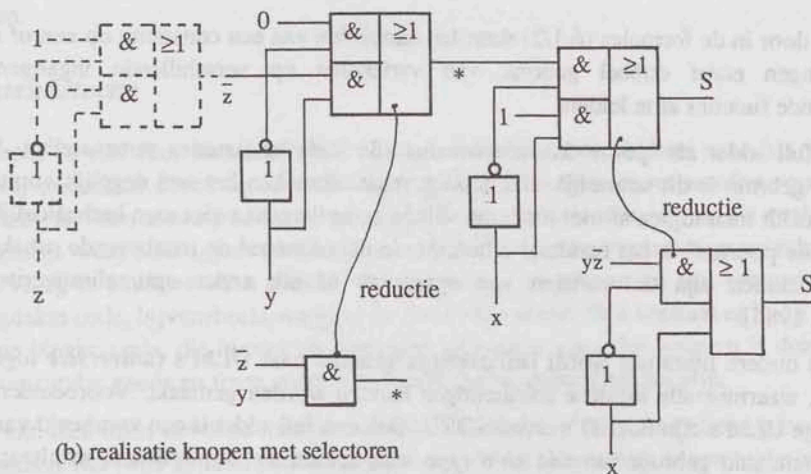
Minimale BDD's en logica met selectoren

Op de afbeelding van logica naar selectoren zijn diverse vereenvoudigingen mogelijk [Dormido, 1978; Pal, 1986; Whitehead, 1977]. De *minimale BDD vorm* van een logische functie (paragrafen 3.12 - 3.14) in n variabelen is hierbij van veel nut. BDD's kunnen direct

op 2-input selectoren afgebeeld worden door op de plaats van elk knooppunt in de graph van de BDD een 2-input selector te plaatsen. Zie figuur 6.22 als voorbeeld.



(a) BDD van de logische functie
 $S = x + y\bar{z}$



(b) realisatie knopen met selectoren

Figuur 6.22. Realisatie van $S = x + y\bar{z}$ met 2-input selectoren.

Zo'n realisatie van een functie via de minimale BDD en 2-input selectoren is niet noodzakelijk minimaal in de zin van het benodigde aantal transistoren/oppervlak. Zo vallen op grond van de stelling 6.1 de laatste selectoren in de 'boom' altijd weg omdat daar sprake is van een functie in één variabele. Die kan, zo hebben we eerder opgemerkt, zonder hardware worden gerealiseerd mits genegeerde variabelen beschikbaar zijn. Een op de BDD gebaseerde realisatie van een logische functie in n variabelen heeft dus (maximaal) $n-1$

niveaus van selectoren. Verdere vereenvoudigingen zijn mogelijk als het betreffende knooppunt van de BDD een variabele en/of een logische constante op een van de ingangen heeft staan. Waar dit toe leidt geeft figuur 6.22.b aan.

Uit dit voorbeeld concluderen we dat BDD's een uitstekend hulpmiddel zijn om logische functies te realiseren. Deze conclusie beperkt zich niet tot de toepassing van selectoren. Door de BDD van een component te vergelijken met een deelgraph van de BDD van de te realiseren functie kan uitgezocht worden of die component toegepast kan worden. In silicon compilers wordt hiervan gebruik gemaakt.

Logische functies in andere componenten dan poorten

De mogelijkheden om logische functies te realiseren met andere componenten dan de bekende poorten beperken zich niet tot de toepassing van selectoren. Nemen we bijvoorbeeld een 1-bit full adder. Naast de logische functies voor som en carry kan deze component ook andere logische functies realiseren, waaronder

$$\begin{aligned} S &= \overline{a_i \oplus b_i} \\ S &= a_i \oplus b_i \\ S &= a_i b_i \\ S &= a_i + b_i \\ S &= \overline{a} \end{aligned} \tag{6.22}$$

Ga dit na door in de formules (6.1/2) door het aanbieden van een constante op een of meer der ingangen en/of dubbel gebruik van variabelen op verschillende ingangen de vorenstaande functies af te leiden.

Met een full adder als 'poort' kunnen we dus alle logische functies samenstellen. Voor algemeen gebruik is dit natuurlijk niet handig, maar soms kan het wel degelijk voordelen bieden. Beeldt men logica af met moderne silicon compilers, dan ziet men herhaaldelijk dat er 'vreemde poorten' in het resultaat zitten. Mede daardoor zal de resulterende schakeling optimaal kunnen zijn ten aanzien van oppervlak of elk ander optimalisatiecriterium [Brayton, 1984].

In de wat oudere literatuur wordt herhaaldelijk gezocht naar ULM's (*universele logische modules*), waarmee alle logische schakelingen kunnen worden gemaakt. Voorbeelden van eenvoudige ULM's zijn NAND's en/of NOR's. Ook een full adder is een voorbeeld van een ULM. Uitsluitend gebruik van één zo'n type leidt zelden tot een optimaal resultaat, een combinatie ervan des te vaker. Zie verder [Yau, 1970] met gegeven referenties.

Naast de Shannondecompositie zijn nog andere decompositiemethoden in gebruik, waaronder de *Reed-Muller decompositie*. Deze methode maakt gebruik van de EXOR/complement EXOR poort. Van tijd tot tijd worden hiermee goede resultaten gerapporteerd. Voor literatuur hierover zie [Mukhopadyay, 1970; Saul, 1992].

6.10 Foutendetectie en foutencorrectie

Bij het ontwerpen van zowel basiscomponenten als grote digitale systemen moet aandacht geschonken worden aan de gewenste betrouwbaarheid van de component of van het systeem. De *betrouwbaarheid* van een component of van een systeem is een functie $R(t)$, die uitdrukt hoe groot de kans is dat een component of een systeem gedurende het interval $[t_0, t]$ correct functioneert. Daarbij wordt uitgegaan van het gegeven dat er op het tijdstip t_0 sprake is van correcte werking.

Afhankelijk van de toepassing (tv, auto-elektronica, pacemaker, satelliet) worden steeds hogere eisen gesteld. Deze eisen worden niet alleen afgeleid van veiligheidsaspecten. Ook economische factoren spelen een belangrijke rol. Zo is het tijdelijk uitvallen van een tv geen ramp, maar als van een bepaalde serie tv's een groot aantal (in de garantietermijn) uitvalt is de economische schade groot. Bij het opstellen van een programma van eisen voor een te ontwikkelen produkt mogen de gewenste betrouwbaarheid en beschikbaarheid (percentage van de tijd dat het produkt beschikbaar is) niet ontbreken.

In de loop der tijd zijn veel (toepassingsspecifieke) technieken ontwikkeld om de betrouwbaarheid en beschikbaarheid van een produkt te vergroten [Johnson, 1989]. Onze eerdere discussies over AC en DC noise margins en compatibiliteit (paragraaf 5.8) en over het dynamisch tijdmodel van poorten (paragraaf 6.7) passen in dit kader. We besteden nu aandacht aan enkele (logische) technieken om foutendetectie en foutencorrectie mogelijk te maken.

Foutendetectie

Wanneer is een fout detecteerbaar? Laten we daartoe eens kijken naar het voorbeeld waarbij een aantal dozen ter identificatie van een binaire code moet worden voorzien. Op de dozen zijn een aantal posities voorgedrukt, waarop al dan niet een rondje geplakt wordt (plakkertje = logische 1/geen plakkertje = logische 0). Is dit coderingssysteem robuust? Om deze vraag te beantwoorden moeten we nagaan wat er gebeurt bij een verstoring van de opgeplakte code, bijvoorbeeld wanneer er een rondje afvalt. Dan ontstaat er in dit geval een andere binaire code, die *binnen de gemaakte afspraken* valt. Het systeem is duidelijk niet robuust omdat goede en foute codes niet van elkaar te onderscheiden zijn.

Wat kan men doen om deze manier van coderen robuuster te maken? Een oplossing is het opplakken van twee kleuren plakkertjes, die verschillen naargelang er een logische 0 of een logische 1 moet worden aangeduid. Nu kan men altijd zien of er een plakkertje ontbreekt. Een ander idee is om het aantal plakkertjes te tellen en de som ervan extra toe te voegen. Dan kan men zien of het *aantal plakkertjes* klopt. Is deze manier van coderen 100% betrouwbaar? Wat wordt er wel en wat wordt er niet gedetecteerd? Een nadeel van deze handelwijze is dat het aantal plakkertjes of bits bij de binaire code toeneemt.

Maken we nu van plakkertjes logische nullen en enen, die als H en L niveaus worden overgezonden. Wanneer een signaalniveau verstoord wordt en niet meer voldoet aan de gespecificeerde grenzen voor H en L, dan zal de ontvanger er een logische 0 of 1 van

maken en wel naarmate het signaalniveau beïnvloed is. Bij de ontvanger ontstaat dan weer een binaire code. Zonder extra maatregelen zal de eventuele fout niet gedetecteerd worden.

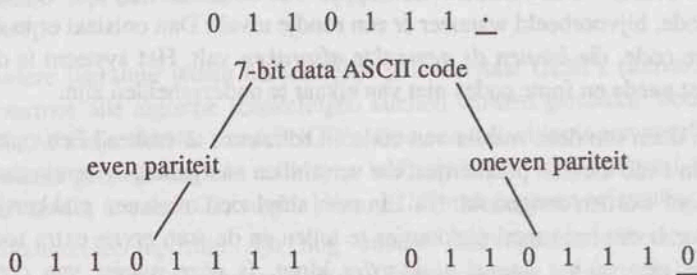
Extra maatregelen kunnen bestaan uit het *testen van het signaalniveau* van het ontvangen signaal. Ligt dit niet in de bedoelde logische niveaus, dan geeft de ontvanger een waarde X af. De ontvanger detecteert nu op {0, 1, X}. Wordt de waarde X ontvangen, dan kan door navraag worden nagegaan wat de juiste inhoud van de boodschap is. Er worden echter geen storingen gedetecteerd die een bepaald signaalniveau doen overgaan in het andere logische niveau. Met name als het signaal enige tijd wegvalt worden nullen gedetecteerd en lijkt alles in orde. De volgende maatregel zou kunnen zijn dat men *drie niveaus* op het transmissiekanaal toepast, een positieve en een negatieve spanning en een aanvullend nulniveau voor het geval er niet gezonden wordt. Komt tijdens het verzenden van een blok data het nulniveau voor, dan weten we zeker dat er een onderbreking is opgetreden. De hier genoemde maatregelen hebben invloed via het niveau van het ontvangen signaal. Voor de ontvanger kan men dan niet volstaan met een geheel logisch werkende schakeling.

Conclusies

- Opdat foutendetectie mogelijk is moet er een *kenmerk* zijn, waaraan we kunnen zien of bij transmissie en/of opslag van data al dan niet een fout is opgetreden. Vaak moet hiertoe de verzameling codecombinaties uitgebreid worden door het toevoegen van een of meer extra bits aan elk codewoord. Foutendetectie leidt aldus tot een zekere overhead in het aantal over te zenden of te registreren bits.
- Foutendetectie kost dus kanaal- of opslagcapaciteit. □

De parity check

Een veel gebruikte beschermingsconstructie is de *parity check*. Van elk codewoord worden de bits die 1 zijn geteld. Vervolgens wordt er één bit toegevoegd, dat het aantal van de bits die 1 zijn *even* of *oneven* maakt. Is bij ontvangst de som niet van de veronderstelde pariteit (even/oneven), dan is er een fout opgetreden.

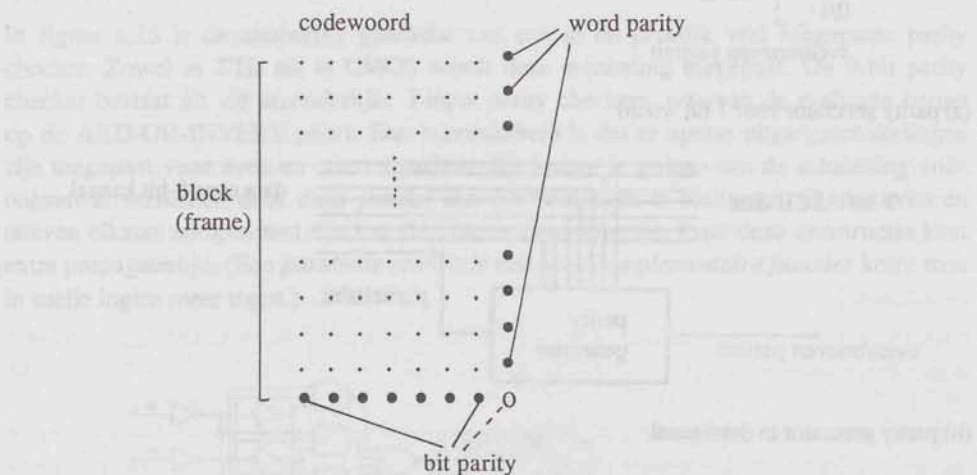


De karakters in de *ASCII code* hebben zeven bits. Deze code heeft als zodanig geen mogelijkheden voor foutendetectie. Met een achtste parity bit krijgt elk codewoord foutendetecterende eigenschappen. In een codewoord kunnen alle enkelvoudige fouten, alle drievoudige fouten, enz. gedetecteerd worden. Even aantallen fouten kunnen daarentegen

niet gedetecteerd worden. In de meeste systemen zal de kans op twee of meer fouten in een codewoord beduidend kleiner zijn dan de kans op geen of één fout. Daarnaast kan, als de kans op k fouten in een codewoord bekend is, uit het gevonden aantal fouten in een bericht de kans dat er fouten onopgemerkt blijven gemakkelijk berekend worden. Zodat impliciet een uitspraak gedaan kan worden over de betrouwbaarheid van een data-overdracht, inclusief de niet-gedetecteerde fouten.

Voorbeeld.

Een zekere extra beveiliging tegen onopgemerkte fouten kan worden bereikt met een *block* of *frame check*. Behalve per woord (horizontale parity) wordt dan ook een parity bit toegevoegd aan alle bits op eenzelfde positie in alle codewoorden van een bericht (verticale parity). Zie figuur 6.23.



Figuur 6.23. Block check.

□

Parity generators en parity checkers

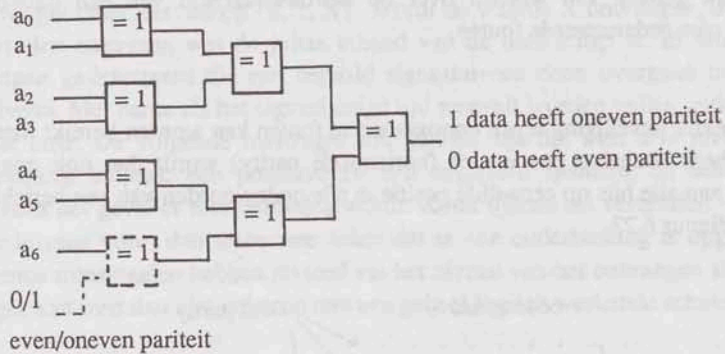
De even/oneven functie over twee bits heeft dezelfde waarheidstabel als de EXOR bewerking op twee variabelen. Zie tabel 2.8.b. Bij meer dan twee variabelen treedt een verschil op, omdat de *exclusieve OF* dan naar *precies één* 1 kijkt en de *even/oneven* functie naar een *even/oneven* aantal enen. Voor drie en meer variabelen kan de even/oneven functie echter wel in de EXOR bewerking op twee variabelen worden uitgedrukt als

$$S_{\text{even/oneven}} = ((x \oplus y) \oplus z) \quad (6.23.a)$$

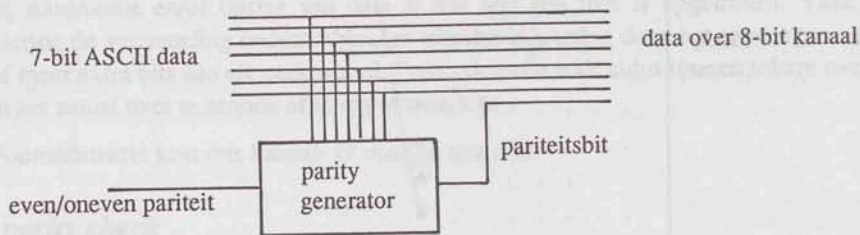
$$\begin{aligned} S_{\text{even/oneven}} &= (((w \oplus x) \oplus y) \oplus z) \\ &= ((w \oplus x) \oplus (y \oplus z)) \end{aligned} \quad (6.23.b)$$

$$S_{\text{even/oneven}} = (((v \oplus w) \oplus (x \oplus y) \oplus z) \quad (6.23.c)$$

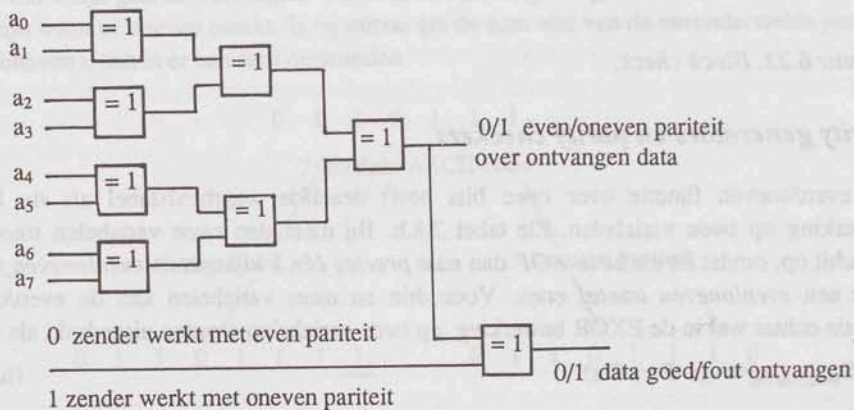
enz. Aldus vinden we de schakeling in figuur 6.24.a. Deze schakeling is geschikt als parity generator voor een 7-bit ASCII woord. Met de gestippelde poort kan ingesteld worden op even of op oneven pariteit. Hoe?



(a) parity generator voor 7-bit woord



(b) parity generator in datakanaal



(c) parity checker in ontvanger

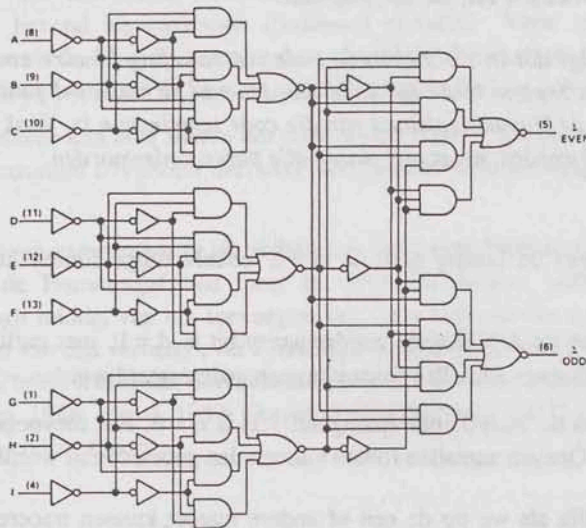
Figuur 6.24. Pariteitsbepaling over zeven bits.

In figuur 6.24.b is aangegeven hoe een parity generator in een 8-bit datakanaal wordt toegepast. Dezelfde schakeling wordt aan de kant van de ontvanger toegepast, maar nu met acht ingangen. Zie figuur 6.24.c. Daarmee wordt de pariteit over de acht ontvangen bits bepaald. Afhankelijk van de aan de zenzijde ingestelde pariteit, even of oneven, wordt het uitgangsbij van de parity checker omgezet in een goed/fout melding.

Opmerking

Bij een 16-bit woord kan de pariteitsbepaling worden gesplitst over twee keer acht bits. Afhankelijk van de foutkans en de aard van de fouten op een kanaal kan het dan zinvol zijn om de ene parity generator toe te passen op de bits met even rangnummer en de andere op de bits met oneven rangnummer. Bij random bitfouten maakt het niets uit, maar bij *burstfouten* wel. Burstfouten zijn foutpatronen waarvan bekend is dat de kans op een volgende fout direct na een andere groter is dan wanneer het voorafgaande bit goed is. □

In figuur 6.25 is de uitvoering geschetst van een in de praktijk veel toegepaste parity checker. Zowel in TTL als in CMOS wordt deze schakeling toegepast. De 9-bit parity checker bestaat uit vijf afzonderlijke 3-input parity checkers, waarvan de realisatie berust op de AND-OR-INVERT poort. Een bijzonderheid is dat er aparte uitgangsschakelingen zijn toegepast voor even en oneven pariteit. Dit laatste is gedaan om de schakeling voor oneven te versnellen door deze parallel aan die voor even te realiseren. Omdat even en oneven elkaars complement zijn kan dit ook met een invertor, maar deze constructie kost extra propagatietijd. (Een *parallele realisatie van twee complementaire functies* komt men in snelle logica meer tegen.)



Figuur 6.25. Uitvoering 9-bit parity checker SN74LS280 (Texas Instruments).

Hammingafstand en foutendetectie/correctie

De *Hammingafstand* (Hamming distance) is een maat voor het foutendetecterend en foutencorrigerend vermogen van een code [Hamming, 1950]. De Hammingafstand is als volgt gedefinieerd.

Definitie 6.1

Onder de Hammingafstand d van twee binaire codecombinaties met een gelijk aantal bits wordt verstaan het aantal bits waarin deze codecombinaties verschillen. $\square$

Voorbeeld

0	1	1	0	1	1	0	
		↓	↓	↓			$d = 3$
0	1	0	1	0	1	0	

Twee codecombinaties die in n bits verschillen kunnen door fouten dan en slechts dan in elkaar overgaan als er bij het verzenden van de ene codecombinatie n fouten zijn opgetreden en wel precies op de plaats van de verschillen met de andere codecombinatie. Dit inzicht heeft geleid tot het definiëren van het begrip Hammingafstand van een code.

Definitie 6.2

Onder de *Hammingafstand van een code* wordt verstaan de kleinste Hammingafstand welke tussen twee codewoorden van die code bestaat. $\square$

Uit het vorenstaande volgt dat in een bepaalde code (verzameling binaire codewoorden van gelijke lengte) tussen twee willekeurige codewoorden van de code $n-1$ fouten kunnen worden gedetecteerd als de Hammingafstand van die code tenminste n is. Vaak kunnen er meer fouten gedetecteerd worden, maar niet tussen alle paren codewoorden.

Voorbeelden

De Hammingafstand d van de binaire code is $d = 1$; enkelvoudige fouten kunnen niet gedetecteerd worden.

De Hammingafstand d van de ASCII code zonder parity bit is $d = 1$; met parity bit is de Hammingafstand $d = 2$. Oneven aantallen fouten kunnen gedetecteerd worden.

De Hammingafstand d van de 2-uit-5 code (paragraaf 1.2) is $d = 2$. Het toevoegen van een parity bit heeft geen zin! Oneven aantallen fouten kunnen dan gedetecteerd worden. $\square$

Foutencorrectie is mogelijk als we op de een of andere manier kunnen traceren wat het oorspronkelijke codewoord was. Nu is dit vaak niet te zeggen, een foutief ontvangen woord kan uit verschillende codewoorden ontstaan zijn. Nemen we aan dat de kans op een fout voor elk bit in een codewoord dezelfde is en onafhankelijk is van de andere fouten, dan is bij een Hammingafstand 2 niet te zeggen van welk codewoord het fout ontvangen woord

afkomstig is. Immers, het codewoord kan met een gelijke kans uit elk van de andere codewoorden met een onderlinge afstand 2 ontstaan zijn.

Voorbeeld

De 2-uit-5 code bevat 10 codecombinaties:

$$\begin{aligned} &\{11000; 10100; 10010; 10001; \\ &\quad 01100; 01010; 01001; \\ &\quad 00110; 00101; \\ &\quad 00011\} \end{aligned} \quad (6.24)$$

Wordt het 'codewoord' 1 0 0 0 0 ontvangen, dan kan dit codewoord door één fout ontstaan zijn uit

$$\{11000; 10100; 10010; 10001\}$$

en door drie fouten uit

$$\{01100; 01010; 01001; 00110; 00101; 00011\}.$$

□

Bij een minimale Hammingafstand $d=3$ ligt de situatie anders. Dan kan het foutief ontvangen codewoord ontstaan zijn door

- een fout in één codewoord;
- twee of meer fouten in andere codewoorden.

De kans dat het laatste optreedt is meestal (veel) kleiner dan die van het eerste geval. Men kiest dan voor het meest waarschijnlijke geval. In de literatuur staat deze wijze van decoderen bekend als *maximum likelihood decoding*. Voor systemen met een kleine foutkans wordt de kans op correcte data-overdracht hiermee sterk verbeterd. (Bedenk dat als de foutkans toeneemt de kans op een verkeerde beslissing ook toeneemt.)

In het algemeen kan men stellen dat voor het corrigeren van (maximaal) k fouten op basis van een maximum likelihood decoding een minimale Hammingafstand van $2k+1$ nodig is.

Het toevoegen van een parity bit verhoogt de minimale Hammingafstand d met (maximaal) één. Om de Hammingafstand meer te vergroten moeten andere oplossingen gezocht worden door middel van het toevoegen van extra bits aan een codewoord en/of door het aanbrengen van een verband over verschillende codewoorden. Hiervoor zijn in de loop der tijd verschillende methoden ontwikkeld. Het voert te ver om hier dieper op in te gaan. Zie [Berlekamp, 1968; Floyd, 1982; Garner, 1958; Peterson, 1972] en recente werken op het gebied van coding theory.

6.11 Samenvatting

In hoofdstuk 4 is voor enkele voorbeelden globaal het ontwerptraject van combinatorische logische schakelingen geschetst. In dit hoofdstuk hebben we als toepassing enkele MSI

logische schakelingen voor standaardbewerkingen ontwikkeld. Uit de gegeven voorbeelden volgen een aantal conclusies.

- De *weg van probleem naar schakeling* is bepaald niet eenduidig. Afhankelijk van te stellen criteria kan het ontwerp geoptimaliseerd worden naar kostprijs in termen van aantal logische poorten of transistoren of naar oppervlak, naar snelheid (aantal poortniveaus achter elkaar) of naar dissipatie. Daarnaast bestaat er in een technologie vaak een voorkeur voor bepaalde logische operatoren.
- Optimalisatiecriteria zijn vaak afhankelijk. Snel = meer parallel = meer oppervlak = meer dissipatie. Men ontkomt er niet aan ten aanzien van *optimalisatiecriteria* een *prioriteitsvolgorde* te stellen. In databoeken (LSI/MSI) vindt men daarom vaak verschillende uitvoeringen van een component. Daarbij moet men er tevens op letten dat oudere uitvoeringen conceptueel vaak achterhaald zijn. In het databoek staan ze dan alleen nog ter *documentatie voor 'vervanging defect'*. Voor nieuwe ontwerpen moet men andere types kiezen. Zo'n verwijzing naar een nieuw type ontbreekt vaak.
- Het *logisch ontwerpniveau* is een *tussenstap* naar het fysisch ontwerpniveau. De schakelalgebra en andere hulpmiddelen op dit niveau zijn *realisatie-onafhankelijk*. Technologiedetails zullen in een later stadium toegevoegd moeten worden, waarbij het niet onmogelijk is dat het logisch ontwerp alsnog aangepast moet worden.
- Het is een goed streven reeds op de hogere ontwerp niveaus in te spelen op de details van de lagere ontwerp niveaus. Zo wordt bijvoorbeeld de (maximale) invloed van de bedradingscapaciteit op de propagatietijd van poorten in het poortmodel vaak meegenomen.
- Door de introductie van het logisch, technologie-onafhankelijk, ontwerpniveau is het mogelijk grote stukken van het *ontwerptraject te systematiseren* en vervolgens te *automatiseren*. Zou men het hele ontwerptraject tegelijk automatiseren, dan is het aantal details te groot. Bij een top-down benadering blijft het ontwerpproces beheersbaar.
- In voorkomende gevallen kan een ontwerper door ervaring op hogere ontwerp niveaus *anticiperen op hetgeen op de volgende ontwerp niveaus aan de orde komt*. Een goed ontwerper maakt hier gebruik van. Ten aanzien van de tools betekent dit dat een *zekere mate van interactie* mogelijk moet zijn. Ontwerpen mag dan in hoofdrichting top-down verlopen, enige iteratieslagen komen vaak voor.

Het volgende hoofdstuk gaat in op het begrip 'structuur' in zowel de probleemstelling als in het logisch ontwerp als in de realisatie. Het beeld, dat in dit hoofdstuk van het ontwerpen van standaardcomponenten geschetst is, is verre van compleet. Allerlei zaken zoals de testbaarheid van een component ten aanzien van specifieke defecten bijvoorbeeld zijn buiten beschouwing gebleven. Met name de vraag of een bepaald ontwerp testbaar is of niet is een heel lastig probleem, dat een grondige kennis van de elektronica en de technologie vergt. In de paragrafen 4.11 tot en met 4.13 hebben we een klein tipje van de sluier opgelicht.

Op veel circuits treffen we verder twee soorten signalen aan, *datasignalen* en *instel-signalen*. Bij een full adder zijn alle ingangssignalen datasignalen, maar bij een selector onderscheiden we naast datasignalen ook aparte instel-signalen. Op de hogere

ontwerpniveaus zullen we dit onderscheid vaker maken, teneinde in een ontwerp de besturingsfunctie te kunnen scheiden van de (inhoud van de) data. Daardoor wordt het mogelijk de besturingsfunctie data-onafhankelijk te bestuderen.

Literatuur

1. R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Boston, 1984.
2. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
3. P. K. Chan, and M. D. F. Schag, *Analysis and Design of CMOS Manchester Adders with Variable Carry-Skip*, Proc. 9th Symp. on Computer Arithmetics, N.Y. 1989, pp. 86-96.
4. B. Dormido, and D. Cantoni, *Systematic Synthesis of Combinational Circuits Using Multiplexers*, Electronics Letters, Vol. 14, 1978, pp. 588-590.
5. T. L. Floyd, *Digital Fundamentals*, Ch. E. Merrill Publishing Cie, Columbus, 1982, pp. 231-236 and 574-579.
6. H. L. Garner, *Generalized Parity Checking*, IRE Trans. on Electronic Computers, Vol. EC-7, 1958, pp. 207-213.
7. R. W. Hamming, *Error Detecting and Error Correcting Codes*, Bell Syst. Tech. J., Vol. 29, 1950, pp. 147-160.
8. IBM Corporation, *Method for Fast-Carry Propagation for VLSI Adders*, IBM Techn. Disclosure Bulletin, Vol. 28, 1985, pp. 1593-1596.
9. B. W. Johnson, *Design and analysis of Fault-Tolerant Digital Systems*, Addison-Wesley Publ. Cie, Reading, Mass., 1989.
10. H. C. Lai and S. Muroga, *Minimum Parallel Binary Adders with NOR (NAND) Gates*, IEEE Trans. on Computers, Vol. C-28, 1979, pp. 648-659.
11. T. K. Liu, K. R. Hohulin, L. E. Shiau and S. Muroga, *Optimal One-Bit Full Adders with Different Types of Gates*, IEEE Trans. on Computers, Vol. C-23, 1974, pp. 63-70.
12. S. Majersky and M. Wiweger, *On the Determination of Optimal Distribution of Carry Skips in Adders*, IEEE Trans. on Electronic Computers, Vol. EC-16, 1967, pp. 45-48.
13. A. Mukhopadhyay, and G. Schmitz, *Minimization of EXCLUSIVE OR and LOGICAL EQUIVALENCE Switching Circuits*, IEEE Trans. on Computers, Vol. C-19, 1970, pp. 132-140.
14. M. Oberman, *Digital Circuits for Binary Arithmetic*, Macmillan, London, 1979.
15. A. Pal, *An Algorithm for Optimal Logic Design Using Multiplexers*, IEEE trans. on Computers, Vol. C-35, 1986, pp. 755-757.
16. W. W. Peterson, and E. J. Weldon, *Error-correcting Codes*, 2nd ed., John Wiley & Sons, New York, 1972.
17. J. Saul, *Logic Synthesis for Arithmetic Circuits Using the Reed-Muller Representation*, Proceedings EDAC 1992, pp. 109-113.
18. J. J. Shedletsky, *Comment on the Sequential Behavior of an End-Around-Carry Adder*, IEEE Trans. on Computers, Vol. C-26, 1977, pp. 271-272.
19. A. Sukarai and S. Muroga, *Parallel Binary Adder with a Minimum Number of Connections*, IEEE Trans. on Computers, Vol. C-32, 1983, pp. 969-976.
20. A. P. Thijssen, H. A. Vink and C. H. Eversdijk, *Digitale Techniek, Deel 2*, Delft University Press, Delft, 1995.
21. W. R. K. Tjon A Loi, *Six CMOS Full Adder Designs*, Technical Report 92-117, Network Theory Section, Fac. ET, Delft University of Technology, The Netherlands.
22. S. Turrini, *Optimal Group Distribution in Carry Skip Adders*, Proc. 9th Symp. on Computer Arithmetics, N.Y. 1989, pp. 96-103.
23. B. W. Y. Wei and C. D. Thompson, *Area-Time Optimal Adder Design*, IEEE Trans. on Computers, Vol. C-39, 1990, pp. 666-675.
24. D. G. Whitehead, *Algorithm for Logic-Circuit Synthesis by Using Multiplexers*, Electronics Letters, Vol. 13, 1977, pp. 355-357.
25. S. S. Yau, and C. K. Tang, *Universal Logic Modules and Their Applications*, IEEE Trans. on Computers, Vol. C-19, 1970, pp. 141-149.

Opgaven hoofdstuk 6

6.1. Bij het ontwerpen van optellers worden twee hulpfuncties toegepast, te weten de carry generate functie G_i ,

$$G_i = a_i b_i$$

en de carry propagate functie P_i ,

$$P_i = a_i \oplus b_i$$

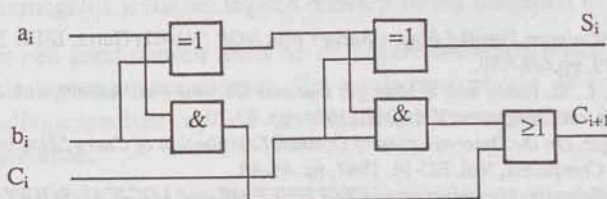
De carry propagate functie P_i mag eventueel vervangen worden door

$$P_i^* = a_i \oplus b_i$$

Dit mag omdat

- de carry generate functie dominant is over de carry propagate functie.
- de carry generate functie altijd het eerst gegenereerd wordt.
- $P_i^* = a_i \oplus b_i$ een snellere schakeling geeft.
- De reden is anders dan onder a t/m c genoemd is.

6.2.



De opbouw van een full adder schakeling is gegeven. Alle poorten ervan hebben een vaste propagatietijd van 3 ns. Voor het tijdsmodel van de schakeling volgens het globale tijdsmodel (willekeurige ingang naar willekeurige uitgang) geldt:

$$3 \leq \text{propagatietijd full adder} \leq 9 \text{ ns}$$

Wanneer we overgaan op een gedetailleerder tijdsmodel (individuele ingang naar individuele uitgang) kan de timing nauwkeuriger gespecificeerd worden van

- de somuitgang S_i .
- de carry-uitgang C_{i+1} .
- beide uitgangssignalen.
- geen van beide uitgangssignalen.

6.3. Figuur als bij opgave 6.2.

De opbouw van een full adder schakeling is gegeven. De propagatietijd van de poorten ervan ligt tussen

$$t_{P(\min)}(\text{poort}) \leq t_{PD}(\text{poort}) \leq t_{P(\max)}(\text{poort}) \quad (\text{tijden in ns})$$

waarbij $t_{P(\min)}(\text{poort})$ en $t_{P(\max)}(\text{poort})$ op de gebruikelijke wijze gedefinieerd zijn. Ook geldt

$$t_{P(\min)}(\text{poort}) \leq t_{P(\max)}(\text{poort})$$

De timing van de full adder wordt gemodelleerd volgens het globale tijdsmodel (willekeurige ingang naar willekeurige uitgang). Als de ingangssignalen van de full adder veranderen tussen $t_0 + 3$ en $t_0 + 12$ ns, hoe lang duurt dan het niet-gedefinieerde gebied aan de uitgangen?

- $9 + t_{P(\min)}(\text{poort}) + 2 \cdot t_{P(\max)}(\text{poort})$. (tijden in ns)
- $12 + t_{P(\min)}(\text{poort}) + 3 \cdot t_{P(\max)}(\text{poort})$.
- $9 - t_{P(\min)}(\text{poort}) + 2 \cdot t_{P(\max)}(\text{poort})$.
- $9 - t_{P(\min)}(\text{poort}) + 3 \cdot t_{P(\max)}(\text{poort})$.

6.4. Figuur als bij opgave 6.2.

Een 4-bit full adder wordt samengesteld uit vier stuks van de in de figuur beschreven full adder. Neem aan dat alle poorten hiervan een vaste propagatietijd $t_{PD}(\text{poort})$ hebben.

Wanneer alle bits a_i , b_i en C_0 tegelijk worden aangeboden, hoe lang duurt het dan voordat de uitgaande carry C_4 beschikbaar is? (Schat de propagatietijd zo nauwkeurig mogelijk.)

- Na $9 \cdot t_{PD}(\text{poort})$.
- Na $10 \cdot t_{PD}(\text{poort})$.
- Na $11 \cdot t_{PD}(\text{poort})$.
- Na $12 \cdot t_{PD}(\text{poort})$.

6.5. Druk C_4 van een 4-bit full adder uit in de P en G functies van de afzonderlijke secties. De hulpfuncties zijn gedefinieerd als $G_i = a_i b_i$ en $P_i = a_i \oplus b_i$. Over deze 4-bit full adder worden drie beweringen gedaan.

- C_4 kan uitsluitend 1 zijn indien $P_3 = 1$ is.
- C_0 heeft alleen invloed (wordt doorgegeven) op C_4 indien $P_3 = P_2 = P_1 = P_0 = 1$ is.
- Indien van twee naast elkaar liggende secties met rangnummer i en $i-1$ gegeven is dat sectie met rangnummer i een $G_i = 1$ genereert en sectie $i-1$ een $G_{i-1} = 0$, dan is het sombit S_i altijd 0.

Van deze beweringen is/zijn er

- geen juist.
- een juist.
- twee juist.
- drie juist.

6.6. Gegeven is de functietabel van een ALU.

Instelling $S_2 S_1 S_0$	Ingestelde bewerking	Resultaat
000	Reset	alle uitgangen 0
001	B minus A	afrekening; numeriek/in het two's complement
010	A minus B	afrekening; numeriek/in het two's complement
011	A plus B	optelling; numeriek/in het two's complement
100	A + B	logische OR
101	A $\oplus$ B	logische EXOR
110	AB	logische AND
111	Preset	alle uitgangen 1

Gegeven is verder de bit string A. Bepaal met de gegeven 'instructies' het complement C_A van de bit string A. Hoeveel instructies moeten hiervoor worden uitgevoerd? (Veronderstel dat de benodigde opslagcapaciteit voor tussenresultaten beschikbaar is.)

- Een.
- Twee.
- Drie.
- Meer dan drie.

6.7. Op de verzameling getallen V is een operator * gedefinieerd. Voor deze operator geldt de communicatieve wet als

$$y * z = z * y$$

de associatieve wet als

$$x * (y * z) = (x * y) * z$$

en de distributieve wet ten aanzien van de optelling als

$$x * (y + z) = x * y + x * z$$

(voor alle x, y en z $\in V$, + is de opteloperator.) Wanneer V de verzameling Z der gehele getallen is en * de vermenigvuldigoperator, dan gelden er van de drie genoemde wetten

- geen.
- een.
- twee.
- drie.

6.8. In de verzameling Z der gehele getallen worden de getallen gerepresenteerd in een talstelsel met grondtal g. Bij de vermenigvuldiging van twee gehele getallen

$$A = \sum_{i=0}^{n-1} a_i \cdot g^i$$

en

$$B = \sum_{j=0}^{m-1} b_j \cdot g^j$$

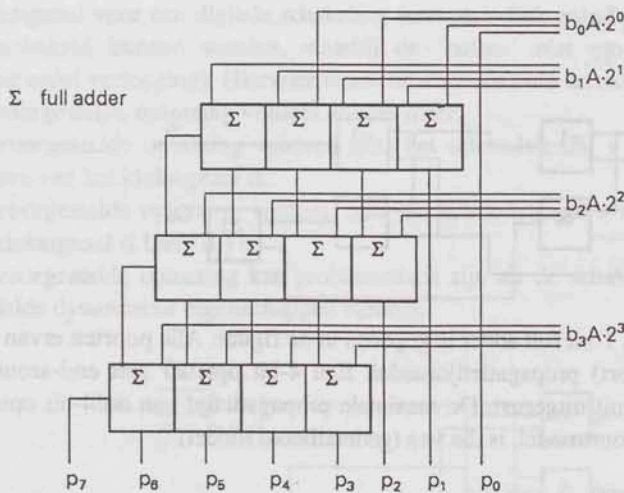
ontstaan partiële produkten $a_i b_j$. Het gewicht van zo'n partiële produkt is altijd een macht van het grondtal g. Bij het vermenigvuldigen van A en B ligt het gewicht van de partiële produkten $a_i b_j$ tussen (grenzen inbegrepen)

- $g^0 \leq \text{gewicht partiële produkt} \leq g^{m+n-2}$.
- $g^0 \leq \text{gewicht partiële produkt} \leq g^{m+n-1}$.
- $g^0 \leq \text{gewicht partiële produkt} \leq g^{m+n}$.
- andere grenzen dan onder a t/m c vermeld.

6.9. Zie de figuur op de volgende bladzijde.

De figuur beschrijft de opteller van een 4x4 bit parallelvermenigvuldiger. Hoeveel van de full adders (aangeduid met Σ) kunnen door half adders vervangen worden?

- Geen.
- Een.
- Drie.
- Vier.



6.10. Een 8x12 bit parallelvermenigvuldiger wordt opgebouwd uit 4x4 bit parallelvermenigvuldigers en een binaire opteller, bestaande uit 4-bit full adders. Het benodigde aantal ingangen per sectie van de binaire opteller (exclusief de interne ingangen voor het doorgeven van de transporten uit vorige secties) voor de diverse gewichten is respectievelijk

	$2^{19} \text{ t/m } 2^{16}$	$2^{15} \text{ t/m } 2^{12}$	$2^{11} \text{ t/m } 2^8$	$2^7 \text{ t/m } 2^4$	$2^3 \text{ t/m } 2^0$
a.	1	2	3	2	1
b.	1	4	6	3	1
c.	1	3	4	3	1
d.	1	4	6	4	1

6.11. Een optelschakeling moet negen bits met een gelijk gewicht bij elkaar optellen. De som geeft in de binaire code het aantal bits weer dat de waarde 1 heeft. Voor de opbouw ervan worden losse full adders toegepast. De schakeling wordt zodanig opgebouwd dat de doorlooptijd (propagation delay) van de signalen minimaal is, uitgedrukt in het aantal achtereenvolgens te doorlopen full adders. Het maximale aantal full adders dat de signalen achtereenvolgens moet doorlopen is

- twee.
- drie.
- vier.
- meer dan vier.

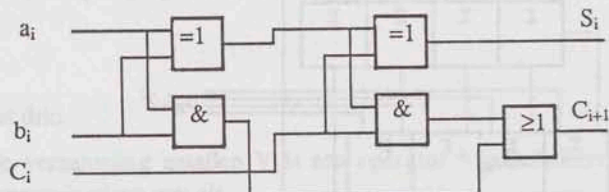
6.12. Een optelsectie voor de BCD code moet twee decimale cijfers $a_3a_2a_1a_0$ en $b_3b_2b_1b_0$ optellen. De sectie wordt gerealiseerd op basis van een 4-bit binaire full adder. De formule voor het uitgaande transport C_{uit} van deze optelsectie, uitgedrukt in de sombits $S_0 \text{ t/m } S_3$ en de carry C_4 van de binaire full adder, is

- $C_{uit} = C_4$.
- $C_{uit} = \overline{C_4}$.

c. $C_{uit} = C_4 + S_1 S_3 + S_2 S_3$.

d. $C_{uit} = C_4 + S_1 S_3 + S_2 \bar{S}_3$.

6.13.

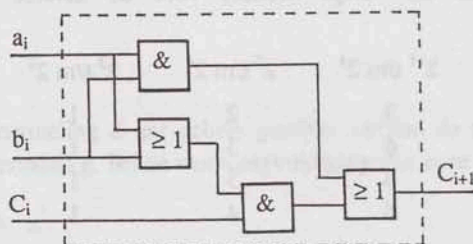


Het carrycircuit van een 1-bit full adder is gegeven in de figuur. Alle poorten ervan voldoen aan het $t_{P(min)}/t_{P(max)}$ (poort) propagatietijdmodel. Een 4-bit opteller met end-around carry wordt met dit carrycircuit uitgerust. De maximale propagatietijd van de 4-bit opteller, op basis van het gegeven poortmodel, is die van (gedetailleerd model)

- acht poorten.
- negen poorten.
- 17 poorten.
- meer dan 17 poorten.

(N.B. Alle a_i 's en b_i 's worden tegelijk aangeboden en veranderen daarna niet meer.)

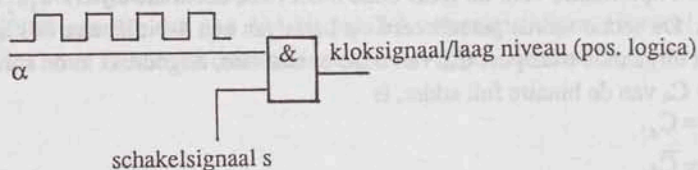
6.14.



Het carrycircuit van een 1-bit full adder is in de figuur gespecificeerd. Een 8-bit full adder met end-around carry wordt met dit carrycircuit uitgerust. Bij een end-around carry $C_8 \rightarrow C_0$ moeten, om de propagatietijd van de carry beperkt te houden, dynamische eisen gesteld worden aan

- een poort per sectie.
 - twee poorten per sectie.
 - alle poorten van een sectie.
 - De carrypropagatietijd kan in deze opstelling nooit gegarandeerd worden.
- (N.B. Alle a_i/b_i 's worden tegelijk aangeboden en veranderen daarna niet meer.)

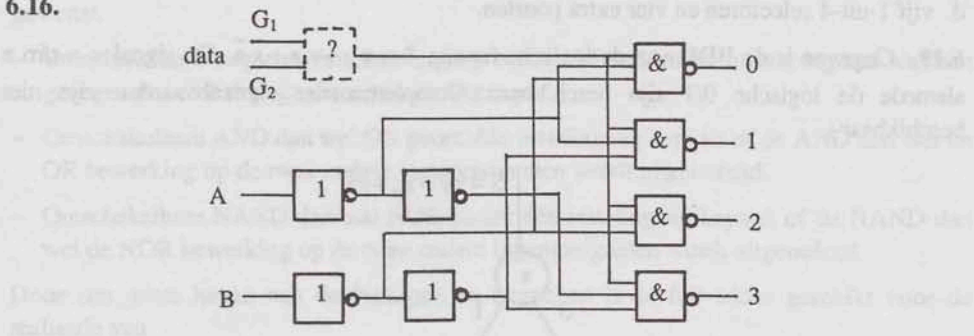
6.15.



Een kloksignaal voor een digitale schakeling (een periodiek pulsvormig signaal) moet aan- en afgeschakeld kunnen worden, waarbij de 'pulsen' niet mogen worden vervormd (inkorting en/of verlenging). Hiervoor wordt de vorenstaande schakeling voorgesteld.

- De voorgestelde oplossing voldoet zonder meer.
- De voorgestelde oplossing voldoet, mits het schakelsignaal s niet verandert rond de flanken van het kloksignaal α .
- De voorgestelde oplossing voldoet, mits het schakelsignaal s uitsluitend verandert als het kloksignaal α Laag (0) is.
- De voorgestelde oplossing kan problematisch zijn als de schakelpoort tevens niet aan bepaalde dynamische eigenschappen voldoet.

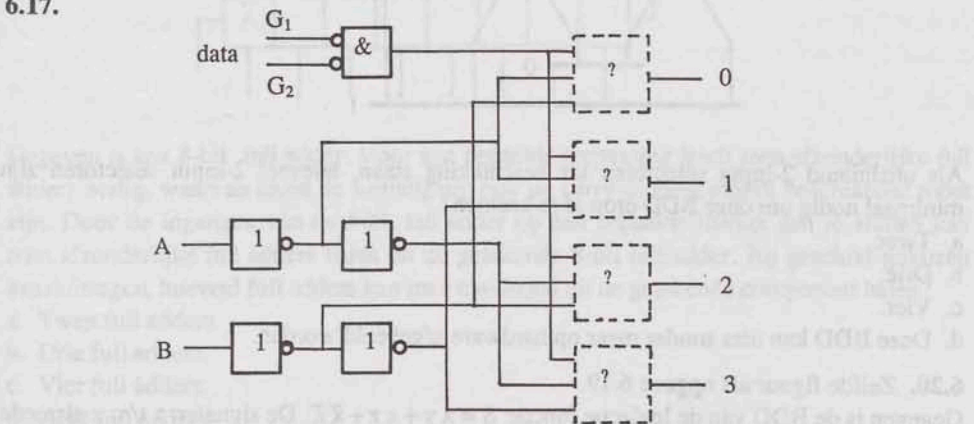
6.16.



Gegeven is het schema van een 2-line-to-4-line decoder/demultiplexer. Om de data op G_2 naar uitgang 1 te krijgen moet de demultiplexer ingesteld worden op:

- $ABG_1 = 011$ en de gestippelde poort is een AND.
- $ABG_1 = 010$ en de gestippelde poort is een NAND.
- $ABG_1 = 010$ en de gestippelde poort is een NOR.
- $ABG_1 = 101$ en de gestippelde poort is een NAND.

6.17.



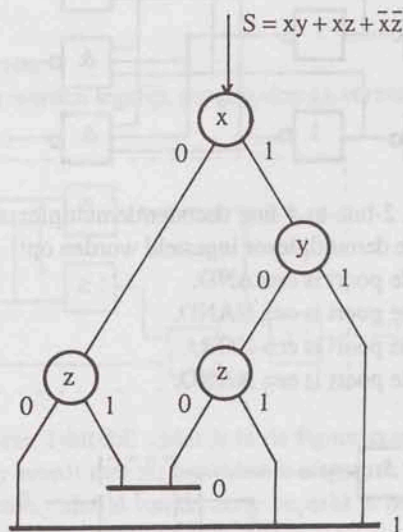
Gegeven is het schema van een 2-line-to-4-line decoder/demultiplexer. Om de data op G_2 naar uitgang 1 te krijgen (niet genegeerd) moet de demultiplexer ingesteld worden op:

- $ABG_1 = 010$ en de gestippelde poort is een AND en de overige uitgangen zijn dan 0.
- $ABG_1 = 010$ en de gestippelde poort is een NAND en de overige uitgangen zijn dan 1.
- $ABG_1 = 010$ en de gestippelde poort is een NOR en de overige uitgangen zijn dan 0.
- $ABG_1 = 101$ en de gestippelde poort is een NAND en de overige uitgangen zijn dan 1.

6.18. Voor een bepaalde toepassing is een 1-uit-16 selector nodig. Deze is niet beschikbaar. Wel zijn 1-uit-4 selectoren beschikbaar en zo nodig extra NAND poorten. De samen te stellen 1-uit-16 selector kan opgebouwd worden uit minimaal

- vier 1-uit-4 selectoren en een extra poort.
- vier 1-uit-4 selectoren en vier extra poorten.
- vijf 1-uit-4 selectoren.
- vijf 1-uit-4 selectoren en vier extra poorten.

6.19. Gegeven is de BDD van de logische functie $S = xy + xz + \bar{x}\bar{z}$. De signalen x t/m z alsmede de logische 0/1 zijn beschikbaar. Complementaire signaalwaarden zijn niet beschikbaar



Als uitsluitend 2-input selectoren ter beschikking staan, hoeveel 2-input selectoren zijn minimaal nodig om deze BDD erop af te beelden?

- Twee.
- Drie.
- Vier.
- Deze BDD kan niet zonder meer op hardware afgebeeld worden.

6.20. Zelfde figuur als opgave 6.19.

Gegeven is de BDD van de logische functie $S = xy + xz + \bar{x}\bar{z}$. De signalen x t/m z alsmede hun complement en de logische 0/1 zijn beschikbaar. Hoeveel 4-input selectoren zijn minimaal nodig om deze BDD erop af te beelden?

- Een.

- b. Twee.
- c. Drie.
- d. Vier.

6.21. De formules voor het somsignaal S_i en het transportsignaal C_{i+1} van een full adder zijn

$$S_i = (a_i \oplus b_i) \oplus C_i$$

$$C_{i+1} = a_i b_i + (a_i \oplus b_i) \cdot C_i$$

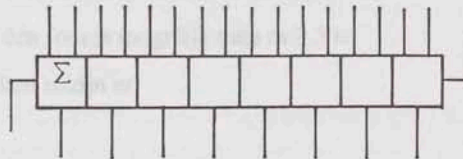
Nagegaan wordt of de full adder, door een juiste keuze van de in- en uitgangen ervan, geschikt is om er andere logische functies mee te realiseren. De volgende functies zijn gewenst.

- Omschakelbare negator: één instelingang bepaalt of een ander ingangssignaal wel/niet genegeerd wordt doorgegeven aan de uitgang.
- Omschakelbare AND dan wel OR poort: één instelingang bepaalt of de AND dan wel de OR bewerking op de twee andere ingangssignalen wordt uitgeoefend.
- Omschakelbare NAND dan wel NOR poort: één instelingang bepaalt of de NAND dan wel de NOR bewerking op de twee andere ingangssignalen wordt uitgeoefend.

Door een juiste keuze van de ingangen en uitgangen is de full adder geschikt voor de realisatie van

- a. geen van genoemde functies.
- b. een van de genoemde functies.
- c. twee van de genoemde functies.
- d. drie van de genoemde functies.

6.22.



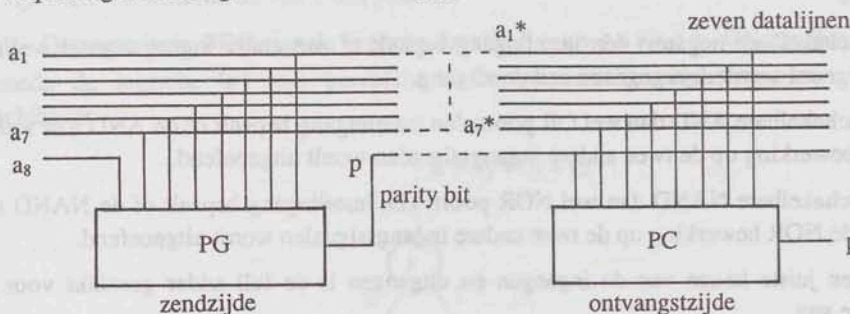
Gegeven is een 8-bit full adder. Voor een bepaalde toepassing heeft men afzonderlijke full adders nodig, waarvan naast de somuitgang ook de carry-uitgang extern beschikbaar moet zijn. Door de ingangen van de 8-bit full adder op een bepaalde manier aan te sluiten kan men afzonderlijke full adders halen uit de getekende 8-bit full adder. Bij geschikt gekozen aansluitingen, hoeveel full adders kan men maximaal uit de getekende component halen?

- a. Twee full adders.
- b. Drie full adders.
- c. Vier full adders.
- d. Acht full adders.

6.23. Onderstaande schakeling kan zowel als parity generator of als parity checker worden toegepast.



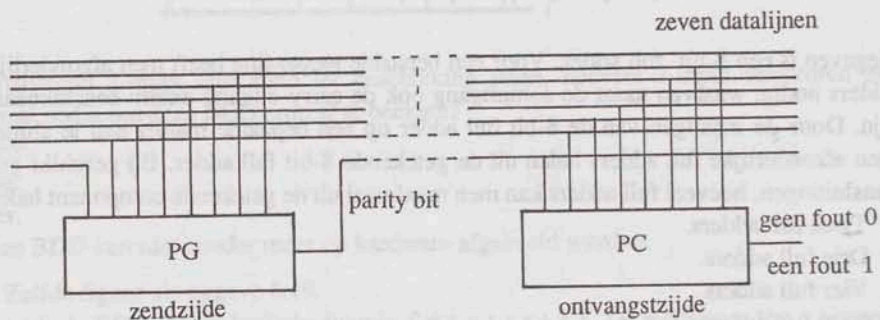
De toepassing is in de figuur hieronder aangegeven.



Bij de generator PG kan de ingang a_8 gebruikt worden om te kiezen tussen even en oneven pariteit. Bij de checker PC geeft de uitgang p^* aan of er een fout is opgetreden. De interpretatie van de waarde van p^* (goed/fout) is afhankelijk van de instelling van de generator (even/oneven pariteit). Geef aan welke waarde a_8 moet hebben bij gebruik van even pariteit en welke waarde p^* dan heeft als er één fout optreedt.

- $a_8 = 0$; $p^* = 0$.
- $a_8 = 1$; $p^* = 0$.
- $a_8 = 0$; $p^* = 1$.
- $a_8 = 1$; $p^* = 1$.

6.24.



Aan de signalen op de zeven datalijnen wordt een achtste bit toegevoegd volgens het recept van 'oneven pariteit' (odd parity). Aan de ontvangstzijde kunnen nu fouten gedetecteerd worden. Wanneer de met PC aangeduide schakeling aan de ontvangstzijde wordt

opgebouwd met uitsluitend EXOR poorten (let op: geen fout $\rightarrow$ 0 uit), dan kost deze schakeling minimaal

- a. vier EXOR poorten.
- b. vijf EXOR poorten.
- c. zeven EXOR poorten.
- d. acht EXOR poorten.

6.25. Gegeven zijn alle codewoorden van de 2-uit-4 code (codewoorden van vier bits waarvan er precies twee de waarde 1 hebben).

```

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

```

De Hammingafstand van de 2-uit-4 code is

- a. één.
- b. twee.
- c. soms twee, soms vier.
- d. vier.

6.26. Een m-uit-n code heeft foutendetecterende eigenschappen. Over deze code worden enkele uitspraken gedaan.

- Eén fout wordt altijd opgemerkt.
- Als m even is worden twee fouten altijd opgemerkt.
- Correctie van één fout is mogelijk mits $m \geq 3$ is.

Van deze uitspraken is/zijn er

- a. geen goed.
- b. één goed.
- c. twee goed.
- d. drie goed.

6.27. De Hammingafstand d in een gegeven code met k codewoorden is 1. Hoeveel bits moeten er minimaal aan elk codewoord toegevoegd worden om de Hammingafstand d = 2 te maken?

- a. Een.
- b. Twee.
- c. $2 \log k$.
- d. k.

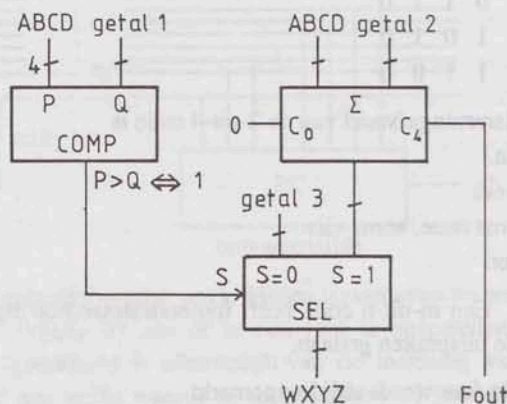
6.28. Gegeven is een 4-bit binaire code met 16 codewoorden. Deze code heeft van nature een (kleinste) Hammingafstand d = 1. Uit de uit 16 codewoorden bestaande code wordt een zo groot mogelijke deelverzameling geselecteerd, waarbinnen de Hammingafstand d = 2 is.

In deze deelverzameling is detectie van één fout mogelijk. Uit hoeveel codewoorden bestaat deze deelverzameling?

- Twée.
- Vier.
- Zes.
- Acht.

6.29. Een schakeling bestaat uit een 4-bit comparator, een 4-bit full adder en een 4-bit selector. Met deze schakeling wordt een code-omzetter met foutmelding gerealiseerd, $ABCD \rightarrow WXYZ$, die in de volgende tabel gespecificeerd is.

ABCD	WXYZ	Fout
0000	0000	0
0001	0001	0
0010	0010	0
0011	0011	0
0100	0100	0
0101	1011	0
0110	1100	0
0111	1101	0
1000	1110	0
1001	1111	0
1010	----	1
1011	----	1
1100	----	1
1101	----	1
1110	----	1
1111	----	1



Welke bitpatronen moeten worden aangeboden bij getal1, getal2 en getal3?

- | | | | |
|----|--------|--------|--------|
| | getal1 | getal2 | getal3 |
| a. | 0100 | 0110 | ABCD |
| b. | 0100 | 1011 | 0000 |
| c. | 1001 | 1011 | ABCD |
| d. | 0101 | 0110 | ABCD |

Stelling 6

Een logisch ontwerp is niet altijd logisch.

7

Het structureel ontwerpniveau

7.1 Probleemaanpak

Het oplossen van een probleem, of in ons geval het bedenken van een logische schakeling ervoor, verloopt vaak moeizaam. Wanneer de definitieve oplossing er eenmaal is dan wekt het verwondering dat het vinden ervan zolang geduurd heeft. Wie zich enig inzicht wil verschaffen in de werking van het menselijk brein bij het ontwerpen vindt een goed aanknopingspunt in de geschiedenis van optelschakelingen [Thijssen, 1995, hoofdstuk 4]. Ook een blik in oude databoeken, vooral ook op de datum van elk ontwerp, spreekt voor zich. In dit hoofdstuk houden we ons bezig met de *probleemaanpak*. In het voorafgaande hebben we gezien dat het verband tussen een probleemstelling enerzijds en de oplossing respectievelijk de schakeling ervoor anderzijds niet uniek is. Vaak zijn er voor een bepaald probleem verschillende oplossingen bekend, waaruit door een zorgvuldige evaluatie van de mogelijkheden en de eigenschappen de definitieve oplossing geselecteerd wordt. In dit hoofdstuk bespreken we enkele voorbeelden die laten zien hoe de *zoekruimte* voor oplossingen systematisch vergroot kan worden door naar bepaalde aspecten te kijken. Elk aspect wordt vervolgens algemener geformuleerd, zodat ook bij een andere probleemstelling ideeën kunnen ontstaan die tot alternatieve oplossingen leiden. Op deze wijze ontstaat een bepaalde *oplossingsruimte*, waarin eventueel (deel)oplossingen tegen elkaar uitgewisseld kunnen worden. Deze bezigheden resulteren in een oplossing of schakeling die optimaal aansluit bij de gestelde eisen. Binnen deze ontwerpactiviteiten speelt het begrip 'structuur' een belangrijke rol.

Onder structuur wordt verstaan de wijze waarop de onderdelen van een geheel ten opzichte van elkaar gerangschikt zijn. Deze rangschikking bepaalt mede de eigenschappen van het geheel in zijn omgeving of toepassing.

De structuur die in een bepaalde probleemstelling zit vormt vaak de sleutel tot het ontwerp van de optimale schakeling. Neem het optelalgoritme. Dit algoritme wordt gekenmerkt door een herhaling van dezelfde deelbewerking. In een schakeling leidt deze herhaling van bewerkingen tot de introductie van de full adder. Een realisatiestructuur met full adders wordt gekenmerkt door een ripple carry. Optellers met een ripple carry hebben daarom een lineair met het aantal bits/secties toenemende propagatietijd. Zie de paragrafen 4.2 en 6.2. Zo'n uit full adders opgebouwde opteller is een goed voorbeeld van hoe de keuze voor een bepaalde *interne structuur* doorwerkt in de *uitwendige eigenschappen* van de schakeling. Naast optellers met een ripple carry bestaan er ook optellers met een parallel over secties van vier bits opgebouwde carry. Zie paragraaf 6.3. Met enkele poorten extra versnelt de carrypropagatie aanzienlijk. Zo'n opteller heeft een *aangepaste interne structuur*.

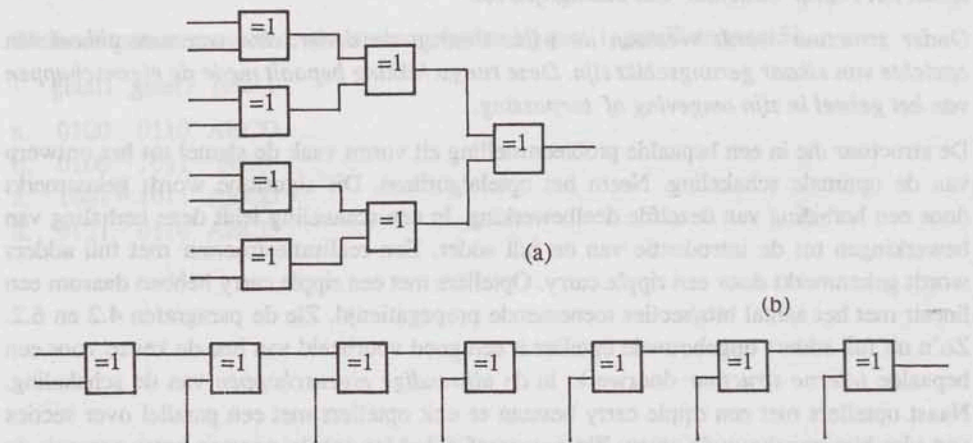
Dit voorbeeld van optelschakelingen geeft aan dat het zinvol is bij een ontwerp te zoeken naar *alternatieve structuren* in de probleemstelling, in het logisch ontwerp en/of in de realisatie van de schakeling. In het voorafgaande hebben we reeds vaak geprobeerd een zekere structuur in een ontwerp aan te brengen. De volgende trefwoorden duiden elk op een aspect van het begrip structuur in de context van het ontwerpen van logische schakelingen:

- interne/externe representatie van de variabelen,
- repeterende deelbewerking,
- modulariteit op logisch niveau,
- regelmaat in de lay-out,
- parallel/serie en serie/parallel conversie,
- volgorde van aanbieden van de te bewerken bits,
- richting van de informatie-overdracht in een schakeling,
- toepassing van hulpfuncties,
- bit- of woordgeoriënteerde aanpak bij optellers.

Veel van deze trefwoorden zijn we eerder in een bepaalde context al tegengekomen. Ga na wanneer en probeer al bladerend in de tekst de lijst aan te vullen met nieuwe trefwoorden. Het zal duidelijk zijn dat een zo compleet mogelijke lijst van trefwoorden bijdraagt tot het genereren van ideeën bij andere ontwerpen. Hiermee wordt de oplossingsruimte vergroot en daarmee de kans op het vinden van een goed ontwerp.

Bij het beoordelen van de voor- en nadelen van een bepaalde structuur moeten we alle relevante aspecten in beschouwing nemen. Daarbij speelt meer mee dan hetgeen we op het logisch ontwerpniveau tegenkomen.

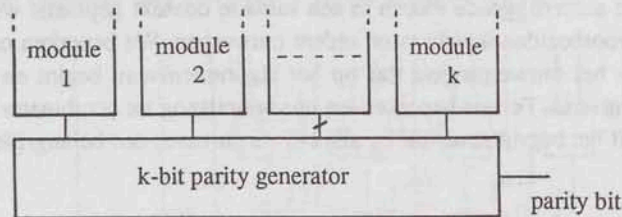
Voorbeeld



Figuur 7.1. Uitvoeringen van een 8-bit parity generator (logisch ontwerpniveau).

Een voorbeeld van hoe de geometrie van een schakeling de eigenschappen ervan beïnvloedt staat in figuur 7.1. De schakelingen in figuur 7.1 zijn ontworpen voor een 8-bit parity generator (logisch ontwerpniveau). Het is duidelijk dat de schakeling van figuur 7.1.a de snelste is (criterium: alle bits worden tegelijk aangeboden). Op basis van het poortgeoriënteerde delay model is de propagatietijd gelijk aan die van drie poorten. Voor de schakeling in figuur 7.1.b zijn het er (maximaal) zeven. Wanneer *maximale bewerkingstijd* een belangrijke parameter is dan verdient de structuur volgens figuur 7.1.a de voorkeur. Bij voldoende bewerkingstijd is er op grond hiervan geen voorkeur voor een van beide structuren.

Ook voor wat betreft de *gemiddelde dissipatie* komt de schakeling van figuur 7.1.a goed uit de bus. Bij de schakeling volgens figuur 7.1.b kan het voorkomen dat poorten aan de zijde van de uitgang bij veranderingen in de ingangsdata meer dan een keer schakelen. Een verandering van data dicht bij de uitgang leidt tot een uitgangsverandering. Zo'n verandering kan vervolgens door veranderingen van eerder in de boom aangeboden data ongedaan gemaakt worden. In CMOS levert het (herhaald) schakelen van een poort de grootste bijdrage aan de dissipatie. Bij het onderzoek naar dissipatie van combinatorische schakelingen speelt het interval waarin de ingangsdata verandert daarom een belangrijke rol. Bij een seriële structuur zou vaak veranderende data bijvoorbeeld dichter bij de uitgang aangeboden kunnen worden dan data die minder vaak verandert.



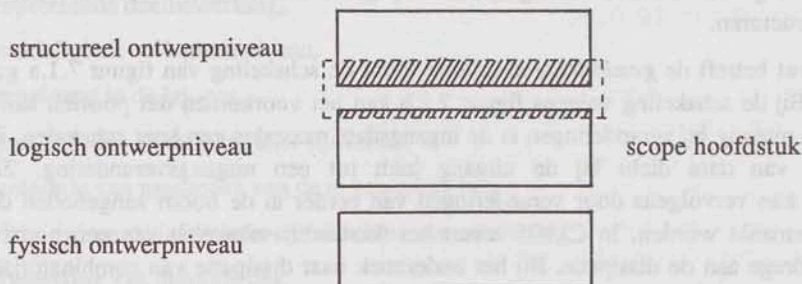
Figuur 7.2. Een parity generator in een lay-out.

Een ander beeld ontstaat als we ook de *lay-out* van andere deelschakelingen in de discussie betrekken. Zie figuur 7.2. Dan zou de opstelling volgens figuur 7.1.b beter kunnen scoren. Een parallel opgebouwde parity generator kost dan meer oppervlak dan een lineaire cascadering. Tevens levert ook de lengte van de verbindingen (de capacatieve belasting ervan) tussen de EXOR poorten een bijdrage aan propagatietijd en dissipatie.

Een ander aspect betreft de *omvang van de modules* waaruit de schakeling wordt opgebouwd. In figuur 7.1 is de basis voor de schakelingen een 2-bit EXOR poort. Figuur 6.25 laat een opbouw met 3-bit modules zien. De module-omvang kan van invloed zijn op het benodigde aantal transistoren en/of het benodigde oppervlak op een IC, alsmede op de propagatietijd en indirect ook op de dissipatie. □

Dit voorbeeld toont aan dat we bij het nemen van een beslissing over een te kiezen structuur zoveel mogelijk relevante aspecten moeten betrekken. Daarmee creëren we de oplossingsruimte, waarbinnen we vervolgens met de ontwerpeisen de definitieve oplossing selecteren. Is er geen oplossing mogelijk, dan zal er onderhandeld moeten worden over

hoeveel water er in de wijn mag, of er moet gezocht worden naar een andere aanpak. Alternatieve (realisatie)structuren geven vaak de aanzet tot een betere oplossing. Het voorbeeld toont ook aan dat we bij het nemen van ontwerpbeslissingen vaak een of meer ontwerpniveaus dieper moeten kijken dan het ontwerpniveau waarop we een keuze willen maken. Veel hangt af van de beschikbare speelruimte om tot een ontwerp te komen. Het verdient daarom aanbeveling om de *eisen aan een schakeling* in eerste instantie zo *ruim mogelijk* te formuleren.



Het doel van dit hoofdstuk is een handreiking te geven tot het verkennen van de oplossingsruimte voor een bepaald probleem. We doen dit aan de hand van voorbeelden. Vaak kunnen de achterliggende ideeën in een ruimere context geplaatst worden. Daarmee verschaffen de voorbeelden inzicht voor andere ontwerpen. We beperken ons in hoofdzaak tot het deel van het ontwerptraject dat op het algoritmeniveau begint en aansluit op het logisch ontwerpniveau. Tevens beperken we ons vooralsnog tot combinatorische logica. (In de praktijk speelt het begrip structuur op alle ontwerpniveaus een belangrijke rol.)

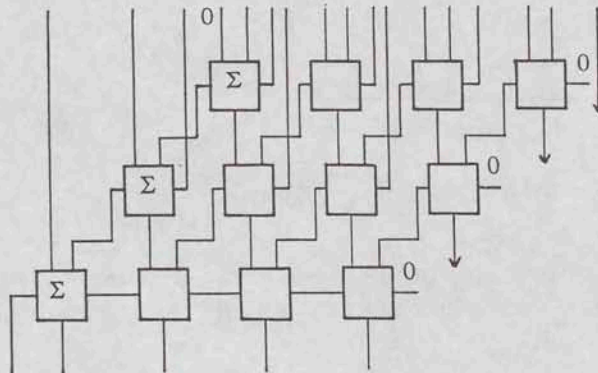
7.2 Structurele aspecten van de optelling

De tot nu toe gevolgde aanpak bij de optelling berust op de *repeterende structuur* in het optelalgoritme. Deze zien we terug in realisaties met een 1-bit modulaire structuur op basis van full adders en in realisaties met een 4-bit modulaire structuur op basis van 4-bit optellers met een parallelle carry-opbouw. Bepalend voor de *duur van de optelling* van twee binaire getallen zijn in eerste instantie het aantal bits ervan en de volgorde waarin de bits aangeboden worden (parallel/tegelijk of serieel in een bepaalde volgorde), het al dan niet toepassen van een techniek om de carrypropagatie te versnellen en verder technologiegebonden details zoals de toepassing van AND-OR-INVERT poorten en op het laagste ontwerpniveau de schakelsnelheid van de transistoren en de invloed van de layout. Dit alles resulteert in een *maximale* of *worst-case propagatietijd*. In schakelingen waartoe de opteller behoort moet rekening gehouden worden met deze maximale propagatietijd. Een optelling is gereed wanneer de maximale propagatietijd na het aanbieden van nieuwe data verstreken is. Vaak ligt de feitelijke opteltijd ver beneden de maximale propagatietijd. Anderzijds, een opteller hoeft ook niet sneller te zijn dan wat de omgeving kan verwerken.

Kenmerken van de operanden

Op deze aanpak zijn verschillende verfijningen mogelijk. Zo kunnen we kijken naar *relevante kenmerken van de operanden*. Wanneer bijvoorbeeld een van de twee operanden een *constante* is, dan is het mogelijk sommige full adders te vervangen door half adders en wel op die posities waar een 0 wordt aangeboden. We hebben deze mogelijkheid reeds aangestipt bij het ontwerp van de opteller van een parallelvermenigvuldiger. Zie paragraaf 6.5/6. Ook wanneer een van de operanden een kleiner bereik heeft dan de andere kunnen aan de voorzijde van de opteller half adders worden toegepast. Deze verfijningen resulteren in minder transistoren/oppervlak, in minder dissipatie en/of in een kortere bewerkingstijd. Anderzijds, als de opteller wordt opgebouwd uit standaard 4-bit of 8-bit full adders, dan zijn deze gegevens over de operanden minder relevant.

Andere verfijningen berusten op het *aantal getallen* dat bij elkaar opgeteld moet worden. Neem de vermenigvuldiging. Bij de vermenigvuldiging is er sprake van opeenvolgende tussenresultaten, de partiële produkten. Deze hebben als kenmerk dat het numerieke bereik ervan verspringt. Dit heeft enerzijds tot gevolg dat de opteller een dropstructuur (parallelogram) heeft omdat verschillende cellen in het optellerarray niet nodig zijn. Daarnaast loopt in een parallelvermenigvuldiger de optelling van de verschillende partiële produkten in de tijd door elkaar heen. *Op een bepaalde laag behoefde de optelling niet klaar te zijn voordat de volgende laag kan starten*. Ga dit in figuur 6.6 na.



Figuur 7.3. Carry-save opteller voor een vermenigvuldiger.

Een speciale vermelding verdient de optellerstructuur van figuur 6.10.a. Deze structuur is in figuur 7.3 gereproduceerd. In deze *carry-save opteller* wordt op de tussenliggende niveaus elk numeriek bit van de tussenresultaten weergegeven door twee bits, het sombit en het carrybit. *Voor de numerieke representatie van de tussenresultaten is overgegaan op een interne representatie met het dubbele aantal bits*. Door de aangepaste interne representatie is het op de tussenniveaus niet nodig de carry over de hele breedte van de opteller te verwerken. Dit wordt in figuur 7.3 tot op het laagste/laatste niveau uitgesteld. Vandaar de naam carry-save opteller [Avizienis, 1961; MacSorley, 1961; Lai, 1982].

Voor het oplossen van problemen rond een carrylooptijd in andere toepassingen kan mogelijk ook met vrucht gebruik gemaakt worden van het idee om een aangepaste interne representatie te kiezen. Uiteraard is dit slechts relevant als het een multi-operand optelling betreft. Denk tevens aan het two's complement, waarbij ten opzichte van teken-en-modulus representatie ook op een aangepaste interne representatie is overgegaan.

Worst-case of gemiddelde propagatietijd?

Veel bewerkingen zijn eerder gereed dan de maximale of worst-case propagatietijd suggereert. Neem bijvoorbeeld de opteller van een $n \times n$ bit parallelvermenigvuldiger. Deze is eerder gereed (worst-case) dan $n - 1$ keer de propagatietijd van de optelling van twee partiële produkten. Dit komt omdat in de tijd gezien in een parallelle optellerstructuur de verschillende bewerkingen *door elkaar heen* lopen.

Hoe zit dit als bijvoorbeeld de inhoud van twee arrays van k n-bit getallen *paarsgewijs* bij elkaar opgeteld moet worden met als resultaat één array met hun sommen? Op het eerste gezicht zit er niets anders op dan rekening te houden met k keer de maximale propagatietijd van één n-bit optelling. Daarbij gaat veel tijd verloren en wel het verschil tussen de maximale propagatietijd en de feitelijke of effectieve propagatietijd van elke optelling.

Voorbeeld.

$$\begin{array}{cccccccccccc}
 & & & & & & & & & & & \downarrow \\
 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & + \\
 \leftarrow & \leftarrow & & \leftarrow & & & & & & & & & \\
 2 & 3 & & 6 & & & & & & & & & \text{carrylengte} \\
 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \text{som/resultaat}
 \end{array}$$

Bij een 12-bit optelling is de *maximale propagatietijd* de propagatietijd van de carry over 12 secties (bij een 1-bit modulaire structuur). De *effectieve propagatietijd* in dit voorbeeld is die over zes secties. □

Bij dit voorbeeld moeten enkele kanttekeningen gemaakt worden. Allereerst wordt de propagatietijd over zes secties slechts gerealiseerd als er intern *geen overgangsverschijnselen* doorlopen over de andere secties. Zie paragraaf 6.7. Met een goed (dynamisch) ontwerp van de secties van de opteller kan hieraan voldaan worden. Daarnaast moeten we niet alleen rekening houden met de opbouw van de nieuwe carry. Het *weghalen van de oude carry* moet ook in rekening gebracht worden. In het voorbeeld is een bit met een pijltje aangegeven. Stel dat bij de volgende optelling alleen dit bit 0 wordt. De maximale lengte van de nieuwe carry is die over drie secties. Het weghalen van de *oude carry* heeft invloed over zes secties.

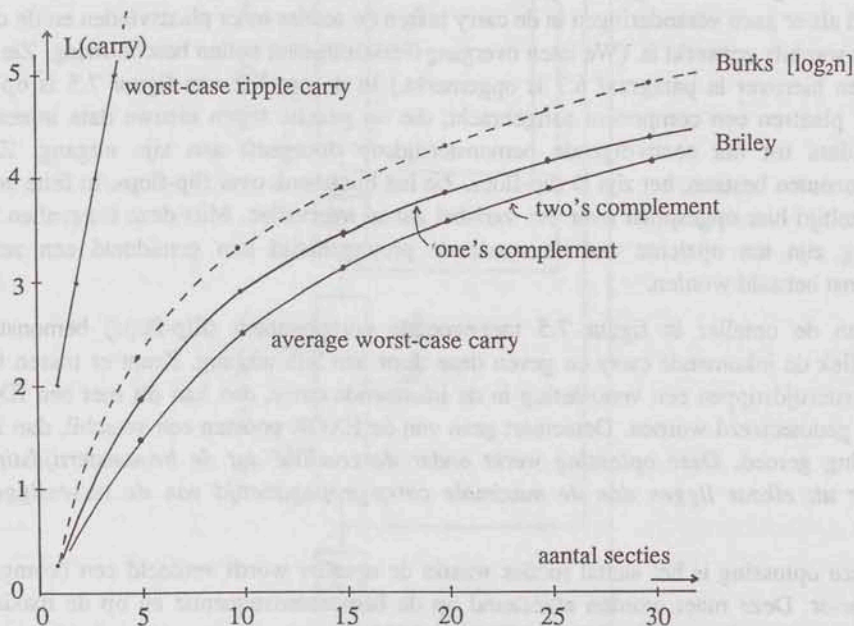
Het is een interessante vraag of het verschil tussen de maximale en de effectieve propagatietijd benut kan worden. Daartoe moet aan drie voorwaarden voldaan worden,

namelijk dat *het verschil de moeite waard is*, dat de *opteller tussentijds kan aangeven dat de optelling gereed is* en dat de *omgeving iets met deze gereedmelding kan doen*. Op de achtergrond speelt nog mee of dezelfde tijdwinst ook op een andere manier bereikt kan worden, bijvoorbeeld door de opteller te versnellen en aldus de worst-case propagatietijd te verkleinen.

De vraag of het verschil tussen worst-case en effectieve propagatietijd benut kan worden heeft bij het ontwerpen van computers reeds in een vroeg stadium een rol gespeeld. Uit de literatuur is bekend [Burks, 1946] dat bij het optellen van twee n -bit binaire getallen met een uniforme verdeling van de getallen over het bereik de *gemiddeld langste of average worst-case carry* (het gemiddelde van de langste carry over een groot aantal optellingen, uitgedrukt in het aantal full adder secties dat de carry doorloopt) evenredig is met $\log_2 n$

$$\text{average worst-case carry} \leq \log_2 n \quad (7.1)$$

keer de propagatietijd van een 1-bit full adder.



Figuur 7.4. Average worst-case carry-propagatietijd voor 1-32 bit optellers.

In [Briley, 1973] is de grens (7.1) wat scherper gesteld en wel

$$\text{average worst-case carry} \leq (\log_2 n - \frac{1}{2})(1 - \frac{1}{n}) \quad (7.2)$$

voor een *two's complement* opteller (geen verwerking van de end-around carry C_n) en

$$\text{average worst-case carry} \leq (\log_2 n - \frac{1}{2})(1 + \frac{1}{n}) \quad (7.3)$$

voor een *one's complement* opteller (verwerking van de end-around carry C_n naar C_0). (Over de tijd die met ontwerpen gemoeid kan zijn: let eens op het verschil in

publicatiedatum van beide referenties!) De afgeleide grenzen voor optellers van 1-32 bits zijn in figuur 7.4 uitgezet.

Voor een 32-bit opteller is de gemiddeld langste propagatietijd van de carry die van $\log_2 32 = 5$ secties volgens Burks en 4,29 secties volgens Briley. Vergeleken met een worst-case carry over 32 secties is er dus een *zeer grote tijdwinst* te behalen. Het is daarom zinvol te onderzoeken of een bewerking van een array van paren getallen op de een of andere wijze gebaseerd kan worden op de gemiddelde opteltijd, met een zekere toeslag voor de spreiding. Zeker bij grote array (beeldbewerking!) kan dit van belang zijn. Deze conclusie geldt uiteraard ook voor andere bewerkingen waarbij sprake is van een variabele tijdsduur en waarbij het verschil tussen maximale en gemiddelde duur de moeite waard is.

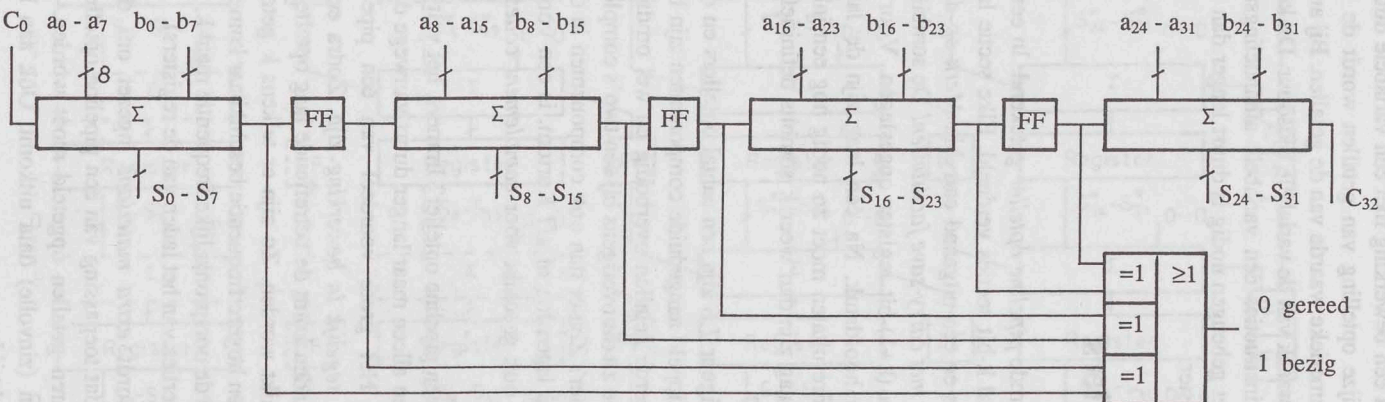
Carry-completion optellers

Resteert de vraag of het mogelijk is optellers uit te rusten met een voorziening waarmee de opteller een gereedmelding kan afgeven. Dit kan inderdaad. Zie figuur 7.5. Een optelling is gereed als er geen veranderingen in de carry tussen de secties meer plaatsvinden en de carry in alle sombits verwerkt is. (We laten overgangsverschijnselen buiten beschouwing. Zie ook hetgeen hierover in paragraaf 6.7 is opgemerkt.) In de opteller van figuur 7.5 is op een aantal plaatsen een component aangebracht, die op gezette tijden nieuwe data inleest en deze data tot het eerstvolgende bemonstertijdstip doorgeeft aan zijn uitgang. Zulke componenten bestaan, het zijn D flip-flops. Zie het hoofdstuk over flip-flops. In feite wordt de opteltijd hier opgesplitst over een variabel aantal intervallen. Mits deze intervallen kort genoeg zijn ten opzichte van de maximale propagatietijd kan gemiddeld een zekere tijdwinst behaald worden.

De aan de opteller in figuur 7.5 toegevoegde componenten (flip-flops) bemonsteren periodiek de inkomende carry en geven deze door aan hun uitgang. Komt er tussen twee bemonstertijdstippen een verandering in de inkomende carry, dan kan dit met een EXOR poort gedetecteerd worden. Detecteert geen van de EXOR poorten een verschil, dan is de optelling gereed. *Deze oplossing werkt onder de conditie dat de bemonstertijdstippen verder uit elkaar liggen dan de maximale carry-propagatietijd van de tussenliggende sectie.*

Bij deze oplossing is het aantal secties waarin de opteller wordt verdeeld een belangrijke parameter. Deze moet worden afgestemd op de bemonsterfrequentie en op de maximale carrylooptijd in de optelsectie tussen twee bemonsterende componenten. In de tussenliggende secties van de opteller kan overigens met vrucht gebruik gemaakt worden van technieken om de carry te versnellen. Daarnaast vergt het bemonsteren ook enige tijd. Een deel van de tijdwinst gaat daardoor verloren. Nader onderzoek moet uitwijzen welke verdeling in secties voor een concrete toepassing gunstig is.

Het is ook mogelijk optellers te ontwerpen die niet op discrete tijdstippen kijken of de optelling gereed is, maar die een continue gereedmelding kunnen afgeven. Zie [Gilchrist, 1955; Thijssen, 1995]. Daarmee kan een verdere versnelling worden bereikt. Een nadeel is dat in de meeste applicaties het signaal 'gereed' nog met de klok van de schakeling moet worden gesynchroniseerd. Dit geeft ook enig tijdverlies. Hierop komen we later terug.



Figuur 7.5. Carry-completion opteller.

Het idee door te gaan zodra een bewerking met een variabele duur gereed is wordt vaker toegepast. Bij de paarsgewijze optelling van getallen wordt de variabele duur van elke optelling bepaald door de numerieke waarde van de getallen. Bij andere toepassingen is het type van de bewerking de oorzaak van de variabele tijdsduur. Dit komt bijvoorbeeld voor in computers, waar de micro-instructies een variabele afhandelingstijd kennen. Een micro-instructie waarbij data uit het geheugen nodig is duurt langer dan het complementeren van de inhoud van een ALU register.

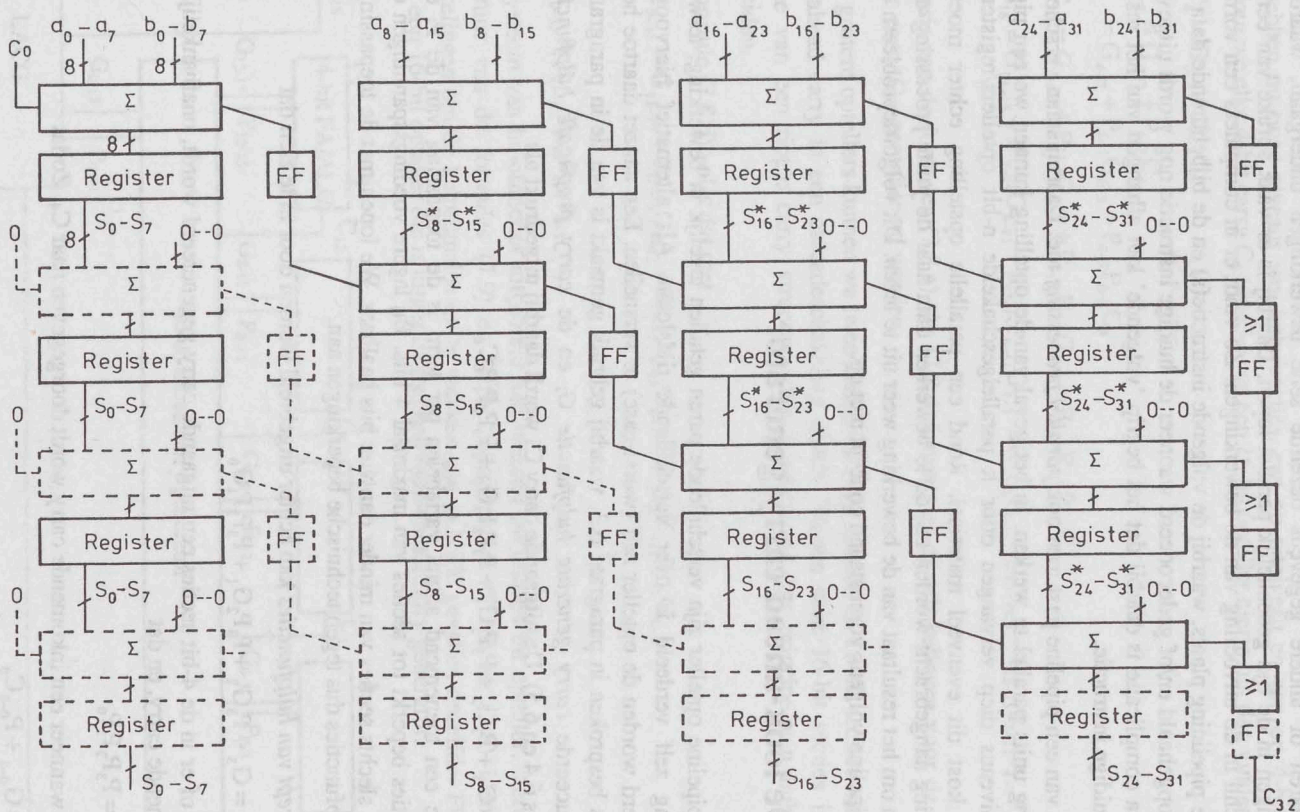
7.3 Pipeline optellers

In figuur 7.6 is een zogenaamde *pipeline opteller* getekend. In een pipeline opteller wordt de n -bit opteller in een aantal k -bit secties verdeeld. Elke sectie levert een tussenresultaat op, dat bestaat uit k sombits en een uitgaand carrybit. *Merk op dat we hier weer gebruik maken van een interne code met carry-save faciliteiten!* De sombits en het carrybit worden vervolgens per sectie in een $(k+1)$ -bit register opgeslagen. Voor de uitvoering van zo'n register zie het betreffende hoofdstuk. Na één laag zijn de laagstwaardige k sombits bekend. In de overige tussenresultaten moet zo nodig nog een inkomende carry verwerkt worden. Na elke volgende laag zijn dan weer k sombits definitief bekend, enzovoort. De optelling is gereed na k lagen.

In de pipeline opteller van figuur 7.6 zijn een aantal optellers en delen van tussenliggende registers gestippeld. De gestippeld aangeduide componenten zijn bij de optelling van twee niet-negatieve binair gecodeerde getallen overbodig en wel omdat de betreffende tussenresultaten definitief zijn. (Hoe zit dit overigens bij een two's complement opteller en hoe bij een one's complement opteller? Zijn er dan ook componenten in de tussenliggende secties overbodig? Neemt het aantal lagen toe of af? Kortom, is een voor niet-negatieve getallen ontworpen pipeline opteller ook geschikt voor *complement codes*? Een belangrijke vraag om bij stil te staan.)

Wat is nu het voordeel van een pipeline opteller? Immers, het valt eenvoudig in te zien dat de optelling van twee getallen alleen maar langer duurt vanwege de tijd voor het laden van de tussenresultaatregisters. Het grote voordeel van een pipeline opteller is dat er verschillende paren getallen *tegelijk in bewerking* zijn. Zodra een tussenresultaat in het onderliggende register is geladen komt de betreffende laag optellers weer vrij en kan voor de volgende optelling gebruikt worden. Zo zijn er telkens k getallenparen in bewerking, waarvan de resultaten met een hogere frequentie beschikbaar komen dan bij toepassing van één lange opteller (maximaal de oorspronkelijke frequentie maal k , waarbij we wel rekening moeten houden met het tijdverlies van het laden van de registers).

In een pipeline opteller wordt *extra materiaal* ingezet om de optelling effectief te versnellen. Het is duidelijk dat toepassing van een pipeline opteller alleen zinvol is indien een relatief groot aantal paren getallen opgeteld moet worden. Elke pipeline heeft een aanloopstuk, waar nog geen (zinvolle) data uitkomt. Ook aan het einde van een serie bewerkingen zit er enige wachttijd.



Figuur 7.6. Pipeline opsteller.

Uiteraard is de toepassing van een pipeline niet beperkt tot de optelling. In het algemeen verstaan we onder een pipeline een structuur of een constructie waarin verschillende getallenparen of andere gegevens dezelfde serie bewerkingen ondergaan, waarbij de bewerkingen in de tijd gedeeltelijk parallel lopen. Daarbij is er vaak sprake van een vast faseverschil in de uitvoering van de bewerkingen. Zo vindt er in computers een vorm van instructie pipelining plaats, waarbij de volgende instructie(s) en de bijbehorende data reeds worden opgehaald en/of gedecodeerd wanneer de huidige instructie nog wordt uitgevoerd. Een extra complicatie is daarbij dat het begrip 'volgende' kan afhangen van het resultaat van de huidige instructie.

In plaats van een pipeline past men ook *parallel processing* toe. Daarbij staan verscheidene processing units parallel te werken. In het geval van de optelling kunnen we een pipeline van k niveaus diep vervangen door k parallelgeschakelde n -bit optellers/registers. In principe kost dit evenveel materiaal. Rond een parallelle opstelling echter moet een voorziening aangebracht worden om de te bewerken data naar de juiste processing unit te sturen en om het resultaat van de bewerking weer uit te lezen. Dit volgordeprobleem treedt bij een pipeline volgens vorenstaand concept niet op.

7.4 De look-ahead carry generator

Bij een pipeline opteller zijn verschillende paren getallen tegelijk in bewerking, terwijl de bewerking zelf verdeeld is over verschillende tijdsloten. Als alternatief hiervoor kan geprobeerd worden de opteller zelf (worst-case) te versnellen. Een aanzet daartoe hebben we reeds besproken in paragraaf 6.3, waarbij gebruik gemaakt is van de in paragraaf 6.2 geïntroduceerde *carry generate hulpfunctie* G_i en de *carry propagate hulpfunctie* P_i (formules 6.4 en 6.5). De uitgaande carry C_4 wordt daarbij uitgedrukt als

$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0 \quad (7.4)$$

Vanwege een toenemend aantal variabelen per term is de toepassing van de P_i en G_i hulpfuncties beperkt tot secties van maximaal 4 bits. Bij lagere voedingsspanningen dan 5 Volt zijn slechts secties van minder dan vier bits haalbaar. We lopen met de toepassing van deze hulpfuncties dus tegen technische beperkingen aan.

Het concept van *hulpfuncties* kan echter uitgebreid worden door in te zien dat

$$G_{0-3} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 \quad (7.5)$$

aangeeft of er in de 4-bit secties een uitgaande carry gegenereerd wordt, onafhankelijk van de inkomende carry, en dat

$$P_{0-3} = P_3P_2P_1P_0 \quad (7.6)$$

aangeeft wanneer een inkomende carry wordt doorgegeven naar C_4 . Zodat

$$C_4 = G_{0-3} + P_{0-3}C_0 \quad (7.7)$$

Bedenk daarbij dat de formules voor G_{0-3} en P_{0-3} slechts uitgedrukt zijn in de ingangsvariabelen en dat G_{0-3} en P_{0-3} onafhankelijk zijn van enige interne carry. Zodra de variabelen

aangeboden zijn kunnen de hulpfuncties geheel onafhankelijk van de interne carry van de opteller berekend worden.

We hebben nu twee niveaus van hulpfuncties, een op bitniveau en een op 4-bit niveau. Drukken we elke 4-bit carry C_4 t/m C_{16} van een 16-bit opteller hierin uit, dan vinden we

$$C_4 = G_{0-3} + P_{0-3}C_0 \quad (7.8.a)$$

$$\begin{aligned} C_8 &= G_{4-7} + P_{4-7}C_4 \\ &= G_{4-7} + P_{4-7}G_{0-3} + P_{4-7}P_{0-3}C_0 \end{aligned} \quad (7.8.b)$$

$$C_{12} = G_{8-11} + P_{8-11}G_{4-7} + P_{8-11}P_{4-7}G_{0-3} + P_{8-11}P_{4-7}P_{0-3}C_0 \quad (7.8.c)$$

$$C_{16} = G_{12-15} + P_{12-15}G_{8-11} + P_{12-15}P_{8-11}G_{4-7} + P_{12-15}P_{8-11}P_{4-7}G_{0-3} + P_{12-15}P_{8-11}P_{4-7}P_{0-3}C_0 \quad (7.8.d)$$

Op basis van de formules (7.8.a) t/m (7.8.d) kunnen nu de schakelingen voor de carry C_4 t/m de carry C_{16} gerealiseerd worden.

Voor grotere optellers kunnen we twee kanten uit. We kunnen de 16-bit secties met interne look-ahead carry in een cascadeschakeling zetten. Tussen deze 16-bit secties is er dan sprake van een ripple carry structuur. We kunnen ook het principe van hulpfuncties uitbreiden.

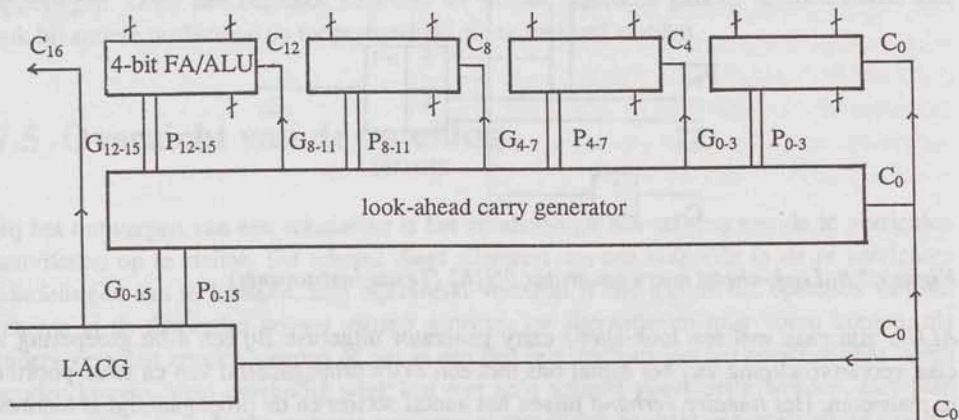
Uit formule (7.8.d) volgt dat met

$$G_{0-15} = G_{12-15} + P_{12-15}G_{8-11} + P_{12-15}P_{8-11}G_{4-7} + P_{12-15}P_{8-11}P_{4-7}G_{0-3} \quad (7.9)$$

en

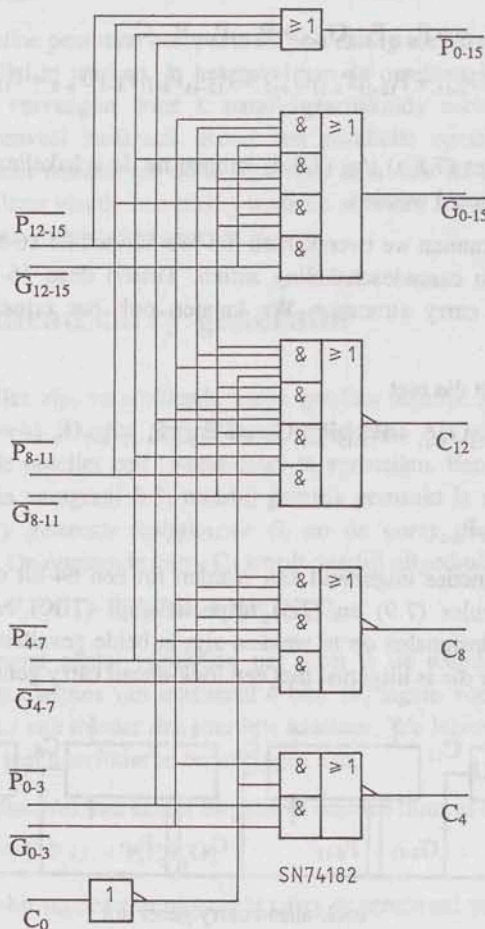
$$P_{0-15} = P_{11-15}P_{8-11}P_{4-7}P_{0-3} \quad (7.10)$$

het systeem van hulpfuncties uitgebreid kan worden tot een 64-bit opteller. Verder is de structuur van de formules (7.9) en (7.5) respectievelijk (7.10) en (7.6) identiek. De schakelingen om de hulpsignalen op te wekken zijn in beide gevallen dezelfde. Figuur 7.7 toont een 16-bit opteller die is uitgerust met een look-ahead carry generator.



Figuur 7.7. Opteller met look-ahead carry generator.

Zodra in figuur 7.7 de bits van de op te tellen getallen zijn aangeboden worden de signalen P en G van elke 4-bit sectie berekend. Dit kan in twee poortniveaus gerealiseerd worden. De *look-ahead carry generator* berekent uit de hulpfuncties en uit de inkomende carry C_0 de carries C_4 t/m C_{12} . Vervolgens berekent elke 4-bit sectie zijn sombits. Voor een uitbreidbaar systeem moet de look-ahead carry generator ook de hulpfuncties G_{0-15} en P_{0-15} berekenen. Voor een niet-uitbreidbaar systeem moet C_{16} berekend worden. Zie ook het ALU-schema in figuur 6.5. Een bipolaire uitvoering van de look-ahead carry generator staat in figuur 7.8. We leiden de formules hiervan niet af.



Figuur 7.8. Look-ahead carry generator SN182 (Texas Instruments).

ALU's zijn vaak met een look-ahead carry generator uitgerust. Bij een 4-bit groepering is elke verviervoudiging van het aantal bits met een extra propagatietijd van ca twee poorten te realiseren. Het *lineaire verband* tussen het aantal secties en de propagatietijd is hiermee doorbroken. Daardoor zijn snelle optellers te realiseren.

Het concept van *hulpfuncties* is vaak toepasbaar in schakelingen waarin een carry of een andere vorm van data-overdracht als een ripple door de schakeling gaat. Neem bijvoorbeeld een comparator, waarmee twee n -bit getallen met elkaar vergeleken worden op $>$, $=$ of $<$. Het resultaat van elke vorige vergelijking van twee bits levert samen met het momentele tussenresultaat een nieuw tussenresultaat op, enzovoort. Zo'n tussenresultaat kan ook over vier of over een ander aantal bits berekend worden. Tevens is een vorm van parallellisme mogelijk. Ga dit voor een binaire comparator na.

Opmerking

Het belang van technieken om de carry te versnellen was zeer groot in de tijd dat de schakelsnelheden relatief laag waren. Bij de huidige hoge schakelsnelheden zal snelheid bij steeds minder toepassingen de bottle neck zijn. Voor deze toepassingen kan een opteller/ALU met een standaard silicon compiler afgebeeld worden op silicium. Anderzijds komen steeds meer toepassingen in zicht, bijvoorbeeld in de beeldbewerking, die eerder door een onvoldoende hoge snelheid niet mogelijk waren. Dan kan een aangepast optellerontwerp wel zinvol zijn. Bij de huidige generatie silicon compilers kan het problemen geven om een specifieke structuur op silicium te genereren. $\square$

Het vorenstaande leidt tot de conclusie dat er diverse manieren zijn om de optelling te versnellen. Het is in feite een geringe greep uit alle bekende mogelijkheden. Zo bestaan er varianten op de hulpfuncties voor een look-ahead carry generator [Doran, 1988]. Het aantal publicaties over optellen en optellers is enorm groot. Zie voor een overzicht [Thijssen, 1995]. Veel publicaties hebben betrekking op het structurele ontwerpniveau, andere behandelen de uitwerking van een principe of structuur binnen een bepaalde technologie-omgeving. Zo zijn er optellerstructuren bekend waarbij er in secties van k bits twee keer opgeteld wordt, een keer met de inkomende carry $C_i = 0$ en een keer met $C_i = 1$. Met de inkomende carry wordt daarna een van de twee antwoorden geselecteerd [Bedrij, 1962]. Op ditzelfde principe berusten tabelmethodes voor code-omzetters zoals sinus-cosinus omzetters. Ook dan zijn de conversies van te voren uitgerekend en in tabelvorm opgeslagen. Door een bepaalde structuur of aanpak algemeen genoeg te formuleren kan ook bij andere problemen de toepasbaarheid ervan herkend worden.

7.5 Overzicht van de optelling

Bij het ontwerpen van een schakeling is het noodzakelijk een schema van de te verrichten activiteiten op te stellen. Dit schema dient allereerst om een *volgorde in de te verrichten handelingen* aan te brengen. Een bijkomend voordeel is dat tijdens het opstellen van het schema in de discussies vrijwel vanzelf aspecten en alternatieven naar voren komen, die anders vergeten zouden worden of pas in een (te) laat stadium van het ontwerp naar voren zouden komen. Veel overbodig werk kan met zo'n schema voorkomen worden. Een plan van aanpak bevat in eerste instantie ook meer aspecten dan er strikt genomen nodig zijn om het probleem te kunnen oplossen. Er moet dan een nadere selectie gemaakt worden. Dit is

een essentieel onderdeel van het ontwerptraject. Daaruit resulteert een definitieve planning van het af te lopen ontwerptraject. Hoe zou zo'n schema er voor de optelling uitzien?

Schema optelling / Domein en representatie van de getallen

Optellen is een rekenkundige bewerking. Voordat we de schakeling gaan ontwerpen moet duidelijkheid bestaan over een aantal zaken die te maken hebben met het 'domein' van de te bewerken getallen, in relatie tot de toepassing. Trefwoorden hierbij zijn getallenverzamelingen, getalrepresentaties, aantal te bewerken getallen, bereik, nauwkeurigheid en eventuele probleem-specifieke aspecten.

Getallenverzameling

De te bewerken getallen behoren tot een bepaalde getallenverzameling. Deze verzameling wordt onder andere gekenmerkt door de rekenregels die erin van toepassing zijn. Welke zijn deze regels? Daarnaast kan de verzameling waartoe het antwoord behoort een andere zijn. Vergelijk de deling van integers. Moet de getallenverzameling dan uitgebreid worden of moet het antwoord door afronding in de oorspronkelijke getallenverzameling afgebeeld worden?

Getallenrepresentatie

Binnen een bepaalde getallenverzameling kunnen getallen op verschillende manieren gerepresenteerd worden. Voorbeelden voor (niet-negatieve) integers zijn de binaire code, de BCD code, het two's complement, het one's complement, de floating point representatie, enzovoort. Verder kan er een verschil bestaan tussen de externe representatie van de operanden en/of het resultaat enerzijds en de interne representatie in de schakeling anderzijds. Vergelijk de toepassing van de carry-save representatie in optellers voor parallelvermenigvuldigers.

Aantal en soort van de getallen

We hebben gezien dat het bij optellers uitmaakt of er twee of meer dan twee getallen bij elkaar opgeteld moeten worden, of dat er array van getallen bewerkt moeten worden. Bij een verzameling van twee getallen is meestal een worst-case gebaseerde aanpak aan te bevelen. Bij bewerkingen op array kan ook een op de gemiddelde bewerkingssduur gebaseerde aanpak overwogen worden, een pipeline structuur of een parallelschakeling van verschillende bewerkingseenheden.

Bereik en nauwkeurigheid.

In digitale schakelingen wordt gewerkt met een beperkt aantal bits voor de getalvoorstelling. Dit heeft gevolgen voor het bereik en de nauwkeurigheid van de operanden en de resultaten van een bewerking. Overschrijding van het bereik moet gedetecteerd kunnen worden (capaciteitsoverschrijding) of het bereik moet van tevoren op de benodigde capaciteit afgestemd zijn. Een groter bereik heeft gevolgen voor de snelheid van de bewerkingen door het grotere aantal te bewerken bits.

Afronden is een noodzakelijk onderdeel van veel bewerkingen. Het kan daarbij verschil uitmaken of alleen het eindresultaat in afgeronde vorm wordt gepresenteerd, of dat er ook tussentijds wordt afgerond. Vaak past men intern een vergroot bereik toe om een gewenste nauwkeurigheid bij het eindresultaat te kunnen garanderen.

Probleem-specifieke kenmerken

We hebben gezien dat het bij de realisatie van optellers van belang kan zijn of een van de operanden een constante is. Zijn er meer van zulke probleem-specifieke kenmerken? Zo is het bijvoorbeeld zinloos om met zeer grote nauwkeurigheid te rekenen als in de operanden reeds een grote mate van onzekerheid of spreiding zit. Met een schijnnaauwkeurigheid is niemand gebaat.

Schema optelling / Algoritmekeuze

De vorenstaande punten en overwegingen vormen de basis voor het op te stellen algoritme. Een algoritme is een serie van voorschriften en handelingen, volgens welke het probleem kan worden opgelost. Bij het opstellen van een algoritme komen onder meer de volgende trefwoorden aan de orde: volgorde van de bits bij het aanbieden/afnemen van data, richting van de interne data-overdracht (carry), modulaire structuur, bit- of woordgeoriënteerde aanpak, bewerkingstijd vast/worst case of variabel, direct de data bewerken of een tabelmethode toepassen.

Volgorde aanbieden van data.

De volgorde waarin de data wordt aangeboden/afgenomen kan van invloed zijn op de keuze van het algoritme. Passen we bij optellen een bit-seriële data-overdracht toe dan verdient het, gezien de richting van de carry, aanbeveling het laagstwaardige bit het eerst over te zenden. Anders dient in het algoritme een of andere vorm van data-opslag voorzien te worden voordat de optelling kan starten.

Modulaire structuur

De optelling kent van nature een 1-bit modulaire structuur. We hebben gezien dat er intern in optellers meestal sprake is van een 4-bit modulaire structuur. Tussen de 4-bit modules kan het informatietransport op basis van een ripple carry plaatsvinden. Ook kan het informatietransport via een aparte structuur lopen, zoals bij de look-ahead carry generator. Desgewenst kan deze structuur in multi-level vorm worden gerealiseerd. De toepassing van aparte schakelingen voor versneld informatietransport is zeker niet beperkt tot de optelling.

Bit- of woordgeoriënteerde aanpak

Wanneer de op te tellen getallen bestaan uit een verzameling bits met hetzelfde gewicht, dan ligt een bitgeoriënteerde aanpak voor de hand. Bij het optellen van woorden met een opklimmend gewicht ligt een woordgeoriënteerde aanpak voor de hand. Soms kan een combinatie van woordgeoriënteerde en bitgeoriënteerde aanpak tot aanzienlijke versnelingen leiden.

Interne representaties

Bij sommige interne representaties kan de verwerking van de carry uitgesteld worden, zoals bij de carry-save representatie. In dit geval duurt alleen de laatste optelslag langer. Kan in het algemeen een gedeelte van de bewerkingen uitgesteld worden en/of met andere worden gecombineerd?

Indien het interne data-transport uit de rekenkundige carry bestaat, dan ligt de codering van wel/geen carry met 1/0 van nature vast. Bij veel toepassingen is dit niet zo en moet er een codering worden toegewezen. Bij een comparator bijvoorbeeld moet de interne en externe codering van $>$, $=$ en $<$ nog worden vastgesteld. Dit kan met een 2-bit codering (in principe 24 mogelijkheden!), maar ook een 3-bit codering (336 mogelijkheden) komt in de praktijk voor. Argumenten om voor een bepaalde codering te kiezen liggen soms niet direct voor het oprapen. Het verdient dan aanbeveling dit aspect open te houden en dit deel van de ontwerp vrijheid later in te vullen.

Bewerkingstijd

Bewerkingstijd is een belangrijke parameter. Soms heeft de *bewerkingstijd* een vaste waarde/tijdsduur. Dan weer is deze variabel. Een variabele tijdsduur kan afhangen van de waarde van de operanden of van een ander kenmerk ervan, maar ook van het type bewerking of instructie. Bij de overweging of het algoritme gebaseerd wordt op een *worst-case benadering* of op een *gemiddelde benadering* speelt mee of het verschil de moeite waard is. Ook de in te zetten *hoeveelheid hardware* speelt mee. Bij serie-parallel vermenigvuldigers kan ten opzichte van parallelvermenigvuldigers een zekere hoeveelheid hardware bespaard worden. De prijs voor deze besparing is een langere bewerkingstijd. Ook voor andere algoritmes is het mogelijk tijd te ruilen tegen hardware. Vergelijk de eerder geïntroduceerde pipeline aanpak.

Een specifieke vorm van versnelling wordt bereikt door gebruik te maken van de *tabelmethode*, waarbij (tussen)resultaten van te voren worden uitgerekend. Bij de bewerking zelf dienen de operanden als adres en wordt het (tussen)resultaat teruggelezen. Dit van te voren uitrekenen kan eens en voor altijd, via een grote tabel met antwoorden. Daaraan dankt de methode zijn naam. Als alternatief is het mogelijk parallele berekeningen slechts vlak van te voren op te starten en dan een van de antwoorden te selecteren. Een voorbeeld van deze laatste benadering is de opteller met carry-select, waarbij de inkomende carry een van de twee eerder berekende tussenresultaten selecteert.

De vorenstaande overwegingen moeten leiden tot de formulering van het te implementeren algoritme. Daarbij kijken we enerzijds terug naar het programma van eisen, maar anderzijds ook vooruit naar de technische mogelijkheden.

Schema optelling / Het logisch ontwerp van de schakeling

De vertaling van het algoritme naar een logisch ontwerp van de schakeling is grotendeels recht toe recht aan. Dit is zeker het geval bij 'low performance' hardware. Een silicon compiler is dan het aangewezen gereedschap en de afbeelding geschiedt langs geautoma-

tiseerde ontwerpstappen. Een groot voordeel is dat het resultaat (meestal) 'correct-by-design' is.

Wanneer de 'performance' van het resultaat in termen van snelheid en/of dissipatie en/of oppervlak te laag is moet een optimalisatieprocedure op het ontwerp worden toegepast [Nagendra, 1996]. Dit kan per module en/of over de modules heen. Voorbeelden hiervan hebben we reeds besproken bij de realisatie van optellers. Een belangrijke (tijd)winst wordt bereikt als we van een ripple carry of andere interne vormen van seriële data-overdracht kunnen overstappen op parallelle of serie-parallelle data-overdracht. In deze gevallen moet er meer voorwerk op het algoritmeniveau gedaan worden en is een *interactieve* silicon compiler nodig. De structuur in het algoritme en/of in het logisch ontwerp moet dan in de een of andere vorm op het silicium behouden blijven. Naast logische functionaliteit speelt dan ook de structuur van het ontwerp een rol tot en met het silicium.

Samenvatting

Het vorenstaande overzicht bevat een groot aantal overwegingen en details. We dienen hierbij te bedenken dat het ontwerp in de praktijk in enkele iteraties zijn definitieve gestalte krijgt. Het 'al doende leert men' is vaak van toepassing. Moge het een pleidooi zijn om het ontwerpproces toch vooral top-down te structureren, waarbij details worden uitgewerkt en ingevuld zodra deze nodig zijn in het ontwerpproces en niet eerder.

Een plan van aanpak, met de aan het momentele ontwerpniveau gekoppelde lijst met variatiemogelijkheden, draagt ertoe bij dat het ontwerp gestructureerd wordt aangepakt en dat de juiste keuzes op het juiste tijdstip gedaan worden. Tot slot nog enkele, ongeordende, conclusies.

- De module-omvang in de schakeling hoeft niet noodzakelijk dezelfde te zijn als in het algoritme.
- De duur van een bewerking kan op verschillende manieren worden teruggebracht. Een snellere schakeling betekent niet automatisch een snellere technologie.
- Bij het opstellen van een specificatie is het zinvol te onderzoeken of de operanden in een bewerking specifieke kenmerken bezitten.
- Interne informatie-overdracht kan via een ander structuur worden opgebouwd dan die door het algoritme wordt gesuggereerd. Logisch heeft men meer vrijheid in de structuur.
- Het aantal parallel te bewerken operanden is een belangrijke parameter. Bij meer dan twee operanden is het vaak mogelijk een deel van de bewerking(en) uit te stellen en in de tijd te combineren met of parallel te laten lopen aan andere bewerkingen.
- De externe representatie van de te bewerken data hoeft niet noodzakelijk overeen te komen met de interne representatie. Een aangepaste interne representatie kan voordelen bieden.

Op deze wijze geformuleerd laten de conclusies meer ruimte over voor toepassing bij andere problemen dan wanneer zij geformuleerd worden aan de hand van de optelling. In de volgende paragraaf geven we een ontwerpopdracht.

7.6 Ontwerpopdracht

We zijn toe aan het toepassen van het voorafgaande. Als onderwerp kiezen we het ontwerp van een comparator. Er zijn verschillende types van comparatoren in gebruik, 'identity' en 'magnitude' comparatoren. Een *identity comparator* gaat na of een aangeboden bit string bit-voor-bit gelijk is aan een eerder ingestelde bit string. Deze bewerking komt onder meer voor bij de adressering van modules welke op een bus zijn aangesloten. Een *magnitude comparator* vergelijkt twee numerieke getallen, welke door binaire bit strings zijn voorgesteld. Het resultaat is nu een uitspraak over welk getal het grootst is, of dat beide getallen aan elkaar gelijk zijn. In een getallenverzameling berusten de begrippen 'groter', 'gelijk' en 'kleiner' op de in die getallenverzameling gedefinieerde *orderrelatie*. De eigenschappen van orderrelaties zijn uit de wiskunde bekend. Welke zijn de eigenschappen van de orderrelatie in de verzameling integers? Blijven deze eigenschappen behouden als we de integers op een bepaalde manier representeren, bijvoorbeeld bij in de BCD code gerepresenteerde niet-negatieve integers? Hoe zit dit bij het two's complement? De beantwoording van deze vragen dient te geschieden voor of tijdens het opstellen van een algoritme.

Is een algoritme opgesteld en zijn de bijbehorende randvoorwaarden bekend, dan zijn we toe aan de implementatie ervan in een schakeling. Allereerst komen dan de *structurele aspecten* aan bod. Zit er een zekere regelmaat in de structuur van een comparator of kan deze er in aangebracht worden? Zijn er parallele of gedeeltelijk parallel werkende structuren te bedenken en zo ja, onder welke voorwaarden. Vergt de toepassing een snelle comparator of zijn er geen specifieke snelheidseisen?

Speciale aandacht verdient de codering van het interne datatransport in een comparator. Voor de codering van de begrippen $>$, $=$ en $<$ zijn tenminste twee signalen nodig. De beschikbare vier codecombinaties kunnen dan op 24 verschillende manieren toegewezen worden. Stellen we een beslissing hierover uit tot op het laagste ontwerpniveau (Welke codering geeft de eenvoudigste schakeling?) of zijn er argumenten die in een eerder stadium tot een beslissing over de toewijzing van een codering leiden? Dat de toewijzing van de codering geen triviaal probleem is geeft tabel 7.1 aan. Hierin staat de codering van de tussenresultaten voor verschillende types comparatoren.

type comparator	SN7485	SN74LS685	SN74AS885
signalen	$> = <$	$\leq \neq$	$\geq \leq$
coding $>$	1 0 0	0 1	1 0
coding $=$	0 1 0	1 0	1 1
coding $<$	0 0 1	1 1	0 1

Tabel 7.1. Coderingen voor intern datatransport bij comparatoren.

Bij comparatoren kan tevens de *richting van het interne datatransport* worden gekozen. Dit in tegenstelling tot bij de optelling, waarvoor de carry altijd van laagwaardig naar hoogwaardig loopt. Zijn er argumenten te bedenken waarom en wanneer een van de twee richtingen de voorkeur heeft?

Gezien het vorenstaande is het ontwerp van een comparator een interessant probleem. Denk voor een toepassing aan een componentenbibliotheek, waarin het comparatorontwerp al dan niet parametrizeerbaar (aantal bits van de getallen instelbaar) is opgeslagen en van waaruit het ontwerp opgeroepen kan worden.

We komen nu tot de ontwerpopdracht, die in fasen uitgevoerd moet worden. De eerste fase bestaat uit het genereren en ordenen van een lijst met trefwoorden, die aangeven welke aspecten gedurende het ontwerptraject aan de orde *zouden kunnen komen*. Maak deze lijst zo compleet mogelijk. Bij grotere groepen cursisten is het uiterst leerzaam gebleken elkaars lijsten te vergelijken en een gecombineerde lijst op te stellen.

Specificeer vervolgens het te doorlopen ontwerptraject middels een schema van de achtereenvolgens te verrichten activiteiten. Betrek hierin ook de eventuele toepassing. (Een snelle comparator heeft andere aandachtspunten dan een comparator in een niet-tijdkritische omgeving.)

Werk de verschillende activiteiten nader uit. Geef eventuele alternatieven aan waar deze zinvol kunnen zijn. Doe dit tot en met het logisch ontwerpniveau/poortniveau. Het resultaat is een poortschema van de comparator, toegelicht met een motivatie van de verschillende ontwerpbeslissingen.

Het uitwerken van deze opdracht in kleine groepjes cursisten is zinvol. Dit te meer daar er verschillende programma's van eisen kunnen worden opgesteld ten aanzien van snelheid, modulariteit of parametrizeerbaarheid en/of een minimale oplossing in termen van aantallen poorten of een ander minimalisatiecriterium. Het vergelijken en bespreken van de verschillende oplossingen kan boeiende discussies opleveren!

Literatuur

1. A. Avizienis, *Signed-Digit Number Representations for Fast Parallel Arithmetic*, IEEE Trans. on Electronic Computers, Vol. EC-10, 1961, pp. 389 - 400.
2. O. J. Bedrij, *Carry-Select Adder*, IEEE Trans. on Electronic Computers, Vol. EC-11, 1962, pp. 340 - 346.
3. B. E. Briley, *Some New Results on Average Worst-Case Carry*, IEEE Trans. on Computers, Vol. C-22, 1973, pp. 459 - 463.
4. A. W. Burks, H. H. Goldstine and J. von Neuman, *Preliminary Discussions of the Logic Design of an Electronic Computing Instrument*, Inst. for Advanced Study, Princeton, 1946.
5. R.W. Doran, *Variants of an Improved Carry Look-Ahead Adder*, IEEE Trans. on Computers, Vol. C-37, 1988, pp. 1110 - 1113.
6. B. Chilchrist, J. H. Pomerence and S. Y. Wong, *Fast Carry Logic for Digital Computers*, IEEE Trans. on Electronic Computers, Vol. EC-4, 1955, pp. 133 - 136.
7. H. C. Lai and S. Muroga, *Logic Networks of Carry-Save Adders*, IEEE Trans. on Computers, Vol. C-31, 1982, pp. 870 - 882.
8. O. L. MacSorley, *High-Speed Arithmetic in Binary Computers*, Proc. IRE, Vol. 49, 1961, pp. 67 - 91.
9. C. Nagendra, M.J. Irwin, and R.M. Owens, *Area-Time-Power Tradeoffs in Parallel Adders*, IEEE Trans. on Circuit and Systems, Vol. 43, 1996, pp. 689 - 702.

10. A. P. Thijssen, H. A. Vink and C. H. Eversdijk, *Digitale Techniek, Deel 2*, Delft University Press, Delft, 1995.

Opgaven

- 7.1. Ontwerp een code-omzetter van de binaire code $\rightarrow$ gespiegeld binaire (Gray) code.

BIN code GRAY code

x	y	z	c	b	a
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

Voor de realisatie van deze code-omzetter staan poorten ter beschikking van het type AND, OR, NAND, NOR, NOT en EXOR. Favoriete poorten bij de realisatie van deze code-omzetter, gelet op de structuur in het verband tussen beide codes, zijn

- AND/OR poorten.
- AND/OR/NOT poorten.
- NAND en/of NOR poorten.
- EXOR poorten.

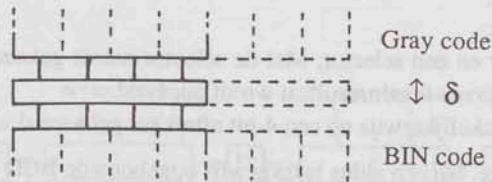
- 7.2. Maak een 16-bit BIN $\rightarrow$ Gray (gespiegeld binaire) code-omzetter. Ter beschikking staan AND, OR, NOT, NAND, NOR, en EXOR poorten. De belangrijkste eis, na uiteraard de correctheid van de omzetting, is een minimale propagatietijd door de code-omzetter en wel uitgedrukt in het aantal te doorlopen poortniveaus. De minimale propagatietijd door de code-omzetter bij de gegeven set poorten is de propagatietijd van

- één poortniveau.
- twee poortniveaus.
- 15 poortniveaus.
- 16 poortniveaus.

- 7.3. Ontwerp een k-bit gespiegeld binaire (Gray code) $\rightarrow$ k-bit binaire code-omzetter. Baseer, indien mogelijk, de code-omzetter op de structuur in het verband tussen beide codes en druk het verband uit in een hierbij passende logische operator of set van operatoren. Neem aan dat met deze logische operatoren overeenkomende poorten beschikbaar zijn. Indien de hoeveelheid poorten/hardware niet zo belangrijk is maar wel de propagatietijd door de code-omzetter (wens: minimale propagatietijd uitgedrukt in poortequivalenten), wat is dan het minimum aantal poortniveaus van de gehele code-omzetter?

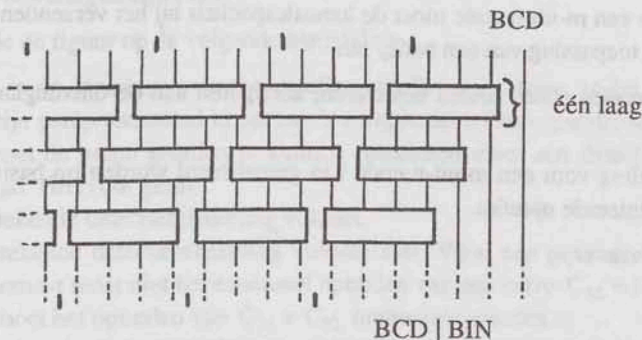
- Eén poortniveau.
- Twee poortniveaus.
- Het aantal poortniveaus is gelijk aan het kleinste gehele getal $\geq 2^{\log k}$.
- Het aantal poortniveaus is gelijk aan $k - 1$ of k .

7.4. Ontwerp een k -bit gespiegeld binaire (Gray code) $\rightarrow$ k -bit binaire code-omzetter. Bij het ontwerp van deze code-omzetter is het enerzijds van belang dat de code-omzetter gemakkelijk uitbreidbaar is en anderzijds dat in de lay-out, tussen de module die de gespiegeld binaire code aanlevert (ook uitbreidbaar) en de module die met de binaire code ingesteld wordt, de ruimte in de layout δ voor de code-omzetter in afstand niet of zo weinig mogelijk toeneemt. De code-omzetting zelf is niet tijd-kritisch in het lopende proces van de gehele schakeling.



- Voor de betreffende toepassing is een regelmatig uitbreidbare code-omzetter met identieke cellen mogelijk. De propagatietijd door de module is constant en onafhankelijk van k .
- Voor de betreffende toepassing is een regelmatig uitbreidbare code-omzetter met identieke cellen mogelijk. De propagatietijd door de module is lineair afhankelijk van k .
- Voor de betreffende toepassing is een regelmatig uitbreidbare code-omzetter met identieke cellen mogelijk. De propagatietijd door de module is niet constant en niet-lineair afhankelijk van k .
- Voor de betreffende toepassing is een regelmatig uitbreidbare code-omzetter met identieke cellen niet mogelijk.

7.5.



De omzetting van decimaal naar binair kan, rekenkundig, gebaseerd worden op herhaalde deling door 2. De achtereenvolgens optredende resten zijn dan de bits van de binaire representatie. Ga dit na. Decimale getallen worden wel voorgesteld door de BCD code. De BCD code heeft van nature een 4n-bit structuur, bij n decimale cijfers. Binnen elke groep van vier bits verhouden de gewichten van de bits zich als $8 : 4 : 2 : 1$. Elke groep van vier bits heeft daarnaast een gewicht, dat gelijk is aan een macht van het grondtal 10_{DEC} .

Binair komt delen door 2 neer op het één plaats naar rechts schuiven van de bits. Het laagstwaardige bit van de oorspronkelijke representatie is de daarbij optredende rest. In de BCD code verloopt delen door 2 niet zo gemakkelijk omdat, als een bit over de grens

tussen groepen van vier bits schuift, het gewicht niet gehalveerd wordt ($10^i \rightarrow 5 \cdot 10^{i-1}$) maar van $10^i \rightarrow 8 \cdot 10^{i-1}$ gaat. Er is dus een correctie nodig als een bit over de grens van vier bits schuift.

Ontwerp een omzetter van de 16-bit BCD code naar de binaire code. Houden we daarbij rekening met de hiervoor verstrekte informatie, dan ligt een laagsgewijze structuur van de omzetter voor de hand, waarbij in elke laag één keer door 2 gedeeld wordt. Elke 4-bit cel bestaat *minimaal* uit

- een 3-bit full adder.
- een 4-bit full adder.
- een 4-bit full adder en een selector. Met de selector wordt gekozen of er 0000 dan wel 0101 bij het verschoven tussenresultaat wordt opgeteld.
- De cel moet noodzakelijkerwijs op een 4-bit aftrekker gebaseerd worden

7.6. Als vorige opgave. Bij een aldus laagsgewijs opgebouwde BCD $\rightarrow$ binair code-omzetter zijn er in een 16-bit uitvoering minimaal

- 13 lagen nodig.
- 14 lagen nodig.
- 15 lagen nodig.
- 16 lagen nodig.

7.7. Om data te beschermen tegen fouten worden verschillende technieken toegepast. Een ervan is het toevoegen van een parity bit aan elk datawoord. Een andere techniek is het afbeelden van elk codewoord op een m-uit-n code. Over deze twee alternatieven worden enkele uitspraken gedaan.

- Het aantal fouten dat gedetecteerd kan worden is voor beide systemen gelijk.
- Bij toepassing van een m-uit-n code moet de kanaalcapaciteit bij het verzenden van data groter zijn dan bij toepassing van een parity bit.
- Beide systemen kunnen alleen fouten detecteren; corrigeren aan de ontvangkant is niet mogelijk.
- De detectieschakeling voor een m-uit-n code kan gerealiseerd worden op basis van een n-bit woordgeoriënteerde opteller.

Van deze uitspraken zijn er

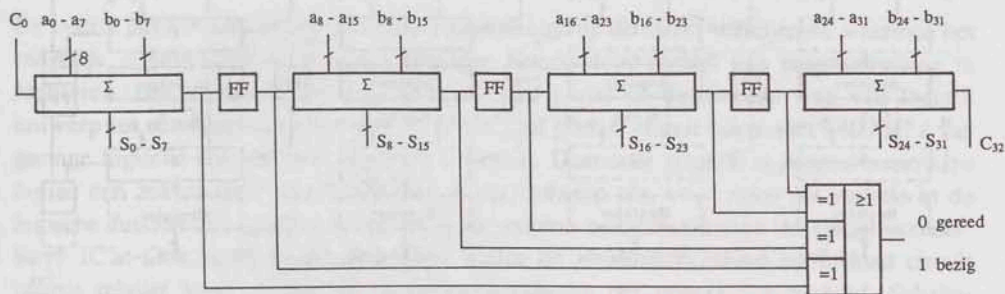
- één juist.
- twee juist.
- drie juist.
- vier juist.

7.8. Het toevoegen van een parity bit aan 7-bit datawoorden (ASCII) maakt het mogelijk een aantal fouten te detecteren. Daarbij kan men kiezen uit even of oneven parity. De voorkeur gaat uit naar

- even parity, omdat bij even parity de parity checker eenvoudiger is.
- oneven parity, omdat de parity checker eenvoudiger is.
- even parity omdat bij even parity kabelbreuk ook gedetecteerd kan worden.
- oneven parity, omdat bij oneven parity kabelbreuk ook gedetecteerd kan worden.

7.9. Gegeven is het schema van een carry completion opteller. Met deze structuur van de opteller is het mogelijk dat de opteller zelf aangeeft dat een optelling gereed is. Over deze opteller worden twee beweringen gedaan.

- Doordat de opteller zelf aangeeft wanneer de optelling gereed is kan de maximale tijd voor het optellen van twee binair gecodeerde getallen gereduceerd worden.
- Voor deze opteller geldt dat het maximale rendement (reductie van de opteltijd) bereikt wordt door na elke full adder een flip-flop te plaatsen. (Voor de werking van flip-flops zie deel 2.)

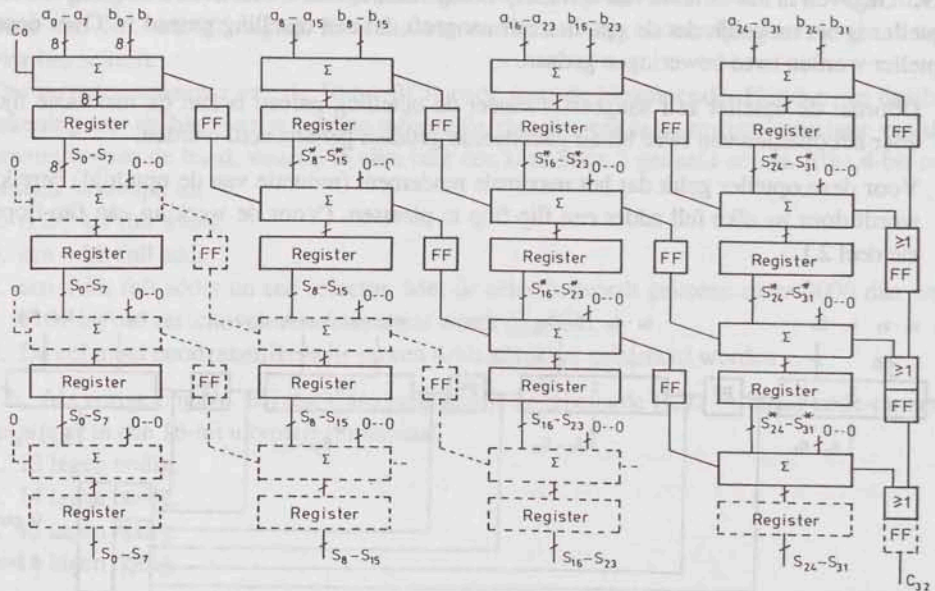


- Van deze beweringen is de eerste fout en is de tweede juist.
- Van deze beweringen is de eerste juist en is de tweede fout.
- Beide beweringen zijn juist.
- Beide beweringen zijn fout.

7.10. (Zie de figuur op de volgende bladzijde.)

Gegeven is de structuur van een pipeline opteller voor twee 16-bit gehele getallen. De getallen zijn gerepresenteerd in het two's complement. Om capaciteitsoverschrijding bij het optellen van de paren getallen te kunnen detecteren moet een detectieschakeling worden toegevoegd. Hiervoor geldt:

- De getekende detectieschakeling voldoet.
- De getekende detectieschakeling voldoet niet. Voor een pipeline opteller in het two's complement moet niet het eventueel optreden van een carry $C_{32} = 1$ onthouden worden, maar moet het optreden van $C_{31} \neq C_{32}$ onthouden worden.
- De getekende detectieschakeling voldoet niet. Voor een pipeline opteller in het two's complement moet niet het eventueel optreden van een carry $C_{32} = 1$ onthouden worden, maar moet het optreden van $C_{31} \neq C_{32}$ onthouden worden. Wanneer in een volgende laag weer $C_{31} \neq C_{32}$ optreedt, dan moet de detectieschakeling gereset worden.
- Geen van de drie genoemde oplossingen voor de detectie van capaciteitsoverschrijding in een op het two's complement gebaseerde pipeline opteller voldoet.



Stelling

De tijd, die wordt geïnvesteerd in het herkennen van structuur in een probleemstelling, brengt meestal een zeer hoog rendement op.

8

Programmeerbare logica

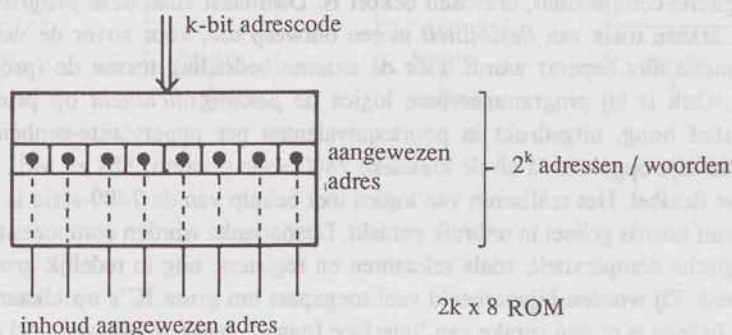
8.1 Read Only Memories

De laatste jaren is een uitgebreide serie componenten op de markt verschenen, waarmee het mogelijk is geworden de gewenste logische functie door middel van programmering te realiseren. Het voordeel van deze componenten is dat de tijdrovende weg van logisch ontwerp tot een speciaal ontworpen IC (ASIC), of printed circuit board met 7400 IC's van geringe logische complexiteit, drastisch bekort is. Daarnaast staat deze *programmeerbare logica* een zekere mate van *flexibiliteit* in een ontwerp toe, voor zover de variatie in de logische functie niet beperkt wordt door de externe bedrading tussen de (programmeerbare) IC's. Ook is bij programmeerbare logica de *pakkingsdichtheid* op printed circuit boards relatief hoog, uitgedrukt in poortequivalenten per oppervlakte-eenheid. Schakelingen, welke zijn opgebouwd uit de klassieke 7400-serie poorten, SSI en MSI logica, zijn duur en niet flexibel. Het realiseren van logica met behulp van de 7400-serie is daarom op printed circuit boards geheel in onbruik geraakt. Desondanks worden componenten met een geringe logische complexiteit, zoals selectoren en registers, nog in redelijk grote volumes geproduceerd. Zij worden bijvoorbeeld veel toegepast om grote IC's op elkaar te kunnen aansluiten. In feite is er dan sprake van 'interface fouten', maar de blik van veel ontwerpers gaat nu eenmaal niet verder dan de pennen van het op dat moment te ontwerpen IC. (Een niet onbelangrijk deel van de industriële ontwerp-activiteiten op logisch gebied wordt in beslag genomen om bepaalde IC's, waaronder geheugens, in te pakken en aldus geschikt te maken voor gebruik in een bepaalde (kloksynchrone) ontwerpomgeving. Zie deel 2.) [Agrawal, 1984; Bolton, 1985; Cavlan, 1975; Fleisher, 1975; Larson, 1980]

Een in programmeerbare logica ontworpen schakeling heeft als nadeel een wat *hogere dissipatie*. Bij veel toepassingen zijn de voordelen groter, vooral de korte tijd die benodigd is om de schakeling te produceren. We hebben recent dan ook een ombuiging naar de toepassing van programmeerbare logica gezien, waarbij vrij grote systemen in één chip kunnen worden geprogrammeerd. Daarnaast blijft men voor de zeer grote aantallen ASIC's ontwerpen, pasklare IC's voor specifieke toepassingen. Voor grote aantallen vallen de hogere ontwerpkosten weg en wordt de eindprijs in oppervlakte-eenheden silicium uitgedrukt. En die is bij programmeerbare logica nu eenmaal hoger. Economische afwegingen, en niet de technische, geven in vrijwel alle gevallen de doorslag. In dit hoofdstuk bespreken we enkele mogelijkheden om schakelingen in programmeerbare logica te realiseren. Er bestaan veel programmeerbare IC's, met verschillende namen, die vrijwel hetzelfde doen. Merkbescherming is hiervan de oorzaak. We maken voor combinatoriek een hoofdindeling in ROM's, PLA's en PAL's. Sequentiële programmeerbare logica bespreken we in deel 2.

Read only memories

Read only memories (ROM's) zijn ontworpen om er *permanente geheugens* mee te realiseren, bedoeld voor opslag van systeemprogramma's en 'harde' data. In concept is een ROM een component die, eenmaal geprogrammeerd, na aanbidding van een adrescode de inhoud van het aangewezen adres *reproduceert* op de uitgangspennen. Met een adresgenerator is het mogelijk opeenvolgende adressen uit te lezen, in de door de generator bepaalde volgorde. Aldus kan een programma of kan data worden uitgelezen. Figuur 8.1 toont het concept. Weldra heeft men ingezien dat een ROM geheugen combinatorische logica kan vervangen. Immers, een combinatorische schakeling genereert na het aanbieden van elke ingangscombinatie een voor die ingangscombinatie specifieke uitgangscombinatie. Het doet er dan niet toe of deze uitgangscombinatie telkens opnieuw wordt berekend, zoals een poortschakeling dit doet, of dat deze combinatie wordt teruggelezen uit een ROM na er eerst in opgeslagen te zijn [Fletcher, 1980; Toshiba, 1985].



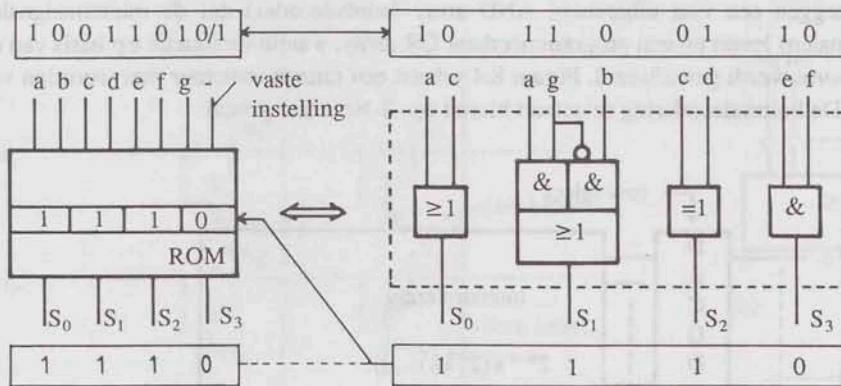
Figuur 8.1. Concept van een ROM.

De poortschakeling in figuur 8.2 heeft zeven ingangen en vier uitgangen. Bij elke ingangscombinatie abcdefg krijgt het uitgangsveld de waarde $S_3S_2S_1S_0$. Rekenen we nu voor alle ingangscombinaties het uitgangsveld van de poortschakeling uit en slaan we die velden op de betreffende adressen van een ROM op, dan reageert de ROM vervolgens (logisch) identiek. Soms zijn niet alle ingangen van de ROM nodig, omdat het aantal ingangsvariabelen van de te vervangen combinatorische schakeling kleiner is. De overbodige adresingangen stellen we dan op een vaste waarde in. Op deze wijze beschouwd zijn een ROM en een combinatorische schakeling in concept *logisch identiek*. De propagatietijd van een ROM is meestal wat groter, o.a. vanwege de complexe adresdecoder. Ook de andere (fysische) eigenschappen, zoals de dissipatie, zijn verschillend. Opdat een poortschakeling vervangen kan worden door een geprogrammeerde ROM moeten *alle relevante eigenschappen* geverifieerd worden en niet alleen de logische!

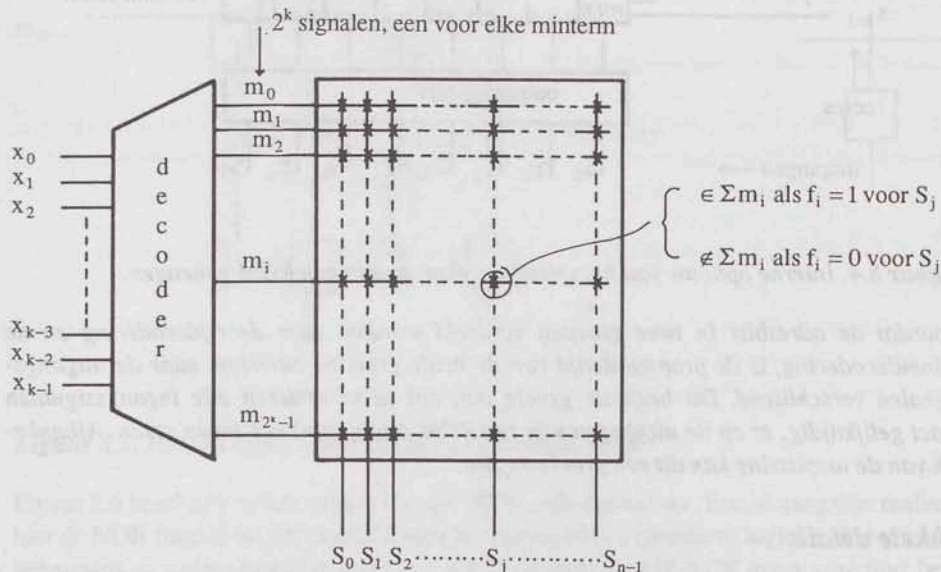
De interne structuur van een ROM

Figuur 8.3 schetst in concept de interne structuur van een ROM geheugen. Een *adresdecoder* zet de k ingangssignalen (adreslijnen) om in 2^k interne signalen. Elk intern signaal

activeert dan één adres in het geheugen. De inhoud van dit adres verschijnt vervolgens aan de uitgangen. Volgens deze 'logica beschouwingswijze' van een ROM mag elke interne adreslijn (uitgang van de decoder) gezien worden als een *mintermsignaal*. De betreffende uitgangsfunctie wordt dan gerealiseerd door de OR functie te nemen van alle m_i 's waarvoor $f_i = 1$ is (voor het begrip minterm: zie paragraaf 2.5). Op deze wijze kunnen *n willekeurige logische functies* in een ROM gerealiseerd worden, op elk van de *n* uitgangen één. In de praktijk heeft een ROM meestal acht uitgangsfuncties, gedefinieerd op ten hoogste *k* variabelen (*k* is het aantal adresbits van de ROM).

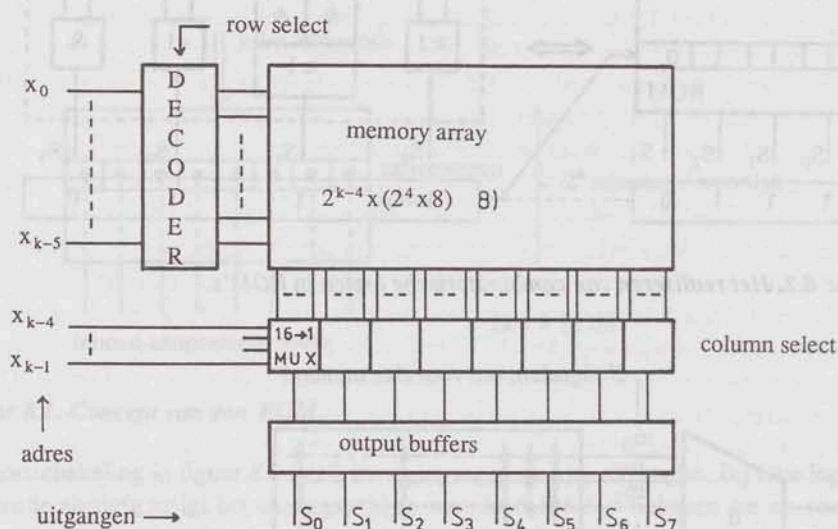


Figuur 8.2. Het realiseren van combinatorische logica in ROM's.



Figuur 8.3. De interne structuur van een ROM geheugen.

In een ROM is het *decoder gedeelte* vast en is het zogeheten *OR array* 'vrij' programmeerbaar. Om praktische redenen (layout op IC niveau) wijkt de interne structuur van grote ROM's af van de geschetste theoretische structuur in figuur 8.3. De adresdecoding geschiedt dan in twee stappen, de *rijdecoding* (row decoding) en de *kolomdecoding* (column decoding). Het geheugenarray is nu opgebouwd uit woorden met een bepaalde breedte, die een veelvoud is van het uitgangsveld. De rijdecoding wijst een woord in het geheugenarray aan. De kolomdecoding selecteert vervolgens het gewenste uitgangsveld. Voor de gebruiker blijft deze interne (logische) structuur gewoonlijk verborgen. Voor een inzicht in de *externe* logische werking kan het model volgens figuur 8.1 gehanteerd worden, dat wil zeggen een vast uitgevoerd AND array (adresdecoder) dat de mintermsignalen (adressignalen) levert en een programmeerbaar OR array, waarin de functie op basis van de *mintermvorm* wordt gerealiseerd. Figuur 8.4 schetst een interne structuur met woorden van 128 bits. De kolomdecoding selecteert hieruit een 8-bit uitgangsveld.



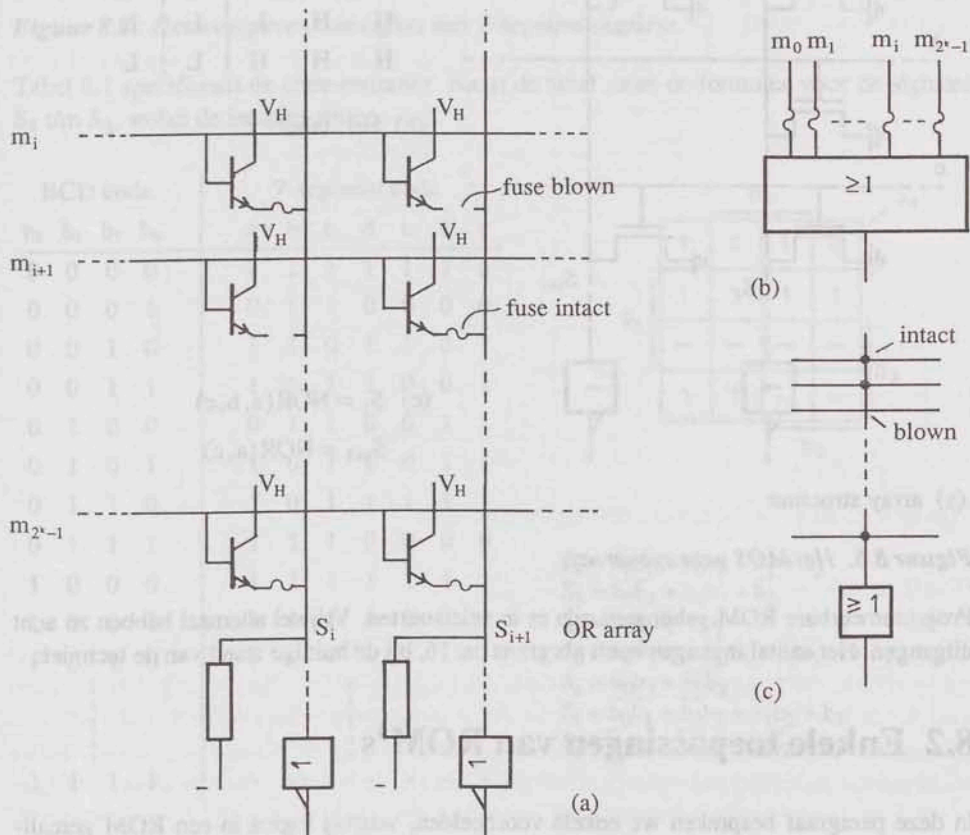
Figuur 8.4. Interne opbouw van het selectiemechanisme in een ROM geheugen.

Doordat de adresbits in twee groepen verdeeld worden, voor de rijdecoding en de kolomdecoding, is de propagatietijd van de beide groepen adresbits naar de uitgangssignalen verschillend. Dit heeft tot gevolg dat, ook al veranderen alle ingangssignalen exact gelijktijdig, er op de uitgangen van een ROM stevige spikes kunnen staan. Afhankelijk van de toepassing kan dit een probleem zijn.

Enkele details

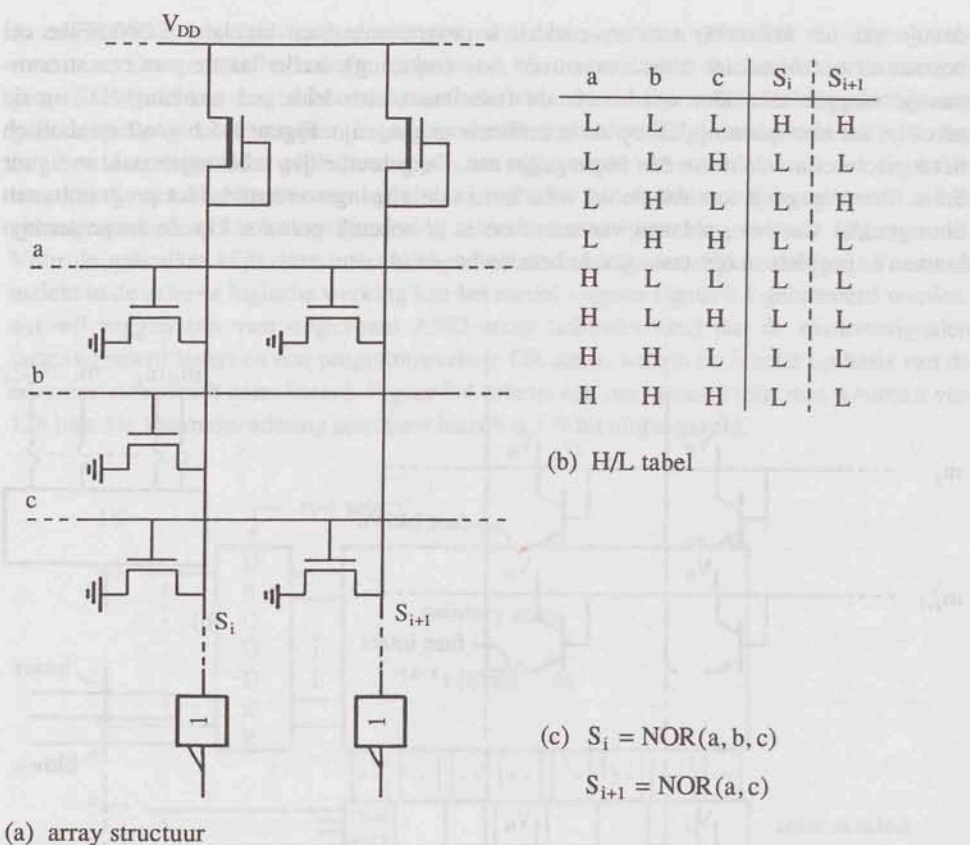
ROM's zijn er in verschillende uitvoeringen, variërend van *mask programmable* bij de fabricage tot '*electrically (re)programmable*' door de gebruiker. Figuur 8.5 geeft enkele

details van het OR array van een elektrisch programmeerbare bipolaire ROM. Elke cel bestaat uit een *bipolaire transistor* en een *fuse* (zekering), welke laatste met een stroomstootje weggeblazen kan worden. Is de fuse intact, dan leidt een spanning 'H' op de adreslijn tot een spanning 'H' op de betreffende uitgangslijn. Figuur 8.5.b geeft symbolisch het logische equivalent van één uitgangslijn aan. De gebruikelijke tekenwijze staat in figuur 8.5.c. Een stip geeft aan dat de cel meedoet in de uitgangssommatie. Het programmeren door middel van het opblazen van een fuse is in onbruik geraakt. Op de lange termijn kunnen er problemen ontstaan met de betrouwbaarheid.



Figuur 8.5. Het OR array van een bipolaire fuseable ROM.

Figuur 8.6 beschrijft enkele details van een MOS geheugenarray. Een uitgangslijn realiseert hier de NOR functie op alle doorgeschakelde ingangslijnen (positieve logica). Hoewel MOS geheugens en andere programmeerbare logica een interne NOR-NOR array structuur bezitten, spreekt men vaak van het AND array en het OR array. Immers, voor de gebruiker blijft het verborgen hoe het ROM geheugen intern is opgebouwd.



Figuur 8.6. Het MOS geheugenarray.

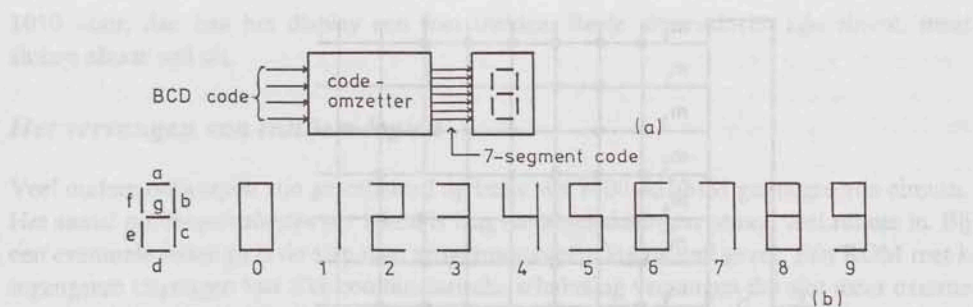
Programmeerbare ROM geheugens zijn er in vele soorten. Vrijwel allemaal hebben ze acht uitgangen. Het aantal ingangen heeft als grens ca. 16, bij de huidige stand van de techniek.

8.2 Enkele toepassingen van ROM's

In deze paragraaf bespreken we enkele voorbeelden, waarbij logica in een ROM gerealiseerd wordt. Het eerste voorbeeld is de realisatie van een *BCD-to-7-segment decoder*, zowel in een ROM als met poortequivalenten.

BCD-naar-7-segment decoder

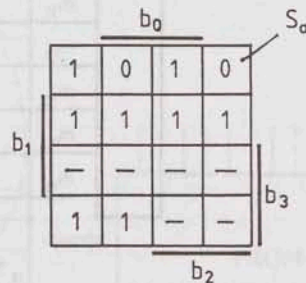
Decimale cijfers worden vaak weergegeven op 7-segment displays. Deze bestaan uit een zevental lampjes. Hoe de cijfers hiermee worden weergegeven toont figuur 8.7. Worden de cijfers aangeboden in de BCD code, dan is voor de aansturing van de lampjes een code-omzetter nodig van de *BCD code* naar de *7-segment code*.



Figuur 8.7. Het weergeven van cijfers met 7-segment displays.

Tabel 8.1 specificeert de code-omzetter. Naast de tabel staan de formules voor de signalen S_a t/m S_g , welke de lampjes sturen.

BCD code				7-segment code						
b_3	b_2	b_1	b_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	-	-	-	-	-	-	-
1	1	1	1	-	-	-	-	-	-	-



$$S_a = \bar{b}_0 \bar{b}_2 + b_0 b_2 + b_1 + b_3$$

$$S_b = \bar{b}_0 \bar{b}_1 + b_0 b_1 + \bar{b}_2$$

$$S_c = b_0 + \bar{b}_1 + b_2$$

$$S_d = b_0 \bar{b}_1 b_2 + \bar{b}_0 b_1 + \bar{b}_0 \bar{b}_2 + b_1 \bar{b}_2 + b_3$$

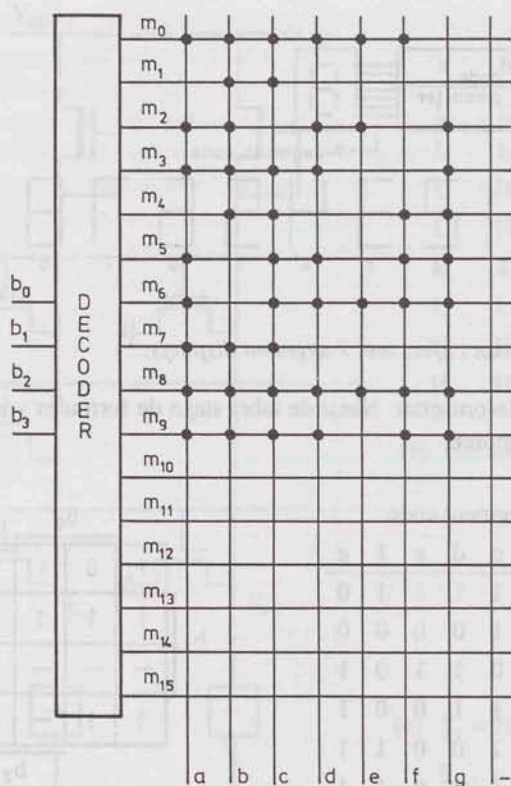
$$S_e = \bar{b}_0 b_1 + \bar{b}_0 \bar{b}_2$$

$$S_f = b_0 b_1 + \bar{b}_0 b_2 + \bar{b}_1 b_2 + b_3$$

$$S_g = b_0 b_2 + \bar{b}_1 b_2 + b_1 \bar{b}_2 + b_3$$

Tabel 8.1. Het verband tussen de BCD code en de 7-segment code.

De realisatie van deze formules in 7400-serie poorten kost veel IC's. De gehele code-omzetter past ook in één ROM. Figuur 8.8 geeft de programmering ervan aan in een 16x8 ROM structuur. (Binnen grote ASIC's heeft men dergelijke code-omzetter lange tijd ook afgebeeld op array-structuren. De voordelen bestonden o.a. uit het gemakkelijk kunnen genereren van de testpatronen. Met de huidige generatie ontwerpsoftware is een vertaling naar 'poorten' gemakkelijk te doen en overwegen de voordelen van een kleiner oppervlak en een kleinere dissipatie.)



Figuur 8.8. BCD-to-7-segment decoder in een ROM.

Don't cares

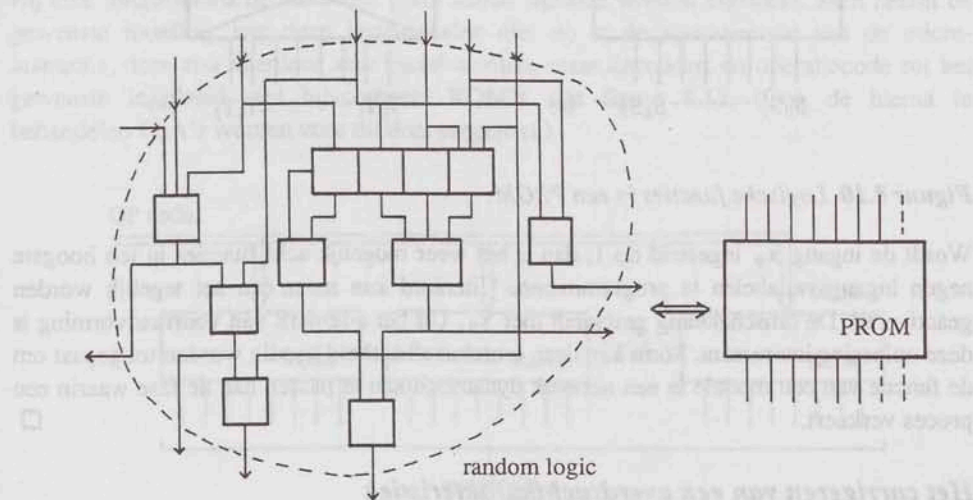
Bij realisaties in poorten is het van belang ontwerp-vrijheid te onderkennen en in *don't cares* om te zetten, teneinde deze te kunnen benutten bij het minimaliseren van de formules. In tabel 8.1 is van de don't cares achter de binaire ingangscombinaties 1010 t/m 1111 gebruik gemaakt om de functies S_a t/m S_g te vereenvoudigen. Bij programmering in een vaste ROM-structuur valt dit aspect weg. Het *aantal cellen* ligt dan vast met het aantal functies (uitgangen) en het aantal ingangsvariabelen (ingangen). Don't cares hebben hierop in principe geen invloed.

Soms is het echter mogelijk don't cares alsnog zinvol in te vullen, bijvoorbeeld voor het uitbreiden van het toepassingsgebied van het IC. Bij het voorbeeld van de BCD-to-7-segment omzetter is het mogelijk de specificatie aan te vullen tot een *hexadecimal-to-7-segment decoder*. In de hexadecimale code komen de bitcombinaties 1010 t/m 1111 wel voor. De aldus gerealiseerde code-omzetter is dan voor twee toepassingen geschikt. Passen we een standaard $2^{10} \times 8$ ROM toe, dan kan de achtste uitgang ook gebruikt worden om te signaleren dat de inhoud $\geq 10_{\text{DEC}}$ is. Dit signaal kan dan als testsignaal gebruikt worden. Ook een vorm van *foutendetectie* kan zinvol zijn. Komt een binaire ingangscombinatie $\geq$

1010 voor, dan kan het display een fout melden. Beide alternatieven zijn zinvol, maar sluiten elkaar wel uit.

Het vervangen van random logica

Veel oudere ontwerpen zijn gerealiseerd op basis van 7400 SSI/MSI geïntegreerde circuits. Het aantal *poortequivalenten* per board is laag en de schakelingen nemen veel ruimte in. Bij een eventuele redesign is de stap naar programmeerbare logica snel gezet. Een ROM met k ingangen/ n uitgangen kan elke combinatorische schakeling vervangen die niet meer externe ingangen/uitgangen heeft. Een en ander op voorwaarde dat de wat grotere propagatietijd van een ROM toelaatbaar is. Zie figuur 8.9. De procedure voor de vulling van de ROM is reeds in paragraaf 8.1 beschreven.



Figuur 8.9. Het vervangen van combinatorische logica door een PROM.

Programmeerbare ROM's zijn er in twee uitvoeringen, zogenaamde PROM's (éénmalig programmeerbaar) en RePROM's (herhaald programmeerbaar na wissen met bijvoorbeeld ultraviolet licht). Prototypes worden meestal met RePROM's gerealiseerd. In de definitieve uitvoering zijn PROM's voordeliger, sneller en mogelijk wat betrouwbaarder. (Zie ook hierna onder PLA's en PAL's).

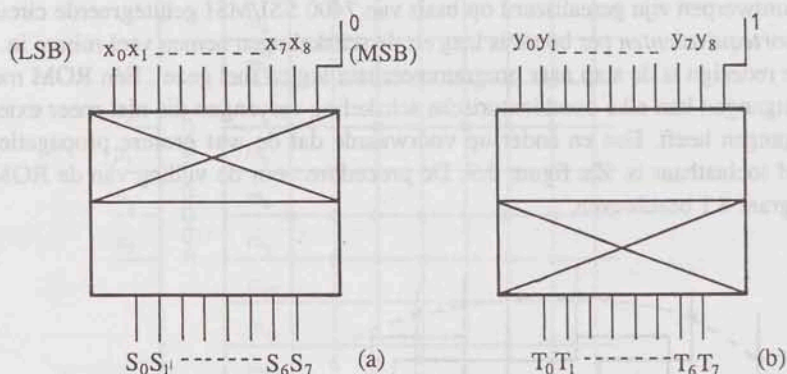
Opmerking

Bij het vervangen van combinatorische logica door PROM's moet men op de propagatietijden letten, welke van PROM's in het algemeen hoger zijn. Vergelijk ook hetgeen in hoofdstuk 5 over compatibiliteit is opgemerkt. □

De programmering van een set logische functies beslaat vaak niet de gehele inhoud van een ROM. Het volgende voorbeeld schetst enkele van de mogelijkheden die dit opent.

Voorbeeld

Voor de realisatie van acht functies, gedefinieerd op maximaal negen ingangsvariabelen, is een standaard PROM met 10 ingangen beschikbaar. De programmering beslaat de helft van de PROM in figuur 8.10.a. De ingang x_9 is ingesteld op 0.



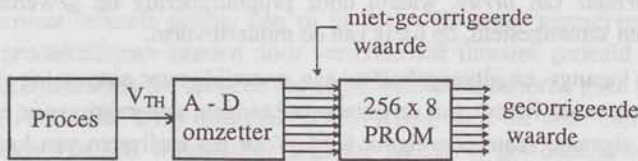
Figuur 8.10. Logische functies in een PROM.

Wordt de ingang x_9 ingesteld op 1, dan is het weer mogelijk acht functies in ten hoogste negen ingangsvariabelen te programmeren. Uiteraard kan maar één set tegelijk worden geactiveerd. De omschakeling geschiedt met x_9 . Uit het oogmerk van voorraadvorming is deze oplossing interessant. Soms kan deze omschakelbaarheid handig worden toegepast om de functie van een module in een netwerk dynamisch aan te passen aan de fase waarin een proces verkeert. □

Het corrigeren van een overdracht karakteristiek

Bij het meten van bepaalde grootheden heeft men vaak last van de niet-lineariteit van sensoren en andere omzetter. Een voorbeeld is het meten van een temperatuur met een thermokoppel. De door het thermokoppel afgegeven gelijkspanning, die niet recht evenredig is met de temperatuur, wordt met behulp van een analoog-digitaal omzetter (A-D omzetter) in een binair getal omgezet. Zie het schema in figuur 8.11. Voor een 8-bit binair gecodeerd getal kan men met een 256x8 PROM het meetresultaat gemakkelijk corrigeren. Voor elke combinatie van acht bits, zijnde een gemeten spanning, wordt de juiste waarde van de temperatuur in het geheugen opgeslagen.

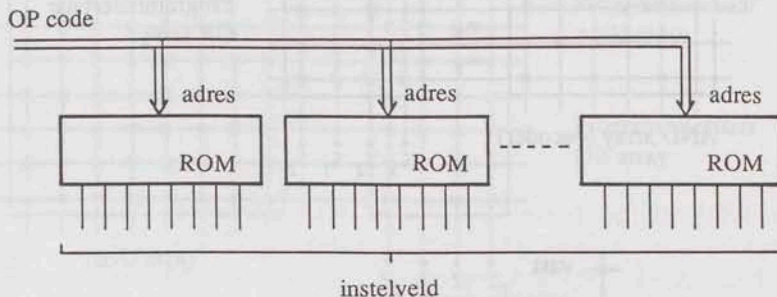
Voor het realiseren van *random logic* bieden ROM's en PROM's vele mogelijkheden. In het bovenstaande voorbeeld is het zeer onpraktisch om de correcties op de gemeten waarden in logische functies uit te drukken en deze vervolgens in losse poorten te realiseren. De schakeling zou te omvangrijk worden. Met PROM's echter is het zelfs mogelijk rekening te houden met de karakteristiek van elke sensor afzonderlijk.



Figuur 8.11. Toepassing van een PROM bij correctie van metingen.

Instructiedecodering

Computers hebben een instructieset. Meestal wordt het uitvoeren van een instructie gerealiseerd via het uitvoeren van een serie micro-instructies. Deze stellen de hardware in. Bij elke micro-instructie moet een groot aantal signalen worden ingesteld. Men neemt de gewenste instelling van deze instelsignalen niet op in de operatiecode van de micro-instructie, deze zou daardoor zeer breed worden, maar decodeert de operatiecode tot het gewenste instelveld met bijvoorbeeld ROM's. Zie figuur 8.12. (Ook de hierna te behandelen PLA's worden voor dit doel toegepast.)



Figuur 8.12. Instructiedecodering.

Bij niet-tijdkritische toepassingen zet men het instelveld vaak op *achtereenvolgende adressen* in één ROM. De operatiecode wordt dan aangeboden aan een *adresgenerator*, welke de bijbehorende adressen genereert. Deze worden achtereenvolgens aan het ROM-geheugen aangeboden. Registers (deel 2) aan de uitgang vangen dan de achtereenvolgende inhoud op en stellen het totale instelveld samen.

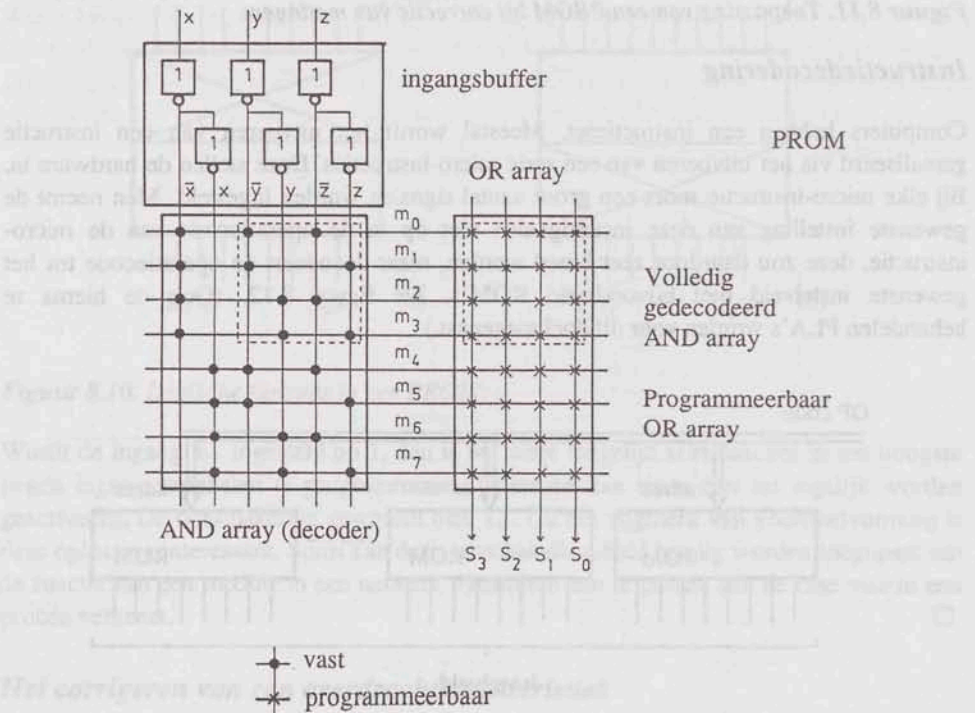
8.3. Programmeerbare logische arrays

Een PROM bestaat in concept uit twee arrays van cellen:

- een *vast uitgevoerd en volledig gedecodeerd AND array* waarin signalen, overeenkomend met de mintermen van de waarheidstabel, gegenereerd worden;

- een *programmeerbaar OR array*, waarin door programmering de gewenste logische functie kan worden samengesteld, op basis van de mintermvorm.

Daarnaast treft men ingangs- en uitgangsbuffers aan, enerzijds voor aanpassing van de fanin van externe ingangen en de fanout van de externe uitgangen, anderzijds voor een aanpassing aan de interne signaalniveaus. Zie figuur 8.13. Voor het realiseren van logica is deze structuur niet noodzakelijk, zoals we hierna zullen zien.



Figuur 8.13. Structuur van een PROM.

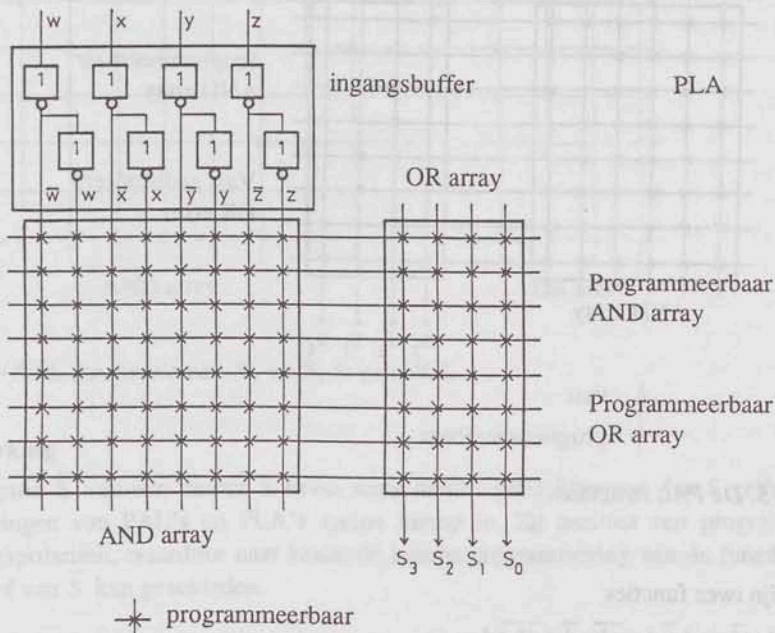
Tevens is in figuur 8.13 met stippellijnen aangegeven hoe groot het AND array en het OR array zouden zijn als de PROM één ingangsvariabele minder zou bezitten. Elke extra ingangsvariabele leidt tot meer dan een verdubbeling van het benodigde aantal cellen. Voor een functie in veel variabelen telt dit hard aan. PROM's zijn hiervoor dan ook minder geschikt.

Een voordeel van een realisatie van logische functies in PROM's is dat het *aantal produkttermen* van een functie er niet toe doet. De functie wordt op *mintermniveau* gerealiseerd. De waarheidstabel kan direct worden ingelezen, er is geen minimalisatie van formules nodig. Tegelijk is dit ook een van de beperkingen. Een OR poort met acht ingangen realiseert men bij voorkeur niet in een PROM.

Een verbetering van de 'silicon efficiency' ontstaat door ook het AND array programmeerbaar te maken. Om twee redenen biedt dit perspectief. Voor elke produktterm in een

logische formule behoeft slechts één rij in het AND array gereserveerd te worden. De gevormde produkttermen kunnen door verschillende functies gedeeld worden. Daarnaast behoeft voor mintermen die tot geen der uitgangsfuncties behoren geen ruimte gereserveerd te worden. Het AND array van een ROM/PROM doet dit wel. Het gemiddelde aantal produkttermen van een willekeurige logische functie is gewoonlijk veel kleiner dan het aantal mintermen ervan.

Een logische array met zowel een AND array als een OR array in programmeerbare uitvoering heet PLA (*programmable logic array*). Indien zij door de gebruiker programmeerbaar zijn heten zij FPLA (*field programmable logic array*) [Hemel, 1976].



Figuur 8.14. De PLA structuur.

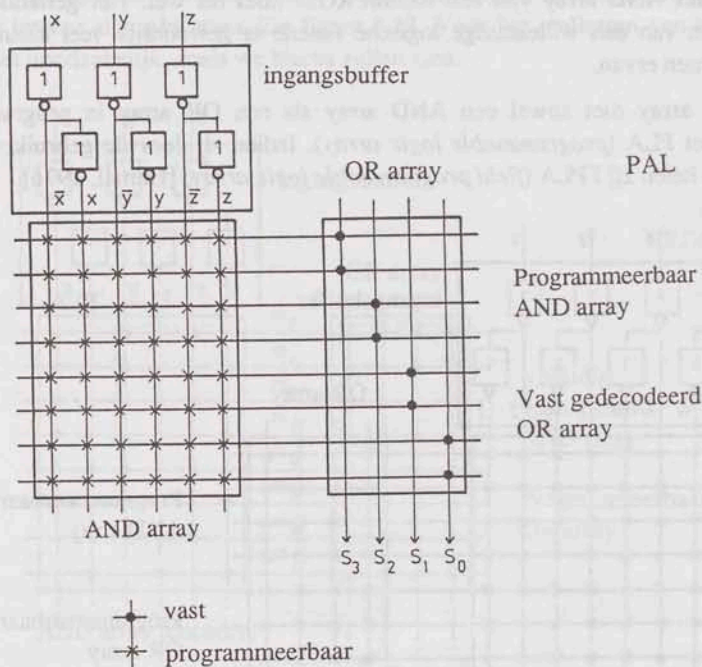
PLA's worden veel toegepast bij de realisatie van besturingslogica, ook binnen VLSI IC's. Een PLA van 16 ingangen en 64 produktrijen bevat in het AND array 2048 cellen. Een PROM met 16 ingangen zou er 1.048.576 bezitten.

Een *bepkering van PLA's* is dat niet iedere logische functie in k variabelen (k is het aantal ingangsvariabelen van de PLA) erin geprogrammeerd kan worden. Het aantal produkttermen is immers beperkt en als regel veel kleiner dan 2^k . ROM's, welke deze beperking niet kennen, en PLA's hebben daarom elk een (deels overlappend) eigen toepassingsgebied.

PAL's

Een beperktere vorm van programmeerbare logica treft men aan in PAL's (*programmable array logic*, merknaam van Monolithic Memories Inc.). In PAL's is het OR array vast

geprogrammeerd. Elke uitgangsfunctie kan slechts een beperkt aantal produkttermen bereiken. Daardoor is een PAL weliswaar goedkoper dan een PLA, maar kan ook minder [Birkner, 1978]. Zie figuur 8.15.



Figuur 8.15. De PAL structuur.

Voorbeeld

Gegeven zijn twee functies

$$S_0 = abcd + a\bar{b}cd + a\bar{b}\bar{c}d + a\bar{b}\bar{c}\bar{d}$$

en

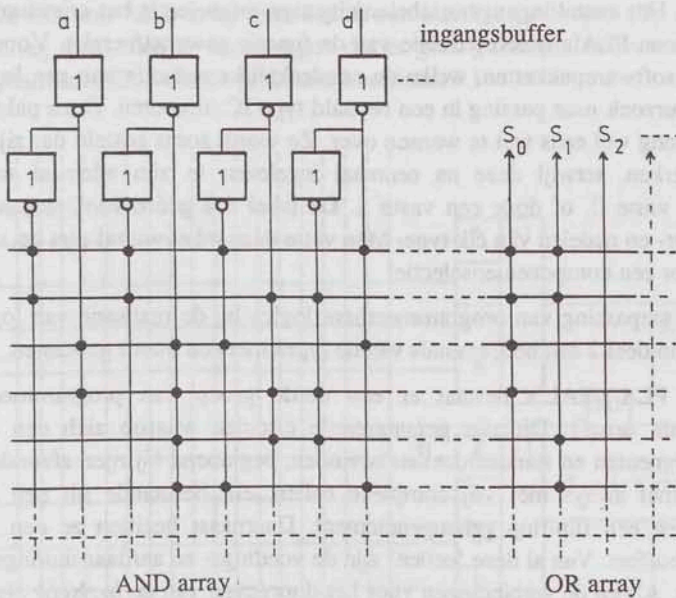
$$S_1 = acd + a\bar{c}d + bcd + \bar{b}cd$$

beide in hun eenvoudigste som-van-produktenvorm. Wanneer zij beide direct in deze vorm in een PLA geprogrammeerd worden, dan zijn hiervoor acht produktrijen nodig. De functie S_1 kan ook geschreven worden als

$$S_1 = abcd + a\bar{b}cd + a\bar{b}\bar{c}d + a\bar{b}\bar{c}\bar{d} + a\bar{b}\bar{c}d + a\bar{b}\bar{c}\bar{d}$$

Nu hebben S_0 en S_1 vier produkttermen gemeenschappelijk en twee niet. De realisatie kost nu zes produktrijen. Zie figuur 8.16.

Zouden we dezelfde twee functies in een PAL realiseren, met tenminste vier produktrijen voor elke uitgangsfunctie, dan vergt programmering altijd acht produktrijen. (Indien in een PAL elke uitgangsfunctie meer dan vier produkttermen kan bevatten, dan blijven bij realisatie van dit voorbeeld produkttermen ongebruikt.) $\square$



Figuur 8.16. Realisatie van S_0 en S_1 in een PLA.

Opmerking

De negatie $\bar{S}$ van een functie S bevat soms minder produkttermen dan S zelf. Sommige uitvoeringen van PAL's en PLA's spelen hierop in. Zij bezitten een programmeerbare uitgangspolariteit, waardoor naar keuze de interne programmering van de functie op basis van S of van $\bar{S}$ kan geschieden. □

	PROM	PLA	PAL	Poorten
Oppervlak PCB	+	++	+/-	--
Oppervlak IC	-	+	+/-	++
Dissipatie	-	-	-/+	++
Propagatietijd	-	-/+	-/+	++
Flexibiliteit	++	+	+/-	-
Ontwerpprocedure	++	+	+/-	-
Aantal variabelen	+/-	++	+/-	-
Aantal produkttermen	++	+	-	-

Tabel 8.2. Eigenschappen programmeerbare logica versus poorten.

PROM's/ROM's hebben als groot voordeel dat de logische functies niet vereenvoudigd behoeven te worden. Het aantal ingangsvariabelen/uitgangsvariabelen is het criterium voor passing. Voor PAL's en PLA's is een reductie van de functie gewenst/vereist. Voor deze bouwstenen bestaan softwarepakketten, welke de noodzakelijke reductie van een logische functie en/of het onderzoek naar passing in een bepaald type IC uitvoeren. Deze pakketten laten op dit moment nog wel eens wat te wensen over. Zo wordt soms gesteld dat zij don't cares kunnen verwerken, terwijl deze na eenmaal ingelezen te zijn allemaal worden vervangen door een vaste 0, of door een vaste 1. De tabel 8.2 geeft, heel globaal, een indicatie over de voor- en nadelen van elk type. Men vatte deze tabel vooral niet op als een *absolute richtlijn* voor een componentselectie!

Voorbeelden van de toepassing van programmeerbare logica bij de realisatie van logische schakelingen komen in deel 2 aan bod, evenals van de types met een intern geheugen.

Naast PROM's en PLA's/PAL's bestaat er een derde groep van programmeerbare componenten, de *gate arrays*. Dit zijn geïntegreerde circuits, waarop zich een aantal poorten, geheugenelementen en standaardcellen bevinden, beginnend bij rijen afzonderlijke transistoren tot en met arrays met vrij complexe cellen, elk bestaande uit een aantal poorten, selectoren en een flip-flop geheugenelement. Daarnaast bezitten ze een aantal ingangs- en uitgangsbuffers. Van al deze 'cellen' zijn de voedings- en aardaansluitingen van te voren aangebracht. Alleen de verbindingen voor het doorgeven van de *logische signalen* ontbreken nog. De logische bedrading wordt in de laatste processtappen aangebracht, op klantenspecificatie. Gate arrays maken het mogelijk een ontwerp snel te realiseren, tegen een gering verlies aan efficiency. Bij de fabricage van digitale apparatuur worden gate arrays veel toegepast. Zij bestaan ook in programmeerbare vorm [MMI, 1985; Signetics, 1977; Xilinx, 1997].

8.4. Optimalisatie in array logica

Soms past een logische functie niet in een beschikbaar array. De huidige generatie software tools verwijst dan vaak naar een groter type, met meer produktijken per sommatie. Veel PAL's en PLA's bevatten intern ook voorzieningen voor uitbreidingen.

Een *open-collector uitgang* staat de wired-AND toe (positieve logica). Daarmee kunnen functies van de vorm

$$S = S_0 \cdot S_1 \cdot \dots \cdot S_{n-1}$$

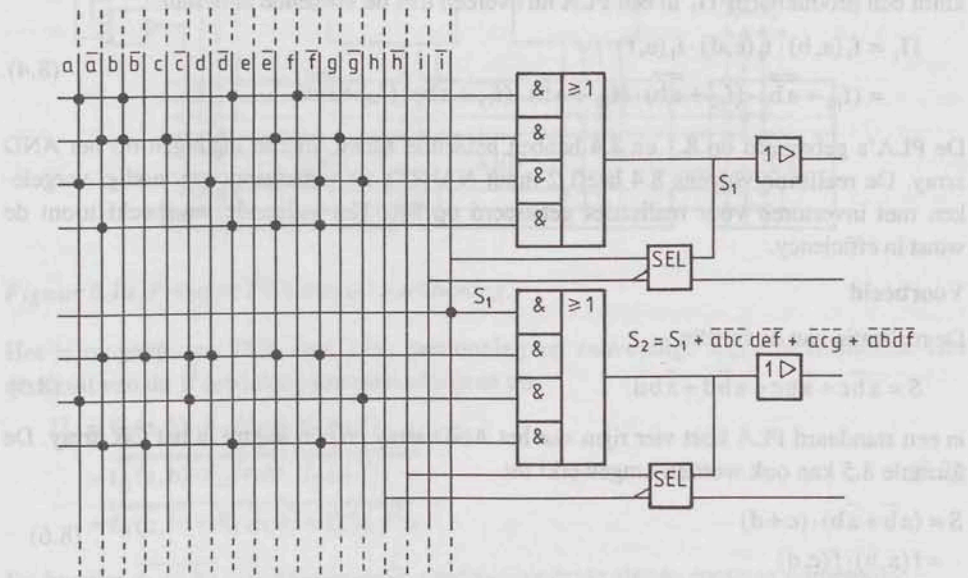
samengesteld worden, waarvan de factoren S_0 t/m S_{n-1} wel direct realiseerbaar zijn. Met een *externe OR poort* kunnen functies van de vorm

$$S = S_0 + S_1 + \dots + S_{n-1}$$

gerealiseerd worden. Beide constructies breiden de mogelijkheden van PAL's en PLA's aanzienlijk uit.

Bij veel PAL's en PLA's kan de OR functie ook intern gerealiseerd worden. De meeste types bezitten de mogelijkheid een uitgangssignaal intern terug te voeren naar het AND

array, ten koste van één extern ingangssignaal. Zie figuur 8.17 voor een voorbeeld. Welk signaal naar het AND array gaat beslist een instelbare selector. (Bij deze vorm van terugkoppeling moet men er voor zorgen dat er geen *geheugenlussen* ontstaan, zowel statisch als dynamisch. Zie voor dit probleem hoofdstuk 9.)



Figuur 8.17. PAL met interne terugvoer van uitgangssignalen.

Bit partitioning

In een PLA met de structuur van figuur 8.14 leidt iedere externe ingang op het AND array tot twee interne ingangen, één voor de variabele zelf en één voor zijn complement. In het AND array vormt iedere rij een produktterm, gebaseerd op de formule

$$\begin{aligned}\Pi_i &= f_i(a) \cdot f_i(b) \cdot f_i(c) \cdots \\ &= (f_{i0} + a) \cdot (f_{i1} + \bar{a}) \cdot (f_{i2} + b) \cdot (f_{i3} + \bar{b}) \cdot (f_{i4} + c) \cdots\end{aligned}\quad (8.1)$$

Een verbinding correspondeert met $f_{ij} = 0$. Bij geen verbinding is $f_{ij} = 1$ en behoort de betreffende variabele of diens complement niet tot de produktterm. Per tweetal variabelen komt één factor van de verzameling

$$\{a, \bar{a}, b, \bar{b}, ab, \bar{a}\bar{b}, a\bar{b}, \bar{a}b, 1, 0\}\quad (8.2)$$

in elke produktterm Π_i voor, of de betreffende variabelen komen niet voor in het produkt.

De verzameling (8.2) is slechts een deelverzameling van alle functies in twee variabelen. De efficiency van een PLA neemt sterk toe als voor elk tweetal variabelen een willekeurige functie ervan tot de produktterm Π_i kan behoren. Zo'n willekeurige functie van twee variabelen a en b kan bijvoorbeeld op de *maxtermvorm* gebaseerd worden (paragraaf 2.6):

$$\begin{aligned}
 f(a,b) &= (f_0 + a + b) \cdot (f_1 + a + \bar{b}) \cdot (f_2 + \bar{a} + b) \cdot (f_3 + \bar{a} + \bar{b}) \\
 &= (f_0 + \bar{a}\bar{b}) \cdot (f_1 + \bar{a}b) \cdot (f_2 + a\bar{b}) \cdot (f_3 + ab)
 \end{aligned}
 \quad (8.3)$$

De structuur van formule 8.3 vertoont veel overeenkomst met formule 8.1. Wanneer de aanwezigheid/afwezigheid van een verbinding weer wordt bepaald met $f_i = 0$ of $f_i = 1$, dan komt een produktterm Π_i in een PLA nu overeen met de volgende structuur:

$$\begin{aligned}
 \Pi_i &= f_i(a,b) \cdot f_i(c,d) \cdot f_i(e,f) \cdot \dots \\
 &= (f_{i0} + \bar{a}\bar{b}) \cdot (f_{i1} + \bar{a}b) \cdot (f_{i2} + a\bar{b}) \cdot (f_{i3} + ab) \cdot (f_{i4} + \bar{c}\bar{d}) \cdot \dots
 \end{aligned}
 \quad (8.4)$$

De PLA's gebaseerd op 8.1 en 8.4 hebben hetzelfde aantal interne ingangen op het AND array. De realisatie volgens 8.4 heeft 2-input NAND's als ingangspoorten nodig, vergeleken met invertoren voor realisaties gebaseerd op 8.1. Het volgende voorbeeld toont de winst in efficiency.

Voorbeeld

De realisatie van de functie

$$S = a\bar{b}c + \bar{a}bc + a\bar{b}d + \bar{a}bd \quad (8.5)$$

in een standaard PLA kost vier rijen van het AND array en één kolom in het OR array. De formule 8.5 kan ook worden omgewerkt tot

$$\begin{aligned}
 S &= (a\bar{b} + \bar{a}b) \cdot (c + d) \\
 &= f(a,b) \cdot f(c,d)
 \end{aligned}
 \quad (8.6)$$

In dit geval kost de realisatie van de functie S slechts één rij in het AND array. Vergeleken met 8.4 moeten de 'fuses' als volgt worden ingesteld:

$$f_{i0} = f_{i3} = 0 \quad \text{en} \quad f_{i1} = f_{i2} = 1$$

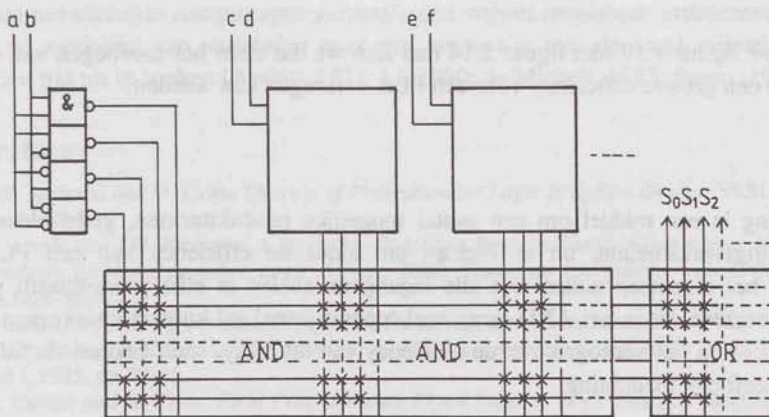
voor het eerste tweetal variabelen a en b en

$$f_{i4} = 0 \quad \text{en} \quad f_{i5} = f_{i6} = f_{i7} = 1$$

voor de variabelen c en d . □

Uiteraard kan men dit principe uitbreiden door de ingangsverzameling in groepen van drie of vier bits op te delen. Drie variabelen kosten standaard zes ingangslijnen van het AND array, terwijl het aantal maxtermen acht bedraagt. Bij vier en meer variabelen wordt deze verhouding nog ongunstiger. In de praktijk beperkt men zich daarom tot het verdelen van de ingangsverzameling in groepen van twee bits. Men noemt dit *bit partitioning*.

Figuur 8.18 beschrijft het principe van een PLA met bit partitioning. Boven het AND array bevinden zich verschillende OR arrays en wel één voor elk tweetal ingangsvariabelen. In zo'n OR array kunnen willekeurige functies in twee variabelen worden geprogrammeerd. Produkten van deze functies worden in het AND array bepaald. Deze produkten worden op hun beurt in het OR array aan de uitgang gesommeerd.



Figuur 8.18. Principe PLA met bit partitioning.

Het is mogelijk een PLA met 2-bit partitioning op eenvoudige wijze te realiseren. Het resultaat van de i^e produktrij kan men schrijven als

$$\begin{aligned} \Pi_i &= f_{i1}(a, b) \cdot f_{i2}(c, d) \cdot f_{i3}(e, f) \cdot \dots \\ &= \overline{f_{i1}(a, b)} \cdot \overline{f_{i2}(c, d)} \cdot \overline{f_{i3}(e, f)} \cdot \dots \\ &= \overline{f_{i1}(a, b) + f_{i2}(c, d) + f_{i3}(e, f) + \dots} \end{aligned} \quad (8.7)$$

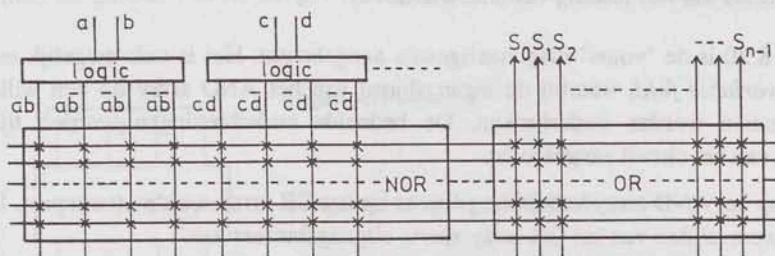
De functies $\overline{f_{i1}(a, b)}$ enz. kunnen weer worden uitgedrukt als een som van mintermen:

$$\overline{f_{i1}(a, b)} = \overline{a} \overline{b} f_{j0} + \overline{a} b f_{j1} + a \overline{b} f_{j2} + a b f_{j3} \quad (8.8)$$

waarin f_{j0} t/m f_{j3} de genegeerde functiewaarden zijn van de oorspronkelijke functie $f_{i1}(a, b)$. Hierdoor kan de formule voor het i^e produkt Π_i geschreven worden als

$$\Pi_i = \overline{a b f_{j0} + \dots + a b f_{j3} + c d f_{k0} + \dots + c d f_{k3} + e f f_{l0} + \dots} \quad (8.9)$$

Dit is de NOR functie. Het OR en AND array, welke samen de produkttermen Π_i vormen in een PLA met bit partitioning, kunnen vervangen worden door één NOR array. Figuur 8.19 beschrijft het resultaat.



Figuur 8.19. PLA met bit partitioning op basis van een NOR/OR array.

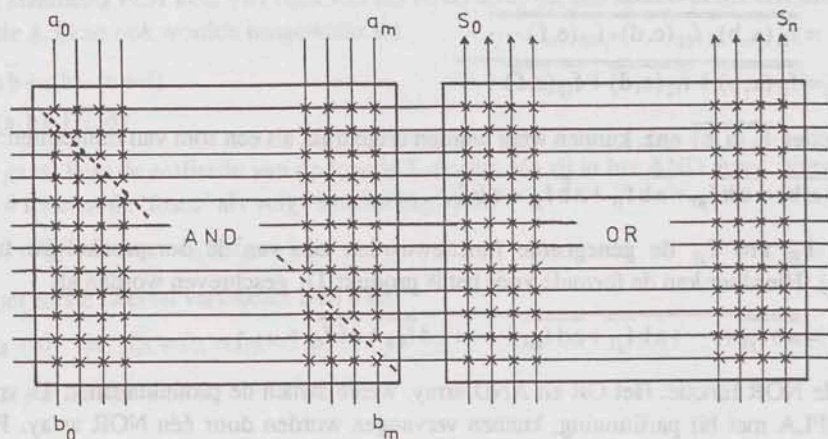
Opmerking

Vergelijken we figuur 8.19 met figuur 8.14 dan zien we dat door het toevoegen van enige extra poorten een grotere efficiency voor een PLA verkregen kan worden. $\square$

Folding

Bit partitioning is een middel om een aantal mogelijke produkttermen, gedefinieerd op dezelfde ingangsverzameling, uit te breiden om aldus de efficiency van een PLA te verhogen. In het algemeen zullen niet alle ingangsvariabelen in elke produktterm voorkomen. Dit betekent dat in het AND array veel 'open ruimte' zal kunnen voorkomen. Een techniek, gericht op het verhogen van de efficiency van dit array, staat bekend als *folding*. Figuur 8.20 geeft een toelichting.

Het produkt Π_0 op de eerste produktrij is uitgedrukt in de variabelen a_0 t/m a_m . Het produkt Π_1 is uitgedrukt in b_0 en a_1 t/m a_m . Het laatste produkt is uitgedrukt in de variabelen b_0 t/m b_m . De uitgangsfuncties S_0 t/m S_n kunnen alle aldus gevormde produktvormen bevatten. De techniek van *folding* maakt het mogelijk de uitgangsfuncties te definiëren op elkaar deels overlappende ingangsverzamelingen.



Figuur 8.20. PLA met folding van het AND array.

In figuur 8.20 is de 'vouw' diagonaalsgewijs aangebracht. Het is ook mogelijk te werken met een *variable fold*, waarbij de ingangslijnen van het AND array op een willekeurige plaats kunnen worden onderbroken. De bedoelde onderbrekingen worden tijdens de fabricage van het circuit aangebracht.

Behalve op het AND array kan folding tevens op het OR array worden toegepast. Dan treft men aan weerszijden van het OR array aparte uitgangsbuffers aan.

Naast bit partitioning maken de verschillende manieren van folding het mogelijk de realisatie van logica via PLA's verder te optimaliseren. Voor grotere PLA's zijn daartoe

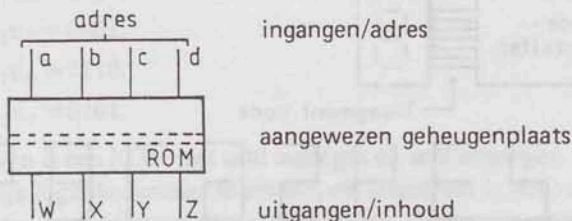
speciaal ontwikkelde computerprogramma's een vrijwel onmisbaar instrument. Problemen zoals de verdeling van variabelen over een vouw zijn met de hand vrijwel niet binnen redelijke tijd uit te zoeken [Arvalo, 1978; Li, 1980; de Micheli, 1983; Sasao, 1984].

Literatuur

1. O.P. Agrawal and D. Laws, *The role of Programmable Logic in System Design*, VLSI Design, 1984, pp. 44-50.
2. Z. Arvalo and J.G. Bredeson, *A Method to Simplify a Boolean Function into a near Minimal Sum of Products for Programmable Logic Arrays*, IEEE Trans. on Computers, Vol. C-27, Nov. 1978, pp.1028-1034.
3. J. Birkner, *PAL, Programmable Array Logic Handbook*, Monolithic Memories, 1978.
4. M.J.P. Bolton, A.C.G.I., D. Phil., *Designing with Programmable Logic*, IEEE Proc., Vol. 132, Pts E and I, 1985, pp. 73-85.
5. N. Cavlan and R. Cline, *Field Programmable PLA's Simplify Logic Designs*, Electronic Design 18, Sept. 1975, pp. 84-90..
6. H. Fleisher and L.I. Maissel, *An Introduction to Array Logic*, IBM J. Res. Develop., Vol. 19, March 1975, pp. 98-109.
7. W.I. Fletcher, *An Engineering Approach to Digital Design*, Prentice-Hall, Englewood Cliffs, N.J., 1980.
8. A. Hemel, *The PLA, a Different Kind of ROM*, Electronic Design 1, Jan. 1976, pp.78-84.
9. M. Wild, *Converting PALs to CMOS Gate Arrays*, VLSI Design, 1984, pp. 58-64.
10. T.L. Larson and C. Downey, *Field Programmable Logic Devices*, Electronic Engineering, Vol. 52, Jan. 1980, pp. 37-54.
11. H.F. Li, *Programmable Logic Array Optimization Techniques*, Int. J. Electronics, Vol. 49, 1980, pp. 287-299.
12. G. de Micheli and Sangiovanni-Vincentelli, *Multiple Constrained Folding of Programmable Logic Arrays: Theory and Applications*, IEEE Trans. on CAD, Vol. CAD-2, 1983, pp. 151-167.
13. MMI Inc. *Programmable Logic Handbook*, Monolithic Memories Inc., Santa Clara, 1985.
14. T. Sasao, *Input Variable Assignment and Output Phase Optimization of PLA's*, IEEE Trans. on Computers, Vol. C-33, 1984, pp. 879-894.
15. *Signetics Field Programmable Logic Array: An Application Manual*, Signetics Corp., Sunnyvale, Ca., 1977.
16. Toshiba Corporation, *MOS Memory Products*, Toshiba, Tokyo, 1985.
17. R.A. Wood, *A High Density Programmable Logic Array Chip*, IEEE Trans. on Computers, Vol. C-28, Sept. 1979, pp. 602-608.
18. Xilinx, *The Programmable Gate Array Design Handbook*, Xilinx, San José, Ca., 1987.

Opgaven hoofdstuk 8

8.1. ROM's (Read-Only Memories) zijn componenten waaruit, afhankelijk van het aangeboden adres, de inhoud van de aangewezen geheugenplaats naar de uitgangen gestuurd wordt.



De getekende ROM structuur heeft vier ingangen en vier uitgangen. Met deze ROM worden vier functies gerealiseerd:

$$W = a\bar{b} + \bar{a}\bar{b}\bar{c}d$$

$$X = a\bar{b}\bar{c}d$$

$$Y = a\bar{b}c + a\bar{c}d$$

$$Z = a + b$$

Voor het adres $abcd = 1011$ is de inhoud van de aangewezen geheugenplaats

- WXYZ = 1100.
- WXYZ = 0011.
- WXYZ = 1010.
- WXYZ = 1111.

8.2. Tekst en figuur als opgave 8.1. en wel voor de volgende vier functies

$$W = a\bar{b}c + \bar{c}d$$

$$X = a\bar{b}\bar{c}d$$

$$Y = a \oplus b \oplus b \oplus d \quad \oplus \leftrightarrow \text{EXOR}$$

$$Z = a + b + c + d$$

Voor het adres $abcd = 1010$ is de inhoud van de aangewezen geheugenplaats

- WXYZ = 1001.
- WXYZ = 1011.
- WXYZ = 1000.
- WXYZ = 0001.

8.3. Tekst en figuur als opgave 8.1. en wel voor de volgende vier functies

$$W = a\bar{b} + \bar{a}\bar{b}$$

$$X = a\bar{b}c\bar{d}$$

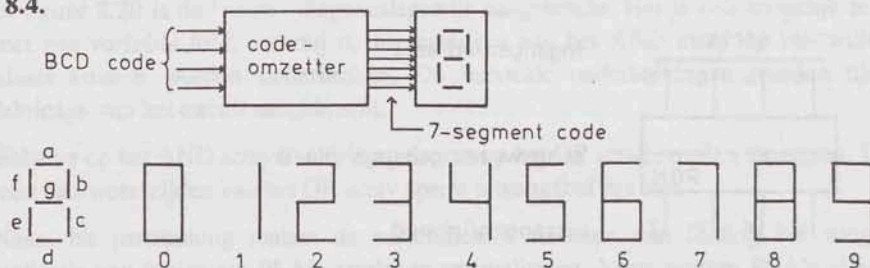
$$Y = a \oplus b \oplus b \oplus d$$

$$Z = a + b + c + d$$

Dan bevat het geheugen in totaal

- 16 enen.
- 24 enen.
- 32 enen.
- een ander aantal enen.

8.4.

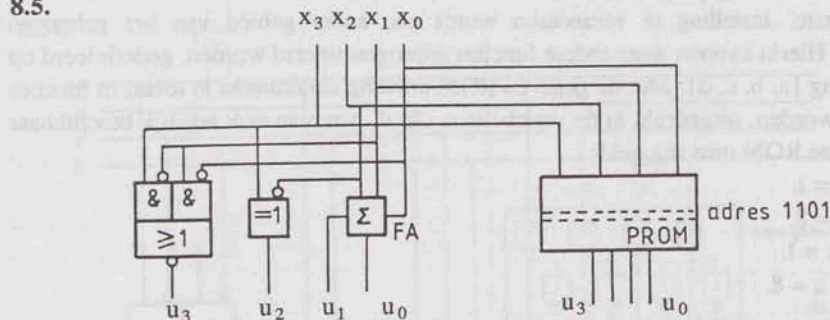


Cijfers worden vaak weergegeven met 7-segment displays. Deze bestaan uit een zevental staaflampjes, aangeduid met a t/m g. Door een bepaalde combinatie van deze lampjes aan te sturen kunnen aldus de decimale cijfers 0 t/m 9 zichtbaar gemaakt worden. Deze cijfers zijn intern in digitale apparatuur vaak in de BCD code vastgelegd. Bij het tonen van de cijfers is derhalve een code-omzetter nodig van de BCD code naar de 7-segment code.

Welke regel uit de waarheidstabel van deze code-omzetter is correct weergegeven?

	W	X	Y	Z	a	b	c	d	e	f	g
	2^3	2^2	2^1	2^0							
a.	0	0	0	0	0	0	0	0	0	0	0
b.	0	1	0	0	0	1	1	0	0	1	1
c.	0	1	1	1	1	0	0	0	1	1	1
d.	1	0	0	1	1	1	1	1	1	1	0

8.5.



Een combinatorische schakeling kan, naar keuze, gerealiseerd worden in afzonderlijke poortschakelingen of de schakeling kan geprogrammeerd worden in een PROM. Bij programmering van de schakeling A met de ingangen x_3 t/m x_0 in een PROM moet de inhoud van adres $x_3x_2x_1x_0 = 1101$ zijn:

- $u_3u_2u_1u_0 = 1110$.
- $u_3u_2u_1u_0 = 1010$.
- $u_3u_2u_1u_0 = 1001$.
- $u_3u_2u_1u_0 = 0110$.

8.6. Als opgave 8.5, voor het adres $x_3x_2x_1x_0 = 0110$

- $u_3u_2u_1u_0 = 0010$.
- $u_3u_2u_1u_0 = 0001$.
- $u_3u_2u_1u_0 = 1110$.
- $u_3u_2u_1u_0 = 0101$.

8.7. Gegeven is een ROM met acht ingangen en acht uitgangen. In deze ROM wil men zo veel mogelijk logische functies realiseren, elk uitgedrukt in vier variabelen a t/m d. Men zet daartoe a t/m d op de eerste vier adresingangen. De overige groep van vier adresingangen wordt daarbij vast ingesteld, waardoor een bepaald gebied van de ROM geselecteerd kan

worden. Daarin worden de functiewaarden van de op de ingangsverzameling {a, b, c, d} gedefinieerde functies geschreven. Met de gegeven ROM omvang kan/kunnen n functies tegelijkertijd gerealiseerd worden, dat wil zeggen dat hun met de ingangscombinatie corresponderende functiewaarden tegelijk beschikbaar zijn. Hoe groot is n?

- n = 1.
- n = 2
- n = 4.
- n = 8.

8.8. Gegeven is een ROM met acht ingangen en acht uitgangen. In deze ROM wil men zo veel mogelijk logische functies realiseren, elk uitgedrukt in vier variabelen a t/m d. Men zet daartoe a t/m d op de eerste vier adresingangen. De overige groep van vier adresingangen wordt vast ingesteld, waardoor een bepaald gebied van de ROM geselecteerd wordt. Daarin worden dan de functiewaarden van de op de ingangsverzameling {a, b, c, d} gedefinieerde functies geprogrammeerd.

Door de 'vaste' instelling te veranderen wordt een ander gebied van het geheugen geselecteerd. Hierin kunnen weer andere functies geprogrammeerd worden, gedefinieerd op de verzameling {a, b, c, d}. Met de gegeven ROM omvang kan/kunnen in totaal m functies gerealiseerd worden, uitgedrukt in de variabelen a t/m d, waarvan er k tegelijk beschikbaar zijn. Voor deze ROM omvang geldt:

- m = 4; k = 1.
- m = 4; k = 8.
- m = 16; k = 1.
- m = 128; k = 8.

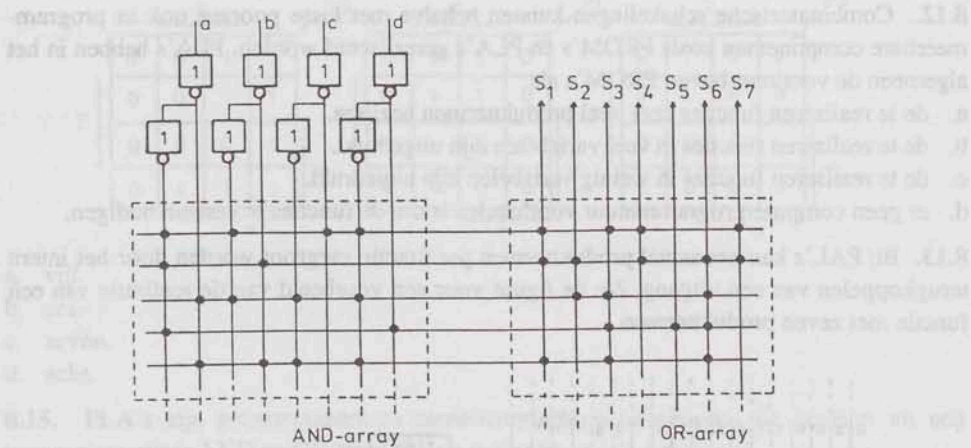
8.9. Een PROM bevat 64 woorden van vier bits. Over de gebruiksmogelijkheden van deze PROM worden vier uitspraken gedaan. Welke uitspraak is juist?

- Deze PROM heeft vier adresingangen en zes uitgangen.
- Uitgaande van het AND-OR (logisch) model realiseert het AND array van de PROM 64 signalen, die overeenkomen met de mintermen van de functies.
- Er kunnen met deze PROM zes functies gerealiseerd worden, uitgedrukt in (maximaal) vier variabelen.
- Deze PROM kan gebruikt worden als vervanger van een combinatorische schakeling met in totaal acht ingangen en vier uitgangen.

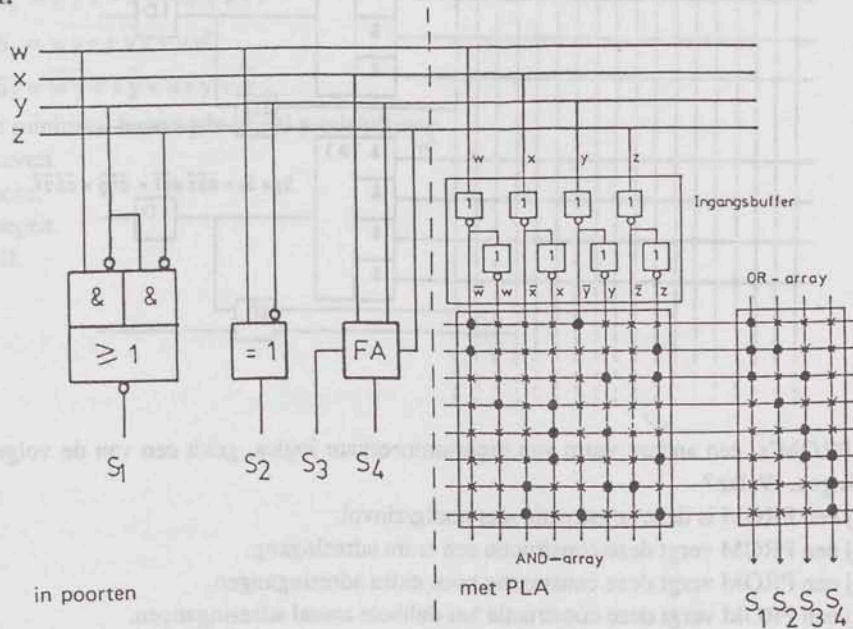
8.10. Gegeven is het schema van een programmable logic array (PLA), dat bestaat uit een AND array en een OR array. Op de plaatsen van een stip zit een verbinding. De uitgang S_3 realiseert de functie

- $S = a\bar{d} + abc + \bar{a}cd.$
- $S = \bar{a}bcd + \bar{a}bcd + \bar{a}d + abcd.$
- $S = (\bar{a} + \bar{b} + c + \bar{d})(\bar{a} + \bar{b} + \bar{c} + \bar{d})(\bar{a} + d)(a + b + \bar{c} + \bar{d}).$
- $S = \bar{a}d + \bar{a}bc + \bar{a}cd.$

Figuur 8.10: zie hierna.



8.11.



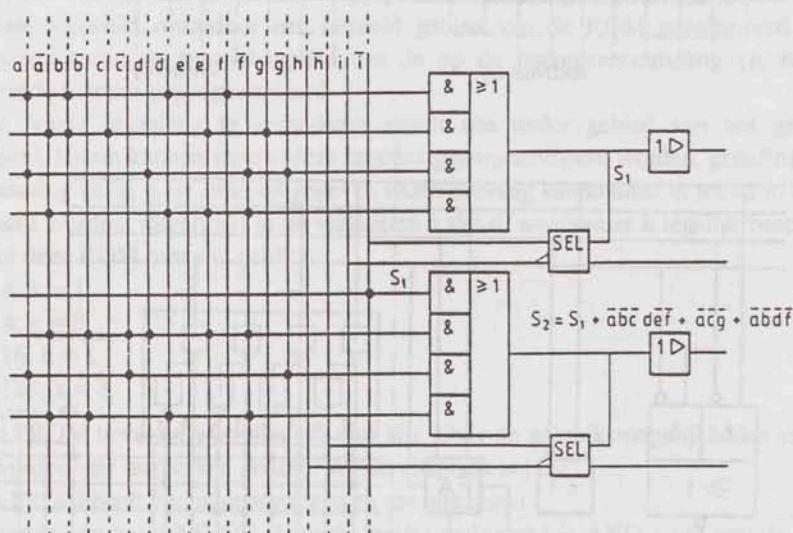
Een combinatorische schakeling kan gerealiseerd worden in aparte poortschakelingen. Het alternatief is programmering in een PLA. De getekende PLA heeft een programmeerbaar AND en een programmeerbaar OR array. Op de plaatsen van een stip zit een verbinding, waardoor de betreffende variabele of produktterm meedoet. Van hoeveel van de signalen S_1 t/m S_4 is de programmering goed uitgevoerd?

- Geen.
- Een.
- Twee.
- Drie.

8.12. Combinatorische schakelingen kunnen behalve met losse poorten ook in programmeerbare componenten zoals PROM's en PLA's gerealiseerd worden. PLA's hebben in het algemeen de voorkeur boven PROM's als

- de te realiseren functies zeer veel produkttermen bezitten.
- de te realiseren functies in veel variabelen zijn uitgedrukt.
- de te realiseren functies in weinig variabelen zijn uitgedrukt.
- er geen computerprogrammatuur voorhanden is om de functies te vereenvoudigen.

8.13. Bij PAL's kan het aantal produkttermen per functie vergroot worden door het intern terugkoppelen van een uitgang. Zie de figuur voor een voorbeeld van de realisatie van een functie met zeven produkttermen.



Voor PROM's, een andere vorm van programmeerbare logica, geldt een van de volgende beweringen. Welke?

- Bij een PROM is deze constructie niet nodig/zinvol.
- Bij een PROM vergt deze constructie een extra adresingang.
- Bij een PROM vergt deze constructie twee extra adresingangen.
- Bij een PROM vergt deze constructie het dubbele aantal adresingangen.

8.14. PLA's zijn combinatorische schakelingen, welke bestaan uit een programmeerbaar AND array en een programmeerbaar OR array. Men wil drie functies in één PLA programmeren. Bij het programmeren streeft men ernaar zo weinig mogelijk produkttermen te gebruiken. Dit om zoveel mogelijk vrije ruimte voor andere functies te kunnen reserveren. Een van de middelen om dit te doen is het zo veel mogelijk gemeenschappelijk benutten van produkttermen bij de programmering van de verschillende functies. Een gevolg van dit streven is dat men de programmering van de verschillende functies niet (altijd) baseert op de eenvoudigste som-van-produktenvorm. Voor de gegeven drie functies is het minimaal benodigde aantal produktlijnen in een PLA

		z						S ₀
		0	0	1	1			
		0	0	1	1			
		0	0	0	1			
		0	1	1	1			
y						w		
	x							

		z						S ₁
		0	1	1	0			
		0	1	1	0			
		1	-	0	0			
		1	-	0	0			
y						w		
	x							

		z						S ₂
		0	1	0	0			
		0	1	0	0			
		1	0	0	-			
		1	0	0	-			
y						w		
	x							

- vijf.
- zes.
- zeven.
- acht.

8.15. PLA's zijn programmeerbare combinatorische schakelingen, die bestaan uit een programmeerbaar AND en OR array. Bij de realisatie van de functies

$$S_0 = w\bar{y}z + x\bar{y} + w\bar{x} + \bar{v}z$$

$$S_1 = w\bar{y}z + \bar{v}z + uz$$

$$S_2 = w\bar{y} + x\bar{y} + uxy + w\bar{x}$$

is het minimaal benodigde aantal produktlijnen

- zeven.
- acht.
- negen.
- elf.

Stelling

Aandelen kopen van een fabriek die soldeer maakt is, op de langere termijn, een slechte belegging.

Appendix symbolen schakelingen I

door prof ir C.H. Eversdijk & ir A.P.Thijssen

Symbolen voor logische schakelingen I

De grondbeginnelsen

A.1	Inleiding	411
A.2	Vorm van de symbolen	412
A.3	Functie-aanduiding	414
A.4	De interface met de fysische wereld	416
A.5	Signaalnamen	417
A.6	Signaalnamen en logische niveaus	418
A.7	Inversie en negatie	421
A.8	Symbolen en formules I	423
A.9	Symbolen en formules II	425
A.10	Dualiteit en positieve en negatieve logica	430

Symbolen voor logische schakelingen II

De afhankelijkheidsnotatie

A.11	De afhankelijkheidsnotatie	434
A.12	Gemeenschappelijke ingangen en uitgangen	436
A.13	De Z afhankelijkheid	438
A.14	G, V en N afhankelijkheid	440
A.15	Enkele toepassingen	443
A.16	EN afhankelijkheid	446
A.17	A afhankelijkheid	449

Appendix symbolen

door Prof. Dr. C.H. Everaak & Dr. A.M. Thijssen

Symbolen voor logische schakelingen I De grondbeginselen

A.1	Inleiding	411
A.2	Weg van de symbolen	412
A.3	Functionele aanduiding	414
A.4	De laatste met de laatste waarde	416
A.5	Signaalnamen	417
A.6	Signaalschakelen en logische bewerkingen	418
A.7	Levens en tegens	421
A.8	Symbolen en formules I	422
A.9	Symbolen en formules II	423
A.10	Formules en positieve en negatieve logica	426

Symbolen voor logische schakelingen II De afhankelijkheidsnotatie

A.11	De afhankelijkheidsnotatie	428
A.12	Omrekeningsregels tussen de notaties	429
A.13	De 2 afhankelijkheidsnotatie	430
A.14	De 3 afhankelijkheidsnotatie	431
A.15	Andere toepassingen	432
A.16	De 2 afhankelijkheidsnotatie	433
A.17	De 3 afhankelijkheidsnotatie	434

Symbolen voor logische schakelingen I

De grondbeginselen

A.1 Inleiding

Digitale Techniek is een relatief jong vakgebied, gekenmerkt door een onstuimige groei van de technische mogelijkheden. Het proces van het ontwerpen van digitale schakelingen heeft de groei van de technische mogelijkheden niet altijd kunnen bijhouden. Het IEC symbolensysteem heeft, op componentenniveau, hieraan zeker positief bijgedragen. Het heeft als katalysator gewerkt op het ontwikkelen van meer systematiek in het ontwerpen, hoewel het oorspronkelijk bedoeld was als documentatiehulpmiddel.

Tot circa 1970 was de complexiteit van logische schakelingen beperkt tot poorten en flip-flops. Voor het weergeven van componenten in tekeningen werden *vormsymbolen* toegepast, voor elk type poort een apart symbool. Deze vormsymbolen zijn gestandaardiseerd in het normblad 76 ANSI/IEEE Y 32E. Tot op de dag van heden worden deze symbolen toegepast. Enkele van deze symbolen zijn in hoofdstuk 4 geïntroduceerd.

Naarmate de complexiteit en de diversiteit van de componenten voor logische schakelingen toenam groeide de behoefte aan meer systematiek bij het weergeven van schakelingen in symbolen. Het was onmogelijk voor elk nieuw circuit een eigen vormsymbool te definiëren. Binnen de IEC (International Electrotechnical Commission), in 1906 opgericht met als doel coördinatie en uniformering van nationale normen op elektrotechnisch gebied, heeft een van de technische commissies, TC3, zich beziggehouden met het opstellen van internationale normen voor grafische symbolen op elektrotechnisch gebied. De werkgroep WG2 van subcommissie SC-3A bestrijkt het terrein van de tekensymbolen voor logische schakelingen. Bij het 'construeren' van symbolen voor logische componenten en schakelingen bleek weldra de noodzaak tot het vastleggen van allerlei tot dan toe intuïtief gehanteerde begrippen in de digitale techniek. In de praktijk bestond op velerlei gebied een grote spraakverwarring. De lezer vergelijke als toelichting hierop de omschrijving van de *timing* van IC's in oudere catalogi. De werkgroep heeft in deze baanbrekend werk gedaan. Een en ander heeft geresulteerd in de op dit moment van belang zijnde publicaties:

- publicatie 617-12 'Graphical symbols for diagrams - Part 12: Binary logic elements', over logicasymbolen van de IEC;
- Nederlandse praktijkrichtlijn NPR 5160, 2e druk, 1984, als toelichting op publicatie 617-12. Hierin is de huidige stand van zaken met betrekking tot het tekenen van logicaschema's vastgelegd;
- NEN 5155, Binaire Logica-elementen, Nederlands Normalisatie-instituut, 1994.

De hierna volgende introductie van het IEC symbolensysteem behandelt het systeem voor zover het nodig is voor het interpreteren van de in de tekst toegepaste symbolen. Voor een volledige bespreking zie de literatuurverwijzingen.

Het doel van een tekening

Een logische schakeling is een realisatie van een logisch ontwerp. Men zou kunnen stellen dat met een goede documentatie van de schakeling ook het er aan ten grondslag liggend logisch ontwerp gedocumenteerd is. De praktijk leert echter anders. Enerzijds bevatten logische schakelingen vaak componenten die geen logische functie bezitten, bijvoorbeeld componenten die zijn toegevoegd om redenen van fanin en fanout, zodat de weg terug naar de logische specificatie van het ontwerp vaak niet eenduidig is. Anderzijds worden logische schakelingen technisch geheel verschillend gerealiseerd, variërend van IC's tot uitvoeringen in pneumatische logica. Er is dus documentatie nodig op de verschillende niveaus van het ontwerptraject en aangepast aan het stadium waarin het ontwerp verkeert. Om toch zoveel mogelijk één symbolentaal te kunnen gebruiken heeft men er bij het opstellen van het normblad IEC 617-12 naar gestreefd *intern in symbolen alle relaties tussen ingangen en uitgangen op logisch niveau vast te leggen*, dat wil zeggen zoveel mogelijk uit te drukken in logische nullen en enen en logische operaties hierop, alsmede met hiervan afgeleide operatoren. Dit streven heeft het voordeel van een uniforme beschrijving van de verschillende fysische realisatievormen in symbolen. Aanvullend hierop moet de interface naar de fysische realiteit gedefinieerd worden. Immers, een onderhoudstechnicus meet spanningen of andere fysische grootheden, terwijl hem/haar vaak niet bekend is wat de logische functie van het signaal is. Ook hierin voorziet IEC 617-12, voor zover het *spanningsniveaus* betreft. Daarnaast moeten ook open-collector en 3-state uitgangsstructies worden weergegeven, alsmede andere details op het grensvlak tussen logica en fysica. Tekeningen van logische schakelingen worden daarom onderscheiden in

- theoretische logicaschema's,
- uitvoeringsschema's.

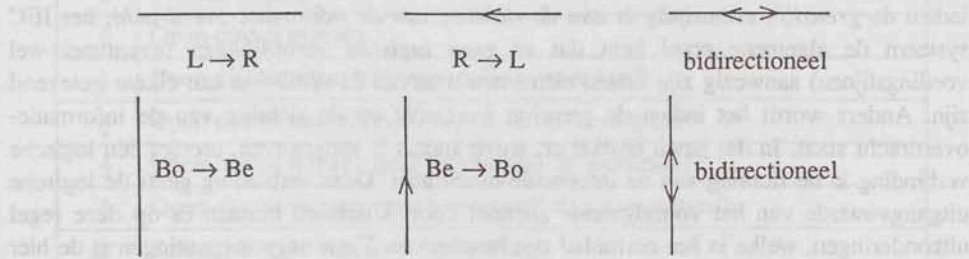
Op een *theoretisch logicaschema* (logic diagram) worden uitsluitend logische grootheden toegepast, zowel intern in de symbolen (volgens de eerder genoemde afspraak) als daarbuiten. Een theoretisch logicaschema documenteert een logisch ontwerp. Nu hebben we reeds gezien dat ook hierin ontwerp-niveaus kunnen worden onderscheiden, van een *theoretisch logisch model* van de schakeling tot een op beschikbare componenten *direct afbeeldbaar logisch model*. Zie hoofdstuk 5. Op *uitvoeringsschema's* wordt daarnaast de *relatie met fysisch meetbare grootheden* vastgelegd. Op uitvoeringsschema's kan aanvullende informatie verstrekt worden, zoals pennummers van IC's en dergelijke. We introduceren de voornaamste concepten van het 617-12 normblad, zodat de in de tekst toegepaste symbolen geïnterpreteerd kunnen worden.

A.2 Vorm van de symbolen

Het kader van de IEC symbolen bestaat uit een *rechthoek van willekeurige afmetingen*. Door de variabele lengte/breedte verhouding is het mogelijk de vorm aan te passen aan het aantal ingangen en/of uitgangen erop. Ook kan men rekening houden met het aantal te vermelden toevoegsymbolen bij een ingang of bij een uitgang. Bij een toenemend aantal

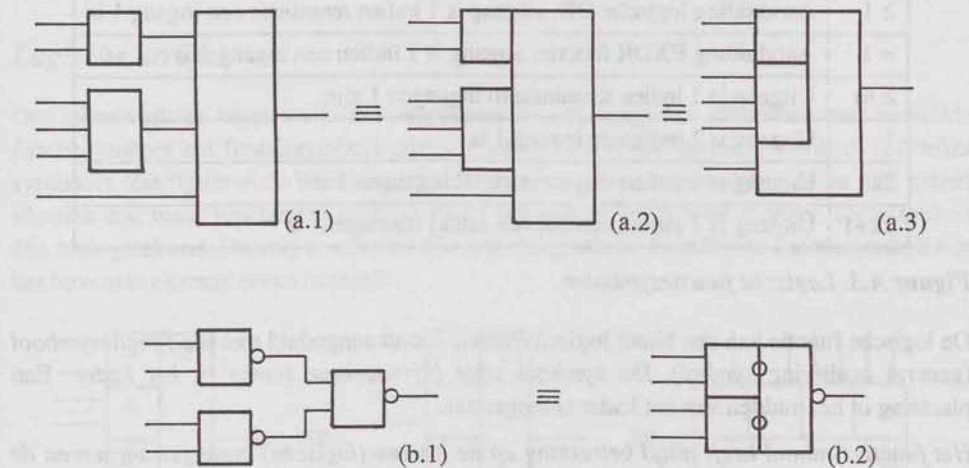
ingangen neemt bij *distinctive shape* symbolen het oppervlak *kwadratisch* toe. Voor *rechthoekige symbolen* is dit globaal *lineair*. Dit bevordert de leesbaarheid van een schema.

In tekeningen wordt als *hoofdrichting van de data- en informatiestroom* aanbevolen de richting van *links naar rechts* aan te houden en/of van *boven naar beneden*. Er worden geen pijlen in verbindinglijnen gezet. Bij *afwijkende richtingen* kan de duidelijkheid vergen dat er in de verbindingslijnen pijlpunten worden opgenomen, zoals in figuur A.1.



Figuur A.1. *Richting informatie-opbouw en data-overdracht.*

Symbolen mogen *geroteerd* worden. Daarbij moeten de inschriften op dezelfde wijze meedraaien als het kader van het symbool. Voor meer informatie hierover zie de NPR 5160, 2^e druk.



Figuur A.2. *Samenvoegen van symbolen.*

Door hun rechthoekige vorm kunnen symbolen gemakkelijk tegen en eventueel in elkaar getekend worden. Figuur A.2 laat enkele mogelijkheden zien. Bij aan en in elkaar getekende symbolen wordt verondersteld dat de *signaalwaarden door de grenslijn* worden doorgegeven. Bij een genegeerde uitgang/ingang moet de negatie-indicator door de grenslijn getekend worden. De in figuur A.2 geschetste middelen maken het mogelijk tekeningen compact uit te voeren. Dit bevordert de overzichtelijkheid. Naast genoemde

middelen om symbolen in tekeningen samen te voegen en data intern door te geven bestaan er nog andere, waaronder een *gemeenschappelijk ingangsblok* (common input block) en een *gemeenschappelijk uitgangselement* (common output element). Hierop worden gemeenschappelijke ingangen of gemeenschappelijke uitgangen van alle eronder gerangschikte componenten aangegeven. Zie voor details het IEC 617-12 normblad.

De hiervoor genoemde middelen om symbolen aan en in elkaar te tekenen vereisen enige spelregels voor de interpretatie van het aldus samengestelde symbool. Een ervan is dat, indien de *grenslijn evenwijdig is aan de richting van de informatie-overdracht*, het IEC systeem de algemene regel kent dat er *geen logische verbindingen* (eventueel wel voedingslijnen) aanwezig zijn tussen elementen waarvan de symbolen aan elkaar getekend zijn. Anders wordt het indien de grenslijn *loodrecht* op de richting van de informatie-overdracht staat. In dat geval bestaat er, tenzij anders is aangegeven, precies één logische verbinding in de richting van de informatie-overdracht. Deze verbinding geeft de logische uitgangswaarde van het voorafgaande element door. Uiteraard bestaan er op deze regel uitzonderingen, welke in het normblad zijn beschreven. Voor onze toepassingen is de hier geformuleerde regel voldoende.

A.3 Functie-aanduiding

&	Aanduiding logische AND; uitgang is 1 indien <i>alle</i> ingangen 1 zijn
≥ 1	Aanduiding logische OR; uitgang is 1 indien <i>tenminste een</i> ingang 1 is
$= 1$	Aanduiding EXOR functie; uitgang is 1 indien een ingang 1 is
$\geq m$	Uitgang is 1 indien tenminste m ingangen 1 zijn
1	Uitgang is 1 indien de ingang 1 is
$2k$	Uitgang is 1 indien een even aantal ingangen 1 is
$2k+1$	Uitgang is 1 indien een oneven aantal ingangen 1 is

Figuur A.3. Logische functiesymbolen.

De logische functie van een binair logisch element wordt aangeduid met een *functiesymbool* (general qualifying symbol). Dit symbool staat *bij voorkeur boven in het kader*. Een plaatsing in het midden van het kader is toegestaan.

Het functiesymbool heeft altijd betrekking op de interne (logische) ingangswaarden en de interne (logische) uitgangswaarden van het door het symbool aangeduide element. Bij het IEC systeem worden de interne waarden altijd uitgedrukt in de logische 0 (False) en de logische 1 (True). (Hoe deze interne logische waarden naar 'buiten' overdragen bespreken we hierna.) Figuur A.3 geeft een overzicht van enkele gangbare functiesymbolen. Naast de in figuur A.3 aangegeven functiesymbolen kent het systeem nog enkele *toevoegsymbolen*, waarmee de elektrische/elektronische eigenschappen van een ingang/uitgang kunnen worden aangegeven, of symbolen die nadere informatie geven over hoe een logische functie van het element is gerealiseerd (wired-AND en dergelijke). Deze toevoegsymbolen zijn

aangegeven in figuur A.4. Indien zij in combinatie met het functiesymbool voorkomen, boven in het symbool, geldt de aanduiding voor alle ingangen respectievelijk alle uitgangen van dat symbool. Staat het toevoegsymbool bij een ingang of uitgang, dan geldt het voor die ingang of die uitgang.

▷	Element met versterkte uitgangen (verhoogde fanout)
▽	3-State uitgang
◇	Open-circuit uitgang
◊	Open-circuit uitgang (vereist pull-up weerstand)
⋈	Open-circuit uitgang (vereist pull-down weerstand)
⊕	Open-circuit uitgang (met interne pull-up weerstand)
⊗	Open-circuit uitgang (met interne pull-down weerstand)
⊠	Schmitt-trigger ingangen (hysteresis)

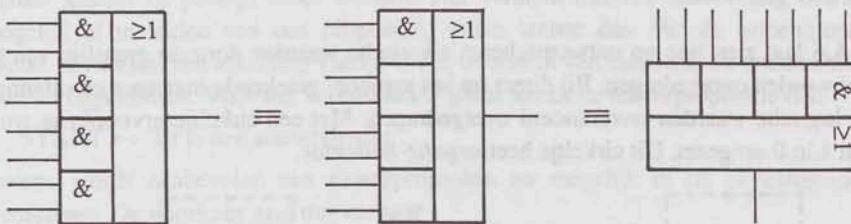
Figuur A.4. Elektrische/elektronische toevoegsymbolen.

Voorbeeld

De combinatie & duidt op de logische AND (intern), terwijl de poort gebufferde uitgangen heeft. □

Logische arrays

Om tekenwerk te besparen is het toegestaan in een array van elementen met hetzelfde functiesymbool het functiesymbool alleen te vermelden in het bovenste symbool (of linker symbool). Zie figuur A.5. Wanneer er twee of meer groepen symbolen zijn die niet geheel identiek zijn maar functioneel toch bij elkaar horen, dan worden de groepen soms toch in één blok getekend. Daarbij worden in elke nieuwe groep de betreffende functiesymbolen in het bovenste element ervan herhaald.



Figuur A.5. Aanduiding logische arrays.

Opmerking

In de IEC publicaties wordt gesproken over de interne logische *toestand* van ingangen en uitgangen. Buiten de symbolen spreekt men vaak over logische *waarden*. Wij hebben er

eerder de voorkeur aan gegeven het begrip *toestand* te reserveren voor het beschrijven van *geheugenwerking*. De logische toestand is dan zichtbaar/observeerbaar aan de logische waarde van het ermee corresponderende uitgangssignaal. Voor het overige spreken we over logische waarden. □

Opmerking

Niet genormaliseerde informatie, bijvoorbeeld de naam van een component in een groter geheel, mag aan de symbolen toegevoegd worden. Afgesproken is deze informatie tussen rechte haken [.....] te plaatsen. □

A.4 De interface met de fysische wereld

Bij een ontwerp dat top-down ontstaat hanteert de ontwerper in eerste instantie uitsluitend logische variabelen en operatoren. Na enige tijd ontstaat een *logisch ontwerp* van de te realiseren schakeling. In dit stadium corresponderen de logische nullen en enen intern in de symbolen meestal direct met de logische nullen en enen, welke de waarheidswaarden van externe signaallijnen aangeven. In een vroeg ontwerp stadium geldt voor de meeste ingangen/uitgangen op symbolen

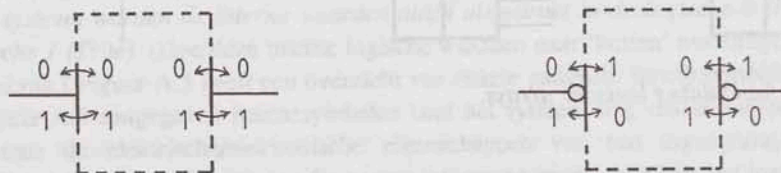
intern 0 $\leftrightarrow$ extern 0

intern 1 $\leftrightarrow$ extern 1

Naarmate een ontwerp vordert en men meer te maken krijgt met de eigenschappen van de componenten wordt dit anders. Zo keren de meeste ingangsaanpassingen (fanin invertoren) en uitgangsaanpassingen (totem pole bijvoorbeeld) het signaalniveau om. De logische beschrijving van een signaalinversie is de *negatie*. Het IEC systeem bevat middelen om dit aan de rand van symbolen aan te geven, zowel op logisch niveau als op elektrisch niveau. Voor de gebruiker van logische schema's betekent dit dat logische ontwerpschema's en diagrammen kunnen verschillen van uitvoeringsschema's.

De negatie-indicator

Figuur A.6 laat zien hoe op ontwerpschema's logische waarden door de grenslijn van een symbool worden overgedragen. Bij direct op het symbool getekende ingangen en uitgangen worden logische waarden onveranderd overgedragen. Met een cirkeltje ervoor/erna wordt 0 in 1 en 1 in 0 omgezet. Dit cirkeltje heet *negatie-indicator*.

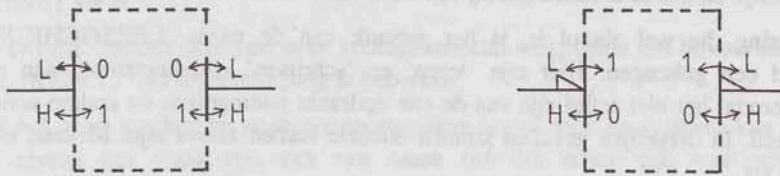


Figuur A.6. Negatie-indicator.

De negatie-indicator kan/moet ook worden toegepast bij overgangen van *embedded symbols*. De negatie-indicator wordt dan dwars door de grenslijn getekend. Zie figuur A.2.

De polariteitsindicator

Op *uitvoeringsschema's* worden externe signaalwaarden uitgedrukt in H en L. Binnen de symboolkaders blijft alles uitgedrukt in logische variabelen en operatoren. De *polariteitsindicator* (met een driehoekje) maakt het mogelijk elk gewenst verband tussen externe signaalwaarden en interne logische waarden weer te geven. Zie figuur A.7.



Figuur A.7. Polariteitsindicator.

Met de polariteitsindicator is de interface tussen de fysische wereld zoals spanningen die men in een schakeling kan meten en de logica intern in de symboolkaders vastgelegd. Voor spanningen is de interface in het IEC 617-12 normblad vastgelegd, voor bijvoorbeeld pneumatische logica (nog) niet. Wel kan men besluiten ook hier H en L te hanteren, mits duidelijk omschreven is wat ermee bedoeld wordt.

A.5 Signaalnamen

Signaalnamen geven de betekenis/functie van een signaal weer. Signaalnamen moeten daartoe duidelijk zijn, hetgeen slechts het geval is indien aan bepaalde voorschriften is voldaan. Zo is de naam START/STOP bij een signaalingang niet duidelijk, omdat deze naam een tegenstrijdigheid bevat. Tevens is niet duidelijk of bij 'aanwezigheid' van het signaal gestart of gestopt moet worden. Het verdient daarom aanbeveling een naam zo mogelijk af te leiden van een propositie, welke laatste dan met de bijbehorende naam (welke er meestal een afkorting van is) in een overzicht van namen in de documentatie moet worden opgenomen. Voor het vorenstaande geval wordt de naam/propositie dan

START $\leftrightarrow$ 'Er is een startopdracht'.

Tevens wordt aanbevolen een naam/propositie zo mogelijk in de bevestigende zin te formuleren. De voorkeur gaat dus uit naar

ALARM $\leftrightarrow$ 'Er is een alarm'

en niet naar

NIETALARM $\leftrightarrow$ 'Er is geen alarm'.

Dit laatste is minder duidelijk, al was het alleen maar omdat 'alarm' in het laatste geval genoteerd zou worden als NIET(NIETALARM) of

NIETALARME

Formeel is deze aanduiding echter juist.

Soms komt men bij de naamgeving in de moeilijkheden. Zo kan het drukken op een knop de functie 'START' hebben indien een apparaat gestopt is, terwijl dezelfde knop de functie 'STOP' heeft indien het apparaat loopt. Men komt dan vrijwel automatisch op de naam START/STOP bij de knop. Nu is in dit geval het probleem geschapen door de wijze waarop de bediening van het apparaat gerealiseerd is en niet door de wijze waarop symbolen worden opgesteld! Over het algemeen is een dergelijke koppeling van functies onoverzichtelijk en dus af te raden (en bij voorkeur dus niet: niet aan te raden).

Een toepassing die wel zinvol is, is het gebruik van de naam 'LEES/SCHRIJF' bij bijvoorbeeld een geheugen. Hier zijn 'lezen' en 'schrijven' ondubbelzinnig aan elkaar gekoppeld omdat het niet actief zijn van de ene opdracht automatisch de andere activeert, en omgekeerd. In dergelijke gevallen kunnen dubbele namen zinvol zijn. Meestal noteert men dit dan als

'LEES(L)/SCHRIJF(H)' of 'LEES/SCHRIJF(0/1)'

of een andere toevoeging die aangeeft hoe de betreffende deelfunctie wordt geactiveerd.

Opmerking

In de beginperiode van het toepassen van SSI/MSI IC's heeft men er duidelijk naar gestreefd het instellen van functies te koppelen aan aparte ingangen. Zie bijvoorbeeld tellerontwerpen met ingangen als LD (load), R (reset), INH (stop) en andere namen. Deze koppeling wordt momenteel losgelaten. De in te stellen functies worden *gecodeerd*. De bij de gewenste functie behorende ingangscombinatie moet dan op de instellingen worden aangeboden. De naam bij dergelijke instellingen heeft dan meer een aanduidende dan een functionele betekenis. Vergelijk de aanduiding setingang en resetingang bij latches (hoofdstuk 9, paragraaf 9.1). □

A.6 Signaalnamen en logische niveaus

Bij een logisch ontwerp is het verband tussen *naam* en *waarheidswaarde* vast. Een naam of de erdoor aangeduide propositie is waar als het ermee corresponderende signaal de logische waarde '1' heeft. Het ligt voor de hand dit in de realisatiefase voort te zetten. Bij een *vast verband* tussen naam en signaalwaarde heeft men de keuze uit twee conventies, *positieve* en *negatieve logica*.

Bij *positieve logica* wordt het 'waar' zijn van een propositie/naam/variabele voorgesteld door respectievelijk afgebeeld op het met H aangeduide spanningsniveau en het 'niet waar' zijn op het met L aangeduide niveau. (Welke gebieden voor H en voor L gedefinieerd zijn wordt apart omschreven. De ligging van H/L is onder andere technologie-afhankelijk.) In de praktijk volgt men (bij een vaste keuze) meestal de positieve logica conventie. Bij negatieve logica beeldt men precies andersom af. Dan correspondeert 'waar' met het spanningsniveau L. Gaan we uit van de *logische specificatie* voor een ontwerp, dan

ontstaan er afhankelijk van de keuze voor positieve of voor negatieve logica *verschillende elektrische specificaties* in termen van H/L tabellen. Zie hoofdstuk 4.

Dit streven naar uniformering van de afbeelding van de waarheidswaarden van namen/proposities naar H/L niveaus wordt doorkruist door technische mogelijkheden en/of voorkeuren. Een buffer-invertor aan een ingang bijvoorbeeld keert het signaalniveau om. Bij veel tellers en andere MSI componenten is de ingang 'RESET' actief indien het niveau L wordt aangeboden. Bij een strikt toepassen van positieve logica zou dan een tweede invertor voor de resetingang nodig zijn om extern te kunnen voldoen aan

RESET $\leftrightarrow$ H

Deze tweede invertor laat men in de praktijk meestal weg, mede om bijvoorbeeld de setup time $t_{su}(\text{RESET})$ van de resetingang te beperken.

Tabel A.1 geeft aan hoe bij positieve en negatieve logica toch aangegeven kan worden met welk niveau het waar zijn van een naam (of het actief zijn van een instelling) correspondeert. Men doet dit door zo nodig de naam te voorzien van een *negatiestreep*.

De afspraak positieve of negatieve logica te gebruiken geldt in het algemeen voor een hele tekening of ontwerp. De afspraak moet wel vermeld worden op de tekening. Hierna ligt de relatie logische specificatie $\leftrightarrow$ elektrische specificatie/werking vast. (We zullen nog beredeneren dat er in de praktijk redenen kunnen bestaan om van de vaste conventie af te wijken.)

logische conventie	naam	naam waar/instelling actief indien het niveau gelijk is aan
positieve logica	< naam >	H
	< $\overline{\text{naam}}$ >	L
negatieve logica	< naam >	L
	< $\overline{\text{naam}}$ >	H

Tabel A.1. Naamgeving bij positieve en negatieve logica.

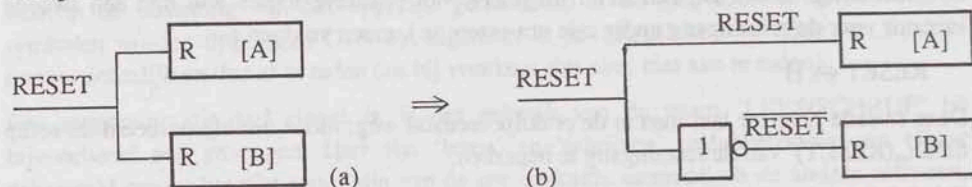
Afspraak

Op schema's met positieve logica en op schema's met negatieve logica past men bij voorkeur de *negatie-indicator* toe om negaties/inversies aan te geven. Een en ander is in overeenstemming met de naamgeving, waarbij het 'andere actieve niveau' ook met een negatiestreep boven de naam wordt aangeduid. $\square$

Positieve of negatieve logica is, zeker op onderhoudschema's, niet altijd even duidelijk bij het interpreteren van het schema. Het volgende voorbeeld gaat hier op in.

Voorbeeld

In een *logisch ontwerp* moet het resetsignaal 'RESET' twee modules bedienen. Zie figuur A.8.a. Tijdens de realisatie van het ontwerp blijkt dat het handig is om voor module A een component te selecteren, waarvan het actieve niveau van de resetingang het niveau H is. Voor module B blijkt L handig te zijn. Het logisch ontwerpschema wordt dus aangepast en omgezet in dat van figuur A.8.b. Deze figuur nu is direct afbeeldbaar op de beschikbare componenten.



Figuur A.8. Aansluitschema resetingang.

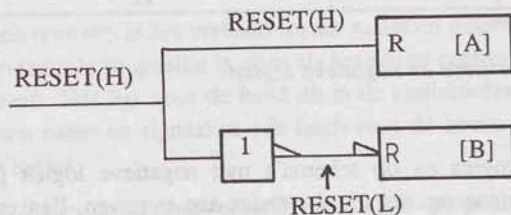
De namen in figuur A.8.a en figuur A.8.b zijn vermeld overeenkomstig de positieve logica conventie (tabel A.1). Op schema (b) komen dan twee namen voor, RESET bij A en $\overline{\text{RESET}}$ bij B. Toch worden beide units *tegelijk* geset of gereset. De naamgeving suggereert echter een verschil in functie, welke er in dit geval niet is. Hetzelfde bezwaar geldt voor schema's waarop negatieve logica is toegepast. De gesignaleerde mogelijkheden tot misverstanden komen voort uit het feit dat er voor het *aangeven van fysische verschillen*, welk niveau 'actief' is, een *logische operator* wordt toegepast. Op *uitvoeringsschema's* past men daarom bij voorkeur *individuele logica* toe. (In Nederland zelfs voorgeschreven in de norm.) Bij elke naam wordt dan genoteerd of de bijbehorende functie actief is met het hoge niveau,

naam(H)

of met het lage niveau,

naam(L).

Het schema van figuur A.8 wordt in individuele logica dat van figuur A.9.



Figuur A.9. Aansluitschema in individuele logica. □

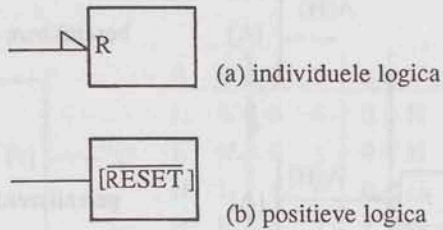
Individuele logica is op het realisatieniveau in het algemeen wat gemakkelijker te interpreteren dan positieve en negatieve logica. In een bepaalde fase van een ontwerp, tijdens het uitvoeringsstadium, gaat men dan ook meestal over van positieve logica naar

individuele logica. Wanneer men overstapt hangt van persoonlijke voorkeur en ervaring van de ontwerper af. Op *uitvoeringsschema's* wordt ten alle tijde *individuele logica* aanbevolen! Ook individuele logica geldt voor een hele tekening of ontwerp. Het gebruik van polariteitsindicatoren is dan voorschreven, in plaats van externe negatie-indicatoren. Het gaat in dit stadium over logische spanningen! In catalogi van geïntegreerde circuits leest men vaak de aanduiding 'actief Laag' en 'actief Hoog' bij een signaalnaam of ingang van een circuit. Men bedoelt hiermee dan te zeggen dat de propositie

'<naam> is present'

waar is als het niveau op de met de naam corresponderende lijn of signaalpunt Hoog (actief Hoog) of Laag (actief Laag) is. Uiteraard verdient het de voorkeur met genormaliseerde aanduidingen te werken, aanduidingen als naam(H), etc.

Het feit dat een instellingang actief is met een bepaald logisch niveau kan men op verschillende wijzen aangeven. In figuur A.10.a is de genormaliseerde aanduiding toegepast. Het toevoegsymbool R aan de ingang is in de norm gedefinieerd. De polariteitsindicator geeft aan dat het actief zijn ('RESET present') van RESET correspondeert met een uitwendig niveau L op de resetingang.



Figuur A.10. Polariteitsaanduiding.

Figuur A.10.b toont een niet-genormaliseerde aanduiding, welke hetzelfde aangeeft als het genormaliseerde symbool. Omdat het voor de gangbare logische schakelingen vrijwel altijd mogelijk is de functie in een genormaliseerd symbool uit te drukken verdient de genormaliseerde aanduiding de voorkeur.

A.7 Inversie en negatie

Definitie

Een *inverter* is een schakeling, die een hoog niveau H aan de ingang omzet in een laag niveau L aan de uitgang en omgekeerd een laag ingangsniveau L in een hoog uitgangsniveau H ($L \rightarrow H$ en $H \rightarrow L$). $\square$

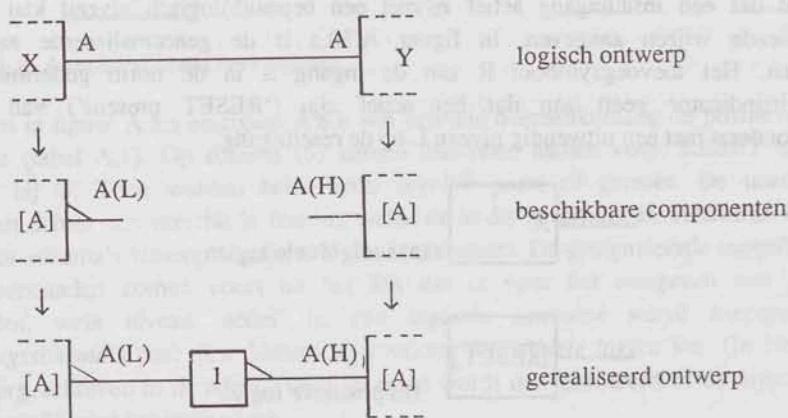
Definitie

Een *negator* zet een aan de ingang aangeboden *logische waarde* om in de tegengestelde waarde aan de uitgang ($0 \rightarrow 1$ en $1 \rightarrow 0$ of $T \rightarrow F$ en $F \rightarrow T$). $\square$

De begrippen *inversie* en *negatie* zijn niet identiek. Een invertor behoeft niet noodzakelijk een logische negatie in te houden, zoals we aan het volgende voorbeeld zien.

Voorbeeld

Figuur A.11.a toont een gedeelte van een ontwerpschema. Bij het omzetten hiervan in een schakeling blijkt dat voor de realisatie van de module X een bouwsteen beschikbaar is die het signaal A(L) afgeeft. Voor module Y is een bouwsteen beschikbaar die het ingangssignaal A(H) nodig heeft (figuur A.11.b). Bij de afbeelding van het logisch ontwerp op de beschikbare modules blijkt het dus nodig te zijn de *signaalniveaus* van de bouwstenen aan te passen. Zie figuur A.11.c. De toepaste *invertor* verricht geen logische negatie (dit is immers niet nodig), maar zet alleen een hoog niveau H in een laag niveau L om en omgekeerd.



Figuur A.11. Voorbeeld van realisatie met een invertor. □

Het omgekeerde komt ook voor, namelijk dat op het logisch ontwerpschema negatoren voorkomen en op het uitvoeringsschema geen expliciet getekende invertoren.

In de praktijk blijken er zeer veel fouten gemaakt te worden bij het afbeelden van logische waarden op H en op L spanningsniveaus. Het is van groot belang op dit punt nauwkeurig te werken en als het kan een vaste conventie aan te houden.

Opmerking

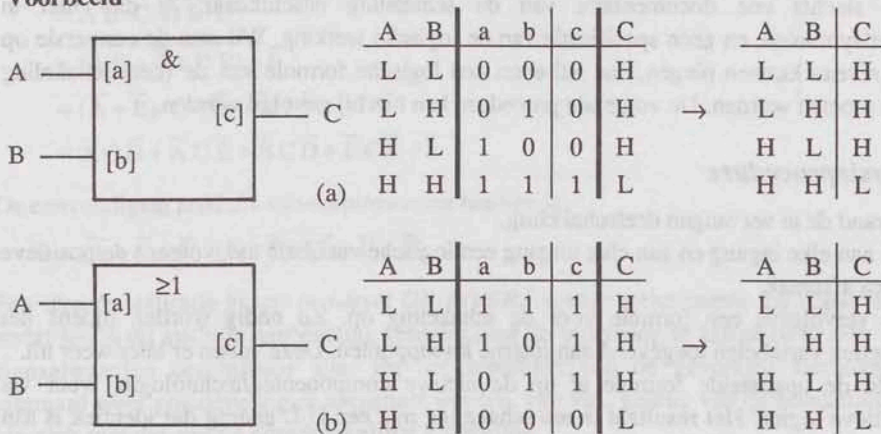
De naamgeving van componenten zoals NAND en NOR is afgeleid van de hiervoor geschetste problematiek. Eigenlijk zou men deze poorten moeten aanduiden als 'AND/OR met geïnverteerde uitgang'. In de praktijk heeft men dit veelal ondervangen door reeds op logisch niveau de *logische* operatoren NAND (NOT-AND) en NOR (NOT-OR) te gebruiken. Vergelijk paragraaf 5.3. □

A.8 Symbolen en formules I

Alvorens het symbolensysteem verder te introduceren is het nuttig in te gaan op enkele problemen bij het toepassen ervan. Bespreking van deze problemen vergroot het inzicht.

Een goed top-down gestructureerd ontwerpproces resulteert op zeker moment in een *logisch ontwerp* van de te realiseren schakeling. Dit logisch ontwerp moet vervolgens worden afgebeeld op componenten welke, in ons geval, werken met H en L spannings-niveaus. Bij de afbeelding moet het logisch ontwerp meestal verder worden uitgewerkt en aangepast aan de beschikbare componenten. Passings/afbeeldingsproblemen lost men meestal op het logisch ontwerpniveau op en niet via uitgebreide manipulaties op H/L tabellen. Daartoe kent men aan de componenten meestal ook een logische functie toe, via de positieve logica interpretatie bijvoorbeeld. Het logisch ontwerp wordt vervolgens via formulemanipulaties omgezet in een *direct afbeeldbaar logisch ontwerp* (detailed logic design). Een argument te meer om afbeeldingsproblemen op logisch niveau op te lossen is het feit dat symbolen voor componenten niet ondubbelzinnig vastliggen op grond van de genormaliseerde regels. Voor veel bouwstenen zijn verschillende symbolen mogelijk, hetgeen het vergelijken op symbolisch niveau lastig maakt.

Voorbeeld



Figuur A.12. Verschillende symbolen met dezelfde H/L tabel.

In figuur A.12.a is het bij een schakeling behorend symbool beschreven. Ernaast is aangegeven hoe hieruit de H/L tabel wordt afgeleid. We beginnen met in de eerste twee kolommen alle mogelijke combinaties van H/L waarden van de punten A en B te noteren. Deze waarden worden naar interne logische waarden omgezet (in de kolommen drie en vier) via $H \rightarrow 1$ en $L \rightarrow 0$ (geen polariteitsindicator). Voor het gemak zijn interne logische variabelen a t/m c geïntroduceerd. Omdat het geen genormaliseerde aanduidingen betreft worden ze tussen rechte haken geplaatst.

Het functiesymbool & legt het logisch verband vast tussen de interne logische ingangswaarden van a en b enerzijds en de logische waarde van het interne uitgangssignaal c

anderzijds. Hiermee ligt ook kolom vijf van de tabel vast. De interne logische waarde van de variabele c wordt vervolgens naar het uitgangssignaal C vertaald. Op de conversie is de polariteitsindicator van toepassing, zodat $1 \rightarrow L$ en $0 \rightarrow H$. Hieruit volgt de tweede tabel, geheel in spanningsniveaus.

Passen we dezelfde procedure toe op het symbool in figuur A.12.b, dan ontstaat dezelfde H/L tabel. De met beide symbolen beschreven componenten blijken identiek te zijn. $\square$

In internationaal verband wordt gewerkt aan het opstellen van een *symbolenbibliotheek* voor standaard IC's. Van veel IC's is het symbool reeds vastgesteld. Met betrekking tot de in het voorbeeld geschetste problematiek lost dit echter niets op. Immers, het afbeelden van een logisch ontwerp resulteert zeker niet zonder meer in een standaardsymbool of in een schema bestaande uit standaardsymbolen. Conclusie: Passingsproblemen moeten bij voorkeur op *logisch niveau* worden uitgezocht.

In de praktijk komt men vaak het volgende probleem tegen. Van een gegeven schakeling moet het ontwerp herzien worden (redesign) omdat de schakeling bijvoorbeeld in een andere technologie gerealiseerd gaat worden, met specifieke voorkeuren voor andere logische operatoren. Uiteraard verdient het aanbeveling om een dergelijke aanpassing uit te voeren met als vertrekpunt een specificatie op een *hoger logisch niveau*. In de praktijk is meestal slechts een documentatie van de schakeling beschikbaar, al dan niet in standaardsymbolen, en geen specificatie van de logische werking. Wil men de conversie op logisch niveau kunnen plegen, dan zal eerst een logische formule van de (deel)schakeling bepaald moeten worden. De volgende procedure kan hierbij gevolgd worden.

Conversieprocedure

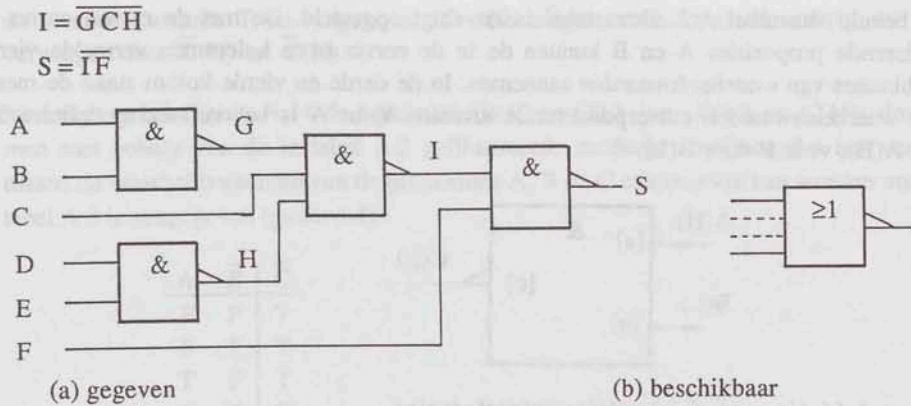
1. Omrand de te vervangen deelschakeling.
2. Ken aan elke ingang en aan elke uitgang een logische variabele toe, volgens de positieve logica afspraak.
3. Stel vervolgens een formule voor de schakeling op. Zo nodig worden tijdens het opstellen variabelen toegekend aan interne knooppunten. Deze vallen er later weer uit.
4. Beeld de opgestelde formule af op de nieuwe componenten/technologie, weer via positieve logica. Het resultaat is een schakeling met een H/L gedrag dat identiek is aan dat van de oorspronkelijke schakeling.
5. Pas aan de randen eventueel de polariteit van signalen aan. (Indien bijvoorbeeld de zender een uitgangsinvertor heeft en de ontworpen schakeling als ontvanger heeft een ingangsinvertor, dan mogen deze beide geschrapt worden, indien relevant.)

Voorbeeld

De schakeling in figuur A.13 heeft zes ingangen en één uitgang. In positieve logica gelden de volgende formules:

$$G = \overline{A}B$$

$$H = \overline{D}E$$



Figuur A.13. Conversie van AND-INVERT naar OR-INVERT realisatie.

Hieruit volgt door substitutie:

$$\begin{aligned}
 S &= \overline{IF} \\
 &= \overline{\overline{GCH} \cdot F} \\
 &= \overline{AB \cdot C \cdot DE \cdot F} \\
 &= \overline{(AB + \overline{C} + DE) + \overline{F}} \\
 &= (\overline{A} + \overline{B}) \cdot C \cdot (\overline{D} + \overline{E}) + \overline{F} \\
 &= \overline{A} \overline{C} \overline{D} + \overline{A} \overline{C} \overline{E} + \overline{B} \overline{C} \overline{D} + \overline{B} \overline{C} \overline{E} + \overline{F}
 \end{aligned}$$

De eenvoudigste produkt-van-sommenvorm hiervan is:

$$S = (\overline{A} + \overline{B} + \overline{F}) \cdot (C + \overline{F}) \cdot (\overline{D} + \overline{E} + \overline{F})$$

Een directe realisatie in een *two-level OR-INVERT* vorm vereist enkele ingangsinvertoren, zodat het in dit stadium aanbeveling verdient naar de omgeving te kijken of er ook inverse signaalwaarden beschikbaar zijn. Bij het afbeelden van de gevonden formules moet uiteraard weer positieve logica toegepast worden. Het H/L gedrag van de schakeling is dan identiek met dat van de oorspronkelijke schakeling. $\square$

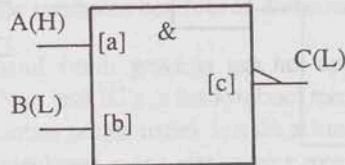
A.9 Symbolen en formules II

Men kan zich afvragen of de in sectie A.8 beschreven procedure niet toegepast kan worden met de oorspronkelijke namen/variabelen in het gegeven schema (individuele logica). We werken daartoe de volgende voorbeelden uit.

Voorbeeld

In figuur A.14 is een symbool getekend met de namen A(H), B(L) en C(L), in individuele logica. We leiden het logisch verband tussen de namen A en B enerzijds en C anderzijds af

met behulp van tabel A.2. Deze tabel is als volgt opgesteld. De met de namen corresponderende proposities A en B kunnen de in de eerste twee kolommen vermelde vier combinaties van *waarheidswaarden* aannemen. In de derde en vierde kolom staan de met deze waarheidswaarden corresponderende *niveaus*. Voor A is het verband gedefinieerd door A(H); voor B door B(L).



Figuur A.14. Naamaanduiding bij individuele logica.

Er staan geen driehoekjes aan de ingangen. Voor beide ingangen correspondeert uitwendig H met inwendig 1. Dit is weergegeven in de vijfde en zesde kolom van tabel A.2 onder a en b. Hieruit volgt, gezien het gegeven functiesymbool &, de zevende kolom onder c.

A	B	A	B	a	b	c	C	C		A	B	C
F	F	L	L	0	1	0	H	F		F	F	F
F	T	L	H	0	0	0	H	F	→	F	T	F
T	F	H	L	1	1	1	L	T		T	F	T
T	T	H	H	1	0	0	H	F		T	T	F

$$C = A \cdot \bar{B}$$

Tabel A.2. Bepaling van een formule bij een symbool.

Aan de uitgang staat de *polariteitsindicator*. Inwendig 1 correspondeert met uitwendig L. Hieruit volgt de kolom in H en L onder C. Vervolgens wordt de uitgangsspanning via C(L) weer vertaald in T en F. Daarmee is ook de negende kolom ingevuld. In een aparte tabel is het verband tussen de waarheidswaarden van de proposities A en B enerzijds en van C anderzijds nogmaals samengevat (eerste, tweede en negende kolom).

$$C = A \text{ EN } \bar{B} \text{ of } C = A \cdot \bar{B}$$

Deze formule beschrijft het verband tussen de waarheidswaarden van de proposities A, B en C zoals dit door de door het symbool van figuur A.14 gerepresenteerde schakeling wordt gelegd.

Opmerking

Het is gemakkelijk in te zien dat men op grond van de geldende afspraken het bijschrift A(H) mag vervangen door $\bar{A}(L)$. Immers, heeft de propositie

‘A is present’

de waarheidswaarde F, dan heeft de propositie

‘NOT-A is present’

de waarheidswaarde T en omgekeerd. Evenzo mag men de bijschriften B(L) en C(L) vervangen door $\bar{B}(H)$ en $\bar{C}(H)$. $\square$

Vervangt men in figuur A.14 de bijschriften B(L) en C(L) door $\bar{B}(H)$ en $\bar{C}(H)$, dan kan men met behulp van de in tabel A.2 geïllustreerde methode verifiëren dat het verband tussen de waarheidswaarden van de proposities A, B en C samengevat kan worden zoals in tabel A.3 is aangegeven (ga dit na!).

A	$\bar{B}$	$\bar{C}$
F	F	T
F	T	T
T	F	T
T	T	F

Tabel A.3. Verband tussen de proposities A, B en C.

Uit tabel A.3 volgt

$$\bar{C} = \bar{A} \text{ OF } \bar{B} \text{ of } \bar{C} = \bar{A} + \bar{B},$$

welke formule op grond van de wetten van de logica gelijkwaardig is met de reeds gevonden formule

$$C = A \cdot \bar{B}.$$

Voorbeeld

Het symbool in figuur A.15 leidt met de bijbehorende (ten opzichte van figuur A.14 gewijzigde!) namen tot dezelfde formule, namelijk

$$C = A \cdot \bar{B}.$$

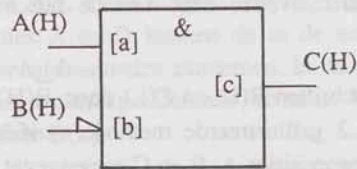
(De wijziging bestaat eruit dat per ingang/uitgang naam(L) $\rightarrow$ naam(H) gaat en dat er een polariteitsindicator is toegevoegd.) We vinden deze formule zoals in tabel A.4 is aangegeven.

A	B	A	B	a	b	c	C	C		A	B	C
F	F	L	L	0	1	0	L	F		F	F	F
F	T	L	H	0	0	0	L	F	$\rightarrow$	F	T	F
T	F	H	L	1	1	1	H	T		T	F	T
T	T	H	H	1	0	0	L	F		T	T	F

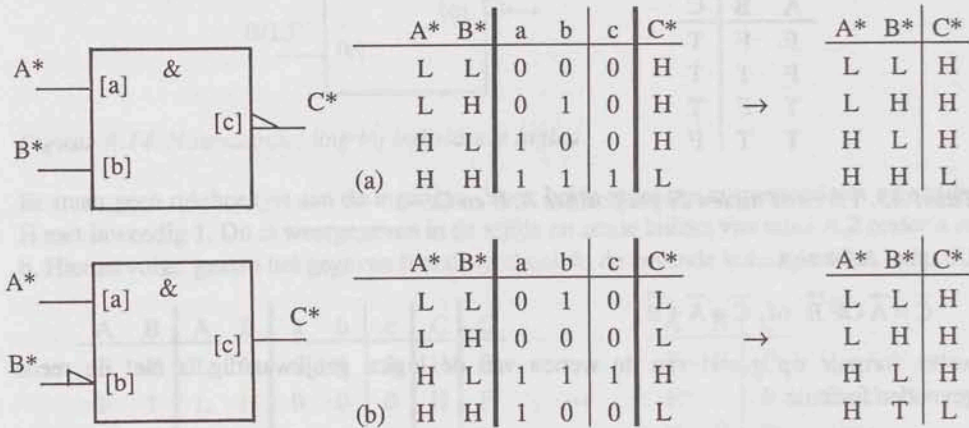
$$C = A \cdot \bar{B}$$

Tabel A.4. Bepaling van een formule bij een symbool.

De bij de symbolen/schakelingen behorende H/L tabellen zijn echter verschillend, zoals figuur A.16 aantoont. Hoewel de formules identiek zijn geldt dit niet voor de bijbehorende H/L tabellen. De schakelingen kunnen elkaar dus *niet* vervangen. (Het wordt aan de lezer overgelaten te beredeneren waar in het bovenstaande de 'fout' gemaakt is.)



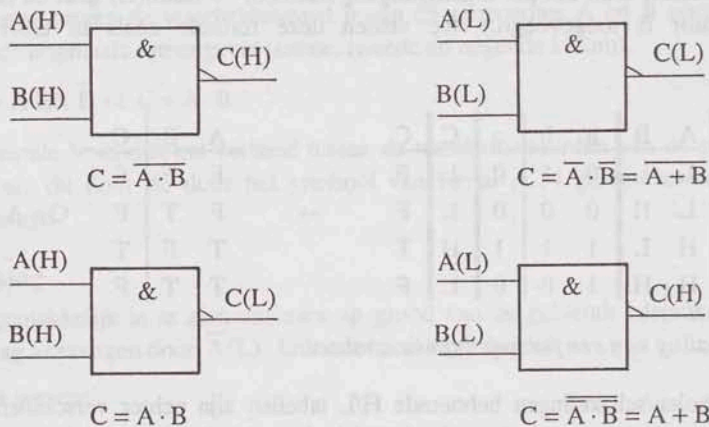
Figuur A.15. Naamaanduiding bij individuele logica.



Figuur A.16. Afleiding H/L tabellen.

Voorbeeld

Figuur A.17 geeft enkele formules die de functie van een component voor een bepaalde interpretatie van A, B en C specificeren. Een en ander kan met behulp van de in tabel A.2 beschreven methode worden afgeleid.



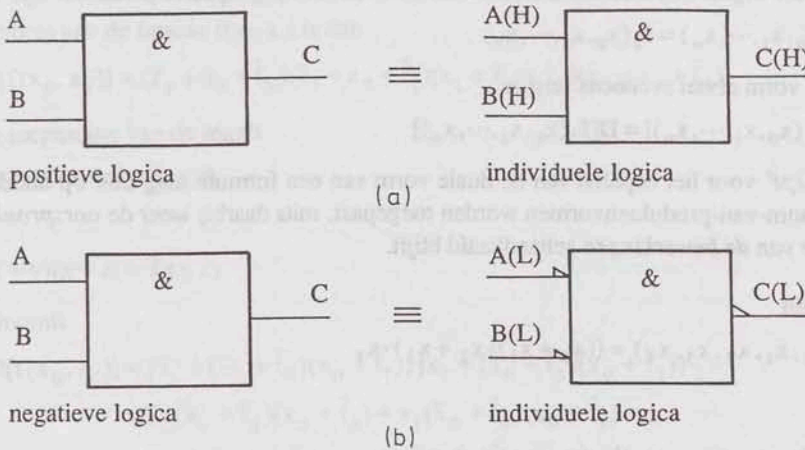
Figuur A.17. Verschillende mogelijke functies van eenzelfde bouwsteen, afhankelijk van de interpretatie van A, B en C.

Uit het voorgaande blijkt dat bij toepassing van *individuele logica* ontwerper en gebruiker de vrijheid hebben om de *signaallnamen* zo te kiezen dat deze het best passen bij *hun interpretatie* of *hun beschrijving* van de gerealiseerde schakeling, mits zij de juiste toevoegingen (H) of (L) er achter plaatsen. Deze vrijheid mist men bij toepassing van het systeem van *positieve logica* of dat van *negatieve logica*. Bij positieve logica dient men een signaalnaam steeds zo te kiezen dat de waarheidswaarde T van de bijbehorende propositie correspondeert met het hoge niveau (toevoeging (H)). Bij negatieve logica moet T steeds corresponderen met het lage niveau (toevoeging (L)). Aangezien dit geldt voor alle in het schema voorkomende signaallnamen kan men dan de toevoegingen (H) en (L) weglaten, mits op het schema wordt vermeld dat positieve (of negatieve) logica is toegepast.

De in figuur A.18 getoonde symbolen realiseren alle de formule

$$C = A \cdot B$$

Voorts representeren de beide symbolen van figuur A.18.a dezelfde schakeling. Ook de symbolen van figuur A.18.b representeren dezelfde schakeling. De in figuur A.18.a gerepresenteerde schakeling verschilt echter van die van figuur A.18.b. Zij hebben namelijk verschillende H/L tabellen.



Figuur A.18. Symbolen in verschillende logische conventies. □

Conclusie

Uit het voorgaande blijkt dat toepassing van positieve of negatieve logica op uitvoeringsschema's het voordeel heeft dat de door de schakeling gerealiseerde formule gemakkelijk uit het symbool is af te lezen. Bij toepassing van individuele logica is dit moeilijker, hetgeen als bezwaar tegen dit systeem kan worden aangevoerd. De praktijk heeft echter geleerd dat dit bezwaar lang niet opweegt tegen het voordeel dat toepassing van individuele logica biedt met betrekking tot de vrijheid in de keuze van signaallnamen en de duidelijke weergave van het fysisch gedrag van de schakeling. Met name bij het onderhoud en het testen van schakelingen weegt dit zwaar. *In deze tekst is daarom, in navolging van de Nederlandse normbladen, op uitvoeringsschema's bij voorkeur individuele logica toegepast.* □

A.10 Dualiteit en positieve/negatieve logica

In de Boole algebra kent men het begrip '*duale vorm*' van een formule. De duale vorm van een formule ontstaat uit de formule door de volgende bewerkingen:

- vervang + door $\cdot$ en $\cdot$ door +
- vervang 1 door 0 en 0 door 1
- handhaaf de volgorde van de oorspronkelijke bewerkingen

Voorbeeld

$$f(w, x, y, z) = w \bar{x} y z + \bar{w} x + \bar{y} \bar{z} \quad (\text{A.1})$$

$$D[f(w, x, y, z)] = (w + \bar{x} + y + z)(\bar{w} + x)(\bar{y} + \bar{z}) \quad (\text{A.2})$$

Om de oorspronkelijke volgorde van de bewerkingen te kunnen handhaven zijn haakjes toegevoegd. $\square$

In de *Boole algebra* wordt bewezen dat indien een bepaalde gelijkheid juist is,

$$f_1(x_0, x_1, \dots, x_n) = f_2(x_0, x_1, \dots, x_n) \quad (\text{A.3})$$

de duale vorm ervan eveneens juist is,

$$D[f_1(x_0, x_1, \dots, x_n)] = D[f_2(x_0, x_1, \dots, x_n)] \quad (\text{A.4})$$

Het '*recept*' voor het bepalen van de duale vorm van een formule mag ook op andere dan zuivere som-van-produktenvormen worden toegepast, mits daarbij weer de *oorspronkelijke volgorde van de bewerkingen* gehandhaafd blijft.

Voorbeeld

$$f(x_0, x_1, x_2, x_3, x_4) = ((x_0 + x_1) \cdot x_2 + x_3) \cdot x_4$$

Dan is

$$D[f(x_0, x_1, x_2, x_3, x_4)] = (x_0 \cdot x_1 + x_2) \cdot x_3 + x_4$$

en niet

$$D[f(x_0, x_1, x_2, x_3, x_4)] \neq x_0 \cdot x_1 + x_2 \cdot x_3 + x_4 \quad \square$$

Toepassing op schakelwetten

Alle schakelwetten in hoofdstuk 2 zijn *in paren* gegeven, welke bij nadere beschouwing elkaars duale vorm blijken te zijn. Op grond van de dualiteit is het dan voldoende om een van de twee wetten te bewijzen. De duale vorm van de wet is dan, op grond van de dualiteit, ook bewezen.

Toepassing op positieve/negatieve logica

Dualiteit kan ook toegepast worden bij het interpreteren van gegeven schakelingen volgens positieve of negatieve logica. Hierop is de volgende stelling van toepassing.

Stelling

$$D[f(x_1, x_2, \dots, x_n)] = f(\overline{x_1}, \overline{x_2}, \dots, \overline{x_n})$$

waarbij $f(\overline{x_1}, \overline{x_2}, \dots, \overline{x_n})$ uit $f(x_1, x_2, \dots, x_n)$ ontstaat door x_i in $f(x_1, x_2, \dots, x_n)$ te vervangen door $\overline{x_i}$ en $\overline{x_i}$ door x_i , voor $0 \leq i \leq n$.

Bewijs

We bewijzen de stelling voor functies in twee variabelen. Voor meer variabelen verloopt het bewijs overeenkomstig.

De mintermvorm van een functie in twee variabelen is

$$f(x_0, x_1) = \overline{x_1}\overline{x_0} \cdot f_0 + \overline{x_1}x_0 \cdot f_1 + x_1\overline{x_0} \cdot f_2 + x_1x_0 \cdot f_3$$

Hierin zijn de functiewaarden f_0 t/m f_3 gelijk aan 0 of 1, afhankelijk van de functie. De duale vorm van de functie $f(x_0, x_1)$ is dan

$$D[f(x_0, x_1)] = (\overline{x_1} + \overline{x_0} + \overline{f_0})(\overline{x_1} + x_0 + \overline{f_1})(x_1 + \overline{x_0} + \overline{f_2})(x_1 + x_0 + \overline{f_3})$$

Onder toepassing van de regels

$$(x + y)(x + z) = x + yz$$

en

$$(\overline{x} + y)(x + z) = \overline{x}z + xy$$

volgt hieruit

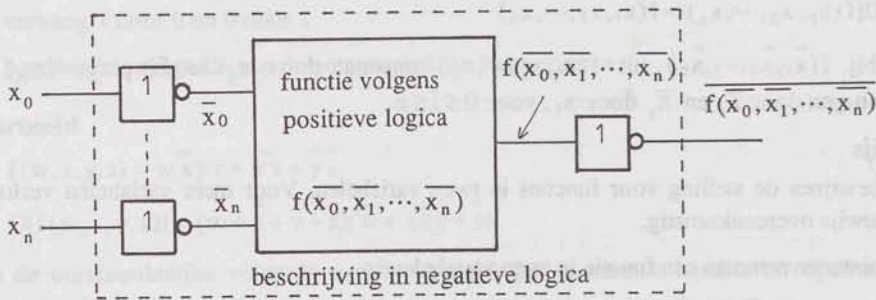
$$\begin{aligned} D[f(x_0, x_1)] &= (\overline{x_1} + (\overline{x_0} + \overline{f_0})(x_0 + \overline{f_1}))\{x_1 + (\overline{x_0} + \overline{f_2})(x_0 + \overline{f_3})\} \\ &= \overline{x_1}(\overline{x_0} + \overline{f_2})(x_0 + \overline{f_3}) + x_1(\overline{x_0} + \overline{f_0})(x_0 + \overline{f_1}) \\ &= \overline{x_1}(\overline{x_0}\overline{f_3} + x_0\overline{f_2}) + x_1(\overline{x_0}\overline{f_1} + x_0\overline{f_0}) \\ &= \overline{x_1}\overline{x_0}\overline{f_3} + \overline{x_1}x_0\overline{f_2} + x_1\overline{x_0}\overline{f_1} + x_1x_0\overline{f_0} \\ &= x_1x_0\overline{f_0} + x_1\overline{x_0}\overline{f_1} + \overline{x_1}x_0\overline{f_2} + \overline{x_1}\overline{x_0}\overline{f_3} \\ &= \overline{\overline{x_1}\overline{x_0}\overline{f_0}} + \overline{\overline{x_1}\overline{x_0}\overline{f_1}} + \overline{\overline{x_1}\overline{x_0}\overline{f_2}} + \overline{\overline{x_1}\overline{x_0}\overline{f_3}} \end{aligned}$$

Deze laatste vorm is de mintermvorm van $\overline{f(\overline{x_0}, \overline{x_1})}$. Hiermee is de stelling voor functies in twee variabelen bewezen. $\square$

Toepassing

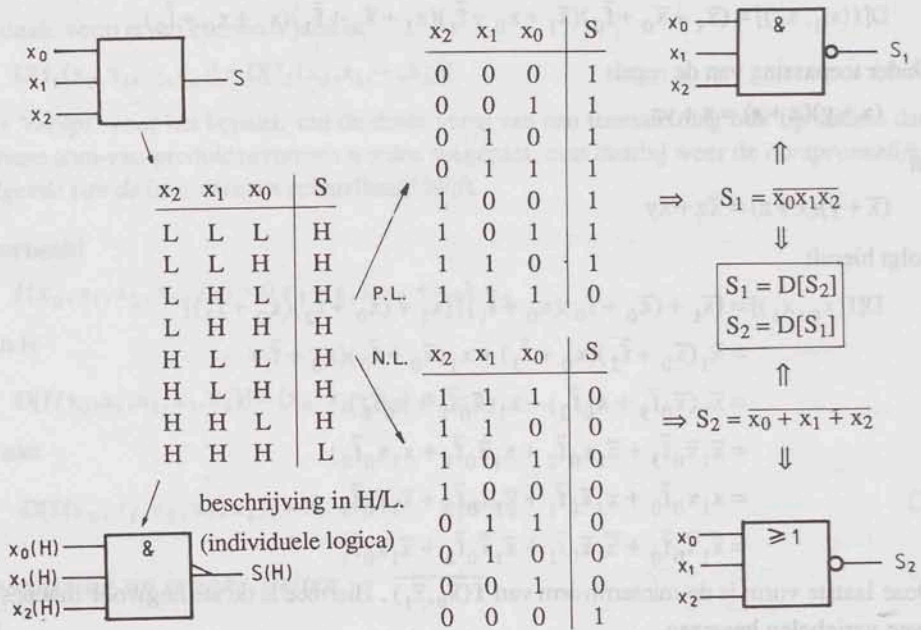
Een gegeven componentbeschrijving, uitgedrukt in H en L, is door middel van positieve logica omgezet in een symbool. Binnen dit symbool, we merken dit nogmaals op, is de specificatie van de functie 'logisch'. Bij dit symbool behoort in positieve logica een

bepaalde logische formule, welke de externe logische werking beschrijft. We hebben reeds eerder gezien dat de logische formule/functie van een gegeven component afhangt van de wijze van interpreteren, volgens positieve, negatieve of individuele logica (paragraaf 4.8). *Op grond van de bovenstaande stelling mag nu geconcludeerd worden dat de formules, af te leiden volgens positieve en negatieve logica, elkaars duale vorm zijn.* $\square$



Figuur A.19. Conversie van positieve naar negatieve logica.

Voorbeeld



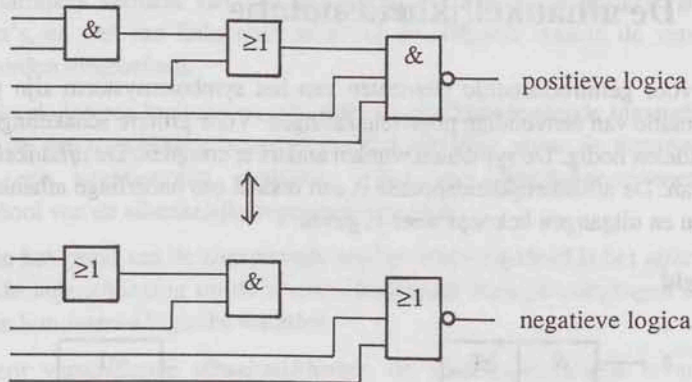
Figuur A.20. Interpretatie volgens positieve en negatieve logica. $\square$

Toelichting

Wanneer in positieve en negatieve logica aan een ingang of uitgang een 'naam' (of variabele) wordt toegekend, dan zijn de met deze namen corresponderende variabelen

elkaars complement. Immers, een niveau H correspondeert in het ene systeem met 'waar' en in het andere met 'niet waar'. Aan de ingangen en uitgangen kunnen we dus van het ene systeem naar het andere overgaan door toepassing van een negator aan elke ingang en uitgang. Zie figuur A.19. □

Voorbeeld



Figuur A.21. Conversie schema's in positieve en negatieve logica. □

Opmerking

Elke 'poort' mag vervangen worden door de duale 'poort' in het andere systeem. De volgorde van de oorspronkelijk bewerkingen blijft daarbij gehandhaafd. □

In schema's volgens positieve of negatieve logica is men bezig op logisch niveau, men interpreteert de schakeling of component volgens een bepaalde afspraak. Binnen het IEC symbolensysteem is afgesproken op schema's met *positieve/negatieve logica uitsluitend negatie-indicatoren* toe te passen. Bij *individuele logica* daarentegen is de *polariteits-indicator voorgeschreven*. Dergelijke schema's zijn meer bedoeld voor onderhoud en leggen het fysisch gedrag vast.

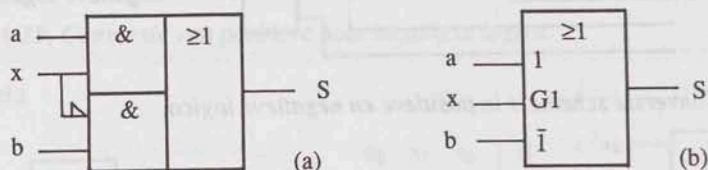
Symbolen voor logische schakelingen II

De afhankelijkheidsnotatie

A.11 De afhankelijkheidsnotatie

De hiervoor geïntroduceerde elementen van het symbolensysteem zijn geschikt voor de documentatie van eenvoudige poortschakelingen. Voor grotere schakelingen zijn krachtiger hulpmiddelen nodig. De symbolen worden anders te complex. De *afhankelijkheidsnotatie* is er een van. De afhankelijkheidsnotatie is een middel om onderlinge afhankelijkheden tussen ingangen en uitgangen beknopt weer te geven.

Voorbeeld



Figuur A.22. *G-afhankelijkheid.*

In figuur A.22 is als voorbeeld de *G-afhankelijkheid* toegepast op het poortschema van een $2 \rightarrow 1$ selector. De *G-afhankelijkheid* duidt aan dat er een *EN-relatie* bestaat tussen de in het symbool met G aangeduide ingang en de erdoor beïnvloede ingangen met *hetzelfde identificatienummer* als achter G staat.

Bestaat er een *EN-relatie* met *het complement* van de met *G aangeduide ingang* (of uitwendig met het inverse signaalniveau) dan wordt dit aangeduid met een negatiestreep boven het identificatienummer. Een en ander is in figuur A.22.b aangegeven. □

Afhankelijkheidsnotatie

Het systeem van afhankelijkheidsnotatie kent twee soorten ingangen/uitgangen, *beïnvloedende ingangen/uitgangen* en *afhankelijke ingangen/uitgangen* (affecting inputs/outputs respectievelijk affected inputs/outputs). Voor de afhankelijkheidsnotatie geldt een aantal regels, waaronder:

- Een *beïnvloedende ingang/uitgang* draagt een *toevoegsymbool* bestaande uit een of meer hoofdletters, gevolgd door een *identificatienummer* (identifying number) genoteerd in het tientallig stelsel.
- Een *afhankelijke ingang/uitgang* draagt als toevoegsymbool het *identificatienummer* van de beïnvloedende ingang/uitgang waarvan hij afhankelijk is.

- De *letter* of *lettercombinatie* in het *toevoegsymbool* van de beïnvloedende ingang/uitgang bepaalt welke invloed deze ingang/uitgang op de ervan afhankelijke ingangen/uitgangen uitoefent.
- Als een ingang/uitgang *afhankelijk is van meer dan één beïnvloedende ingang/uitgang*, dan worden in het *toevoegsymbool* van de afhankelijke ingang/uitgang de identificatienummers vermeld van *alle* beïnvloedende ingangen/uitgangen, gescheiden door komma's, en wel van links naar rechts in de *volgorde* waarin de verschillende invloeden worden uitgeoefend.
- Indien het niet de interne logische waarde zelf van een beïnvloedende ingang/uitgang is die de door de lettercombinatie bepaalde invloed uitoefent, maar de *negatie* daarvan, dan wordt een negatiestreep geplaatst boven het identificatienummer in het *toevoegsymbool* van de afhankelijke ingangen/uitgangen.
- Behoudens in het geval van de zogenaamde enable-afhankelijkheid is het *effect* van een beïnvloedende ingang/uitgang op de ervan afhankelijke ingangen/uitgangen *uitgedrukt* in termen van hun *interne* logische waarden.

Het systeem kent verschillende afhankelijkheden, de letter(combinatie)s ervan zijn de volgende.

A	adresaafhankelijkheid
C	commando-afhankelijkheid
E	enable-afhankelijkheid
G	EN-afhankelijkheid
M	mode-afhankelijkheid
N	negatie-afhankelijkheid
R	resetaafhankelijkheid
S	setafhankelijkheid
V	OF-afhankelijkheid
Z	hulpmiddel voor het aangeven van interne verbindingen

Enkele ervan zullen we hierna kort behandelen. Binnen een symbool kunnen verschillende types afhankelijkheden voorkomen. Wanneer er meer dan een afhankelijkheid voorkomt, dan gelden aanvullende regels.

- Twee beïnvloedende ingangen/uitgangen met verschillende letter(combinatie)s hebben altijd verschillende identificatienummers.
- Beïnvloedende ingangen/uitgangen met *dezelfde letter(combinatie)* en hetzelfde *identificatienummer* staan in een OF-afhankelijkheid ten opzichte van elkaar.
- *Identificatienummers* van beïnvloedende ingangen/uitgangen behoeven *niet opeenvolgend* gekozen te worden. Zij dienen echter zo gekozen te worden dat *geen verwarring* kan ontstaan met andere in *hetzelfde* symbool voorkomende getallen zoals gewichten bij ingangen en uitgangen van code-omzetters.

Met deze regels kan de onderlinge hiërarchie tussen verschillende afhankelijkheden bepaald worden.

Tot slot nog een regel die enige toelichting behoeft.

- Een beïnvloedende ingang/uitgang heeft uitsluitend een effect op de ervan afhankelijke ingangen/uitgangen. Het is geen 'normale' ingang van het element waaraan hij is getekend.

In figuur A.22.b heeft de G1-ingang geen invloed op de OF-functie, zoals die in het symbool is voorgesteld, althans niet direct. G1 doet dit uitsluitend bij aanwezigheid van signaal a of van signaal b.

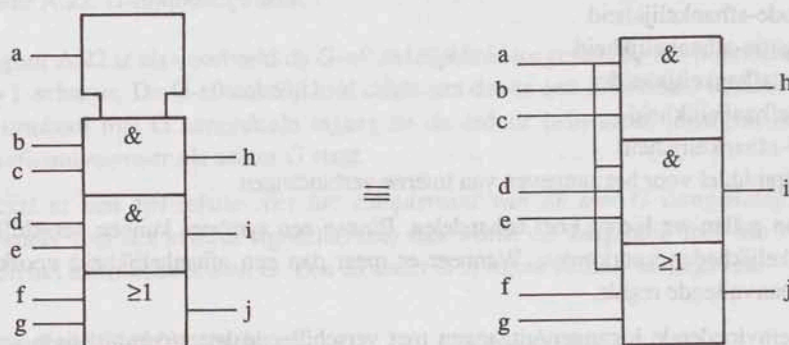
Opmerking

De afhankelijkheidsnotatie wordt bij voorkeur niet gebruikt bij kleine schakelingen, op het poortniveau bijvoorbeeld. □

A.12 Gemeenschappelijke ingangen en uitgangen

Het gemeenschappelijk ingangsblok

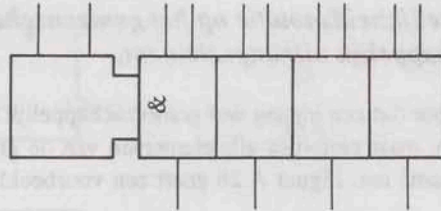
Indien een groep (array) van aan elkaar getekende, doch niet noodzakelijk gelijke, elementen een of meer gemeenschappelijke ingangen bezit, dan kunnen deze ingangen getekend worden aan een *gemeenschappelijk ingangsblok*. Dit blok wordt aan één der uiteinden van de groep van elementen getekend.



Figuur A.23. Toepassing van het gemeenschappelijk ingangsblok.

Ingang a in figuur A.23 is een gemeenschappelijke ingang van alle elementen van de groep. Vooral als er meer gemeenschappelijke ingangen zijn biedt toepassing van het gemeenschappelijk ingangsblok voordelen, omdat een groot aantal lijnkruisingen hierdoor kan worden vermeden. Figuur A.24 toont de toepassing van een gemeenschappelijk ingangsblok met twee ingangen op een horizontaal georiënteerde groep van elementen. Vaak zijn gemeenschappelijke ingangen afkomstig van de *besturing* (control) van de schakeling. Het gemeenschappelijk ingangsblok heet daarom ook wel *gemeenschappelijk besturingsblok* (common control block). Door consequent alle besturingssignalen hierop aan te sluiten kan

een overzichtelijke tekening ontstaan, met een duidelijke scheiding tussen datapad en besturing.



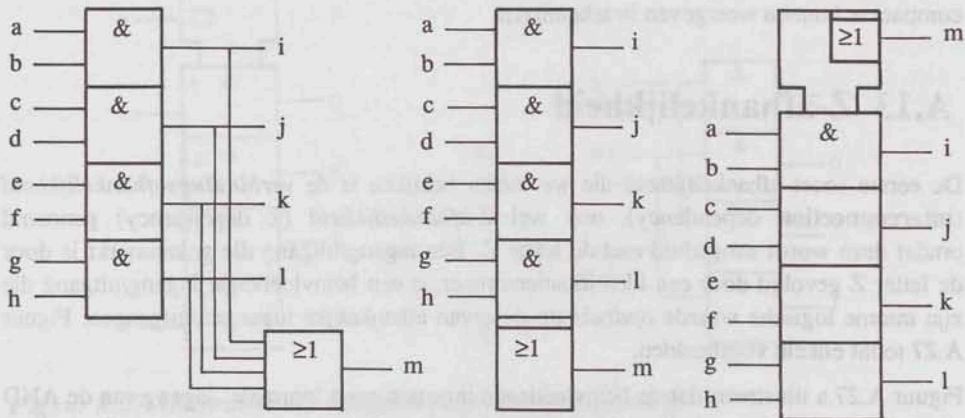
Figuur A.24. Toepassing van het gemeenschappelijk ingangsblok.

Opmerking

Wanneer een gemeenschappelijk ingangsblok wordt toegepast bij een groep van elementen waarin een bepaalde rangorde schuilt, gewichten van uitgangen bijvoorbeeld, dan wordt het element met de uitgang met de laagste rangorde het dichtst op het ingangsblok getekend. □

Het gemeenschappelijk uitgangselement

Vaak bezitten schakelingen die opgebouwd zijn uit een groep van elementen behalve de uitgangen van de elementen zelf ook uitgangen die een functie zijn van de uitgangen van *alle* elementen. In dat geval kan men gebruik maken van een of meer *gemeenschappelijke uitgangselementen* (common output elements). Dit element dat bestaat uit een rechthoek waarvan een zijde dubbel getekend is, wordt aan één van de uiteinden van de groep van symbolen getekend, daarvan gescheiden door genoemde dubbele streep, of in het gemeenschappelijk ingangsblok.



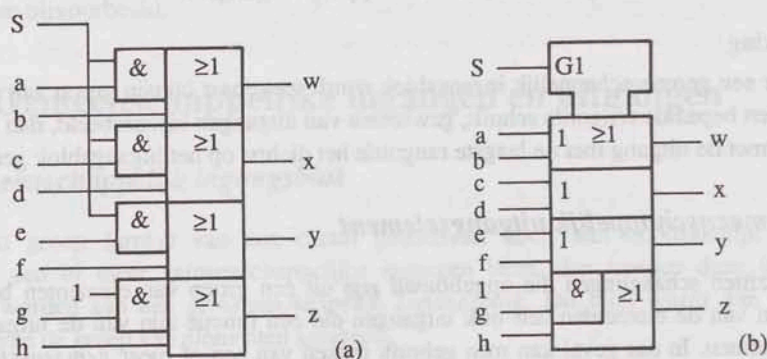
Figuur A.25. Toepassing van het gemeenschappelijk uitgangselement.

Het gemeenschappelijk uitgangselement bespaart veel kruisende lijnen in tekeningen. Bedenken we daarnaast dat een gemeenschappelijk uitgangselement ook externe ingangen

mag bezitten, dan is het een middel dat de overzichtelijkheid van een tekening sterk kan bevorderen.

Toepassing van de afhankelijkheidsnotatie op het gemeenschappelijk ingangsblok of gemeenschappelijk uitgangselement.

Vaak doet zich het probleem voor dat een ingang wel gemeenschappelijk is voor een groot aantal elementen van een groep, maar niet voor alle elementen van de groep. In dat geval past men de afhankelijkheidsnotatie toe. Figuur A.26 geeft een voorbeeld voor G-afhanke-lijkheid.



Figuur A.26. Afhankelijkheidsnotatie in gemeenschappelijk ingangsblok.

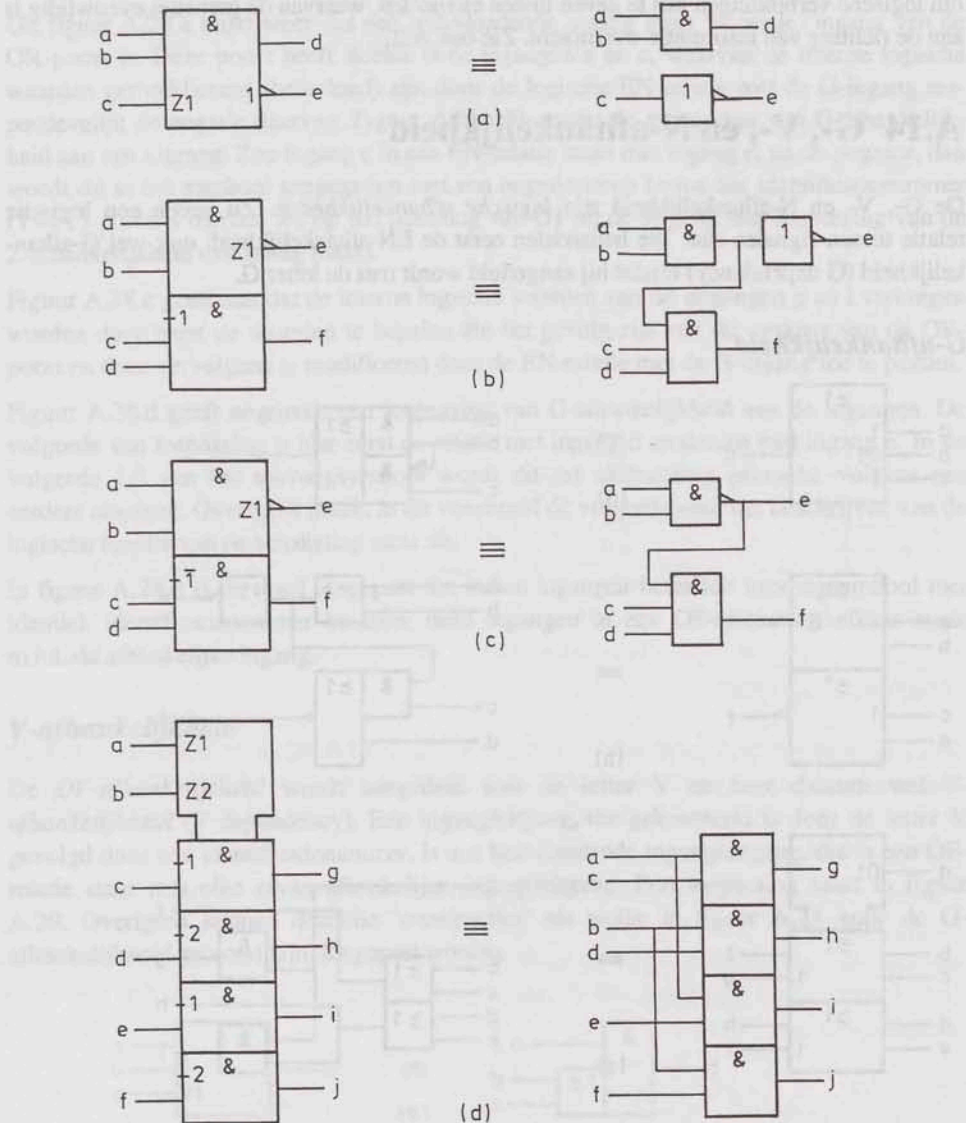
Afhankelijkheidsnotatie, in combinatie met het gemeenschappelijk ingangsblok en het gemeenschappelijk uitgangselement, vormt een machtig hulpmiddel om schakelingen compact te kunnen weergeven in tekeningen.

A.13 Z-afhankelijkheid

De eerste soort afhankelijkheid die we zullen bekijken is de *verbindingsafhankelijkheid* (*interconnection dependency*), ook wel *Z-afhankelijkheid* (*Z dependency*) genoemd omdat deze wordt aangeduid met de letter Z. Een ingang/uitgang die gekenmerkt is door de letter Z gevolgd door een identificatienummer, is een beïnvloedende ingang/uitgang die zijn interne logische waarde *opdrukt* op de ervan afhankelijke ingangen/uitgangen. Figuur A.27 toont enkele voorbeelden.

Figuur A.27.a illustreert dat de beïnvloedende ingang c geen 'normale' ingang van de AND poort is, maar uitsluitend zijn inwendige logische waarde oplegt aan de ervan afhankelijke uitgang e. Daarmee is uitgang e tevens onafhankelijk van de ingangen a en b; c is onafhankelijk van de werking van de AND poort. In de figuren A.27.b t/m A.27.d is gebruik gemaakt van een kort streepje om een zogenaamde *inwendige ingang* aan te duiden. Hoewel een dergelijke inwendige ingang volgens de IEC-afspraken geacht wordt

steeds de interne logische waarde 1 te bezitten, wordt deze 1 hier overheerst door de invloed van de Z-afhankelijkheid.



Figuur A.27. Illustraties van het gebruik van Z-afhankelijkheid.

De figuren A.27.b en A.27.c tonen duidelijk dat het de *interne logische waarde* van de beïnvloedende Z-uitgang is die op de afhankelijke ingang wordt overgedragen.

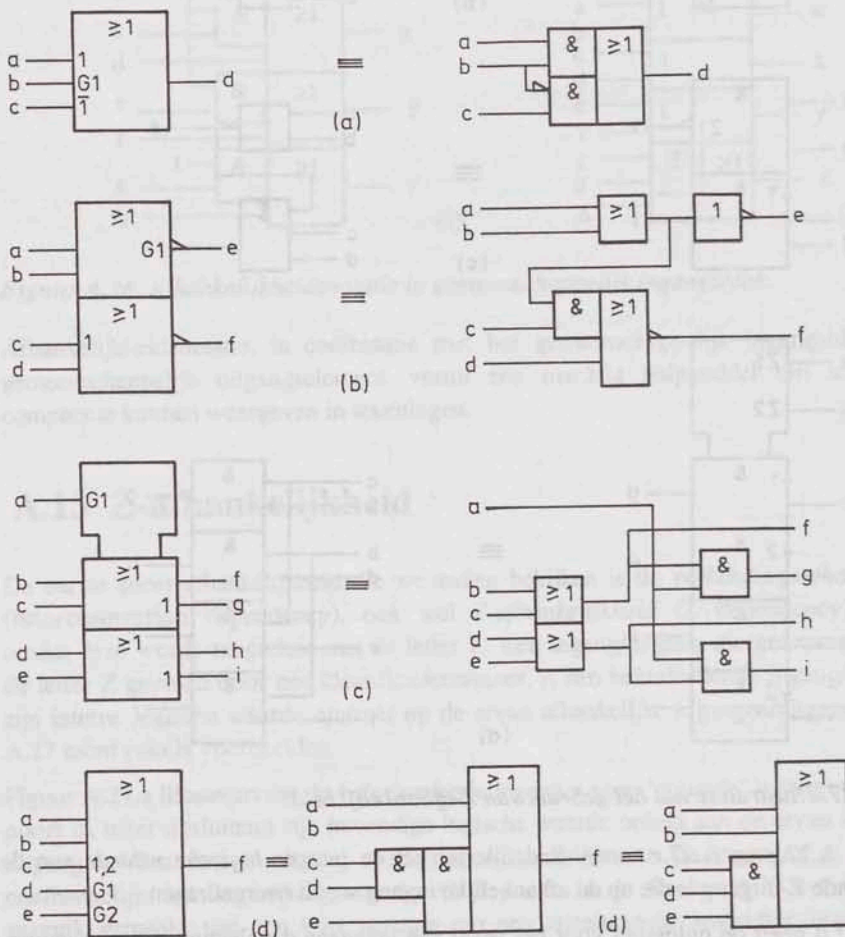
Figuur A.27.d geeft de oplossing voor het geval van ingangen die slechts gemeenschappelijk zijn voor een *aantal* elementen van een groep en niet voor alle.

Met de Z-afhankelijkheid kunnen verbindingen in een symbool worden aangegeven die anders extern zouden moeten worden aangegeven, of die aanleiding geven tot 'embedded symbols'. Tevens tonen bovenstaande figuren dat de Z-afhankelijkheid het mogelijk maakt om logische verbindingen aan te geven tussen elementen, waarvan de grenslijn evenwijdig is aan de richting van informatie-overdracht. Zie ook A.2.

A.14 G-, V-, en N-afhankelijkheid

De G-, V- en N-afhankelijkheid zijn *logische afhankelijkheden*. Zij geven een logische relatie tussen signalen aan. We behandelen eerst de EN-afhankelijkheid, ook wel G-afhankelijkheid (G dependency) omdat hij aangeduid wordt met de letter G.

G-afhankelijkheid



Figuur A.28. Illustraties van het gebruik van G-afhankelijkheid.

Een ingang/uitgang, die gekenmerkt is door de letter G gevolgd door een identificatienummer, is een beïnvloedende ingang/uitgang, die in een *logische EN-relatie* staat met elke ervan afhankelijke ingang/uitgang. Figuur A.28 toont enkele voorbeelden.

Uit figuur A.28.a blijkt weer dat een beïnvloedende ingang geen 'normale' ingang van de OR-poort is. Deze poort heeft slechts twee ingangen a en c, waarvan de interne logische waarden gemodificeerd (beïnvloed) zijn door de logische EN-relatie met de G-ingang respectievelijk de negatie daarvan. Figuur A.28.b illustreert de toepassing van G-afhankelijkheid aan een uitgang. Zou ingang c in een EN-relatie staan met ingang e, na de negator, dan wordt dit in het symbool aangegeven met een negatiestreep boven het identificatienummer ($1 \rightarrow \bar{1}$). Tevens merken we op dat plaatsing van G1 bij de uitgang hier toepassing van de Z-afhankelijkheid overbodig maakt.

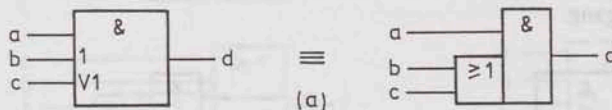
Figuur A.28.c geeft aan dat de interne logische waarden van de uitgangen g en i verkregen worden door eerst de waarden te bepalen die het gevolg zijn van de werking van de OR-poort en deze vervolgens te modificeren door de EN-relatie met de G-ingang toe te passen.

Figuur A.28.d geeft nogmaals een toepassing van G-afhankelijkheid aan de ingangen. De volgorde van toepassing is hier eerst de relatie met ingang d en daarna met ingang e. In de volgorde 1,2 van het toevoegsymbool wordt dit tot uitdrukking gebracht, volgens een eerdere afspraak. Overigens maakt in dit voorbeeld de volgorde voor het beschrijven van de logische functie van de schakeling niets uit.

In figuur A.28.d is de regel toegepast dat indien ingangen hetzelfde toevoegsymbool met identiek identificatienummer bezitten, deze ingangen in een OF-relatie tot elkaar staan m.b.t. de afhankelijke ingang.

V-afhankelijkheid

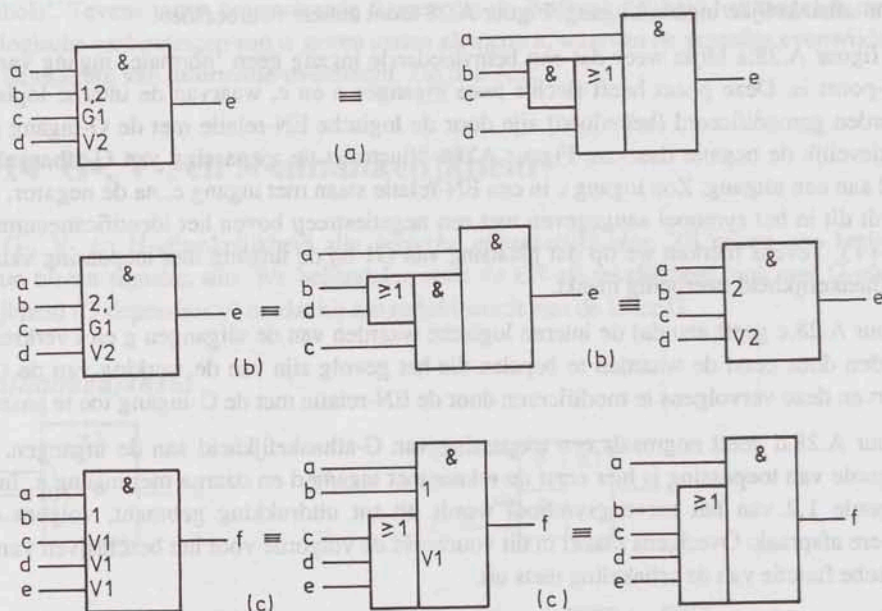
De *OF-afhankelijkheid* wordt aangeduid met de letter V en heet daarom wel *V-afhankelijkheid* (V dependency). Een ingang/uitgang, die gekenmerkt is door de letter V gevolgd door een identificatienummer, is een beïnvloedende ingang/uitgang, die in een OF-relatie staat met elke ervan afhankelijke ingang/uitgang. Een toepassing staat in figuur A.29. Overigens kunnen dezelfde 'constructies' als welke in figuur A.28 voor de G-afhankelijkheid getoond zijn, toegepast worden.



Figuur A.29. Toepassing V-afhankelijkheid.

Is een ingang/uitgang afhankelijk van een aantal beïnvloedende ingangen/uitgangen van verschillende soort, bijvoorbeeld G- en V-afhankelijkheden, dan dient men goed op de *volgorde van de identificatienummers* te letten. Zie figuur A.30.a en A.30.b. Figuur A.30.c illustreert nog een interessante toepassing van V-afhankelijkheid in combinatie met de regel

die zegt dat beïnvloedende ingangen met hetzelfde identificatienummer in een OF-relatie met elkaar staan.



Figuur A.30. Voorbeelden van meervoudige afhankelijkheid.

N-afhankelijkheid

Het IEC-systeem kent twee manieren om een negatie aan te geven. Allereerst kan een identificatienummer bij een afhankelijke ingang/uitgang voorzien worden van een negatiestreep. De ingang/uitgang is dan afhankelijk van het complement van de beïnvloedende ingang/uitgang.

Daarnaast kent het systeem ook de **negatie-afhankelijkheid** (**negate dependency**), ook wel **N-afhankelijkheid** genoemd omdat hij aangeduid wordt met de letter N. Een ingang/uitgang, die gekenmerkt is door de letter N gevolgd door een identificatienummer, is een beïnvloedende ingang/uitgang, die in een EXCLUSIEVE OF-relatie staat met elke ervan afhankelijke ingang/uitgang.



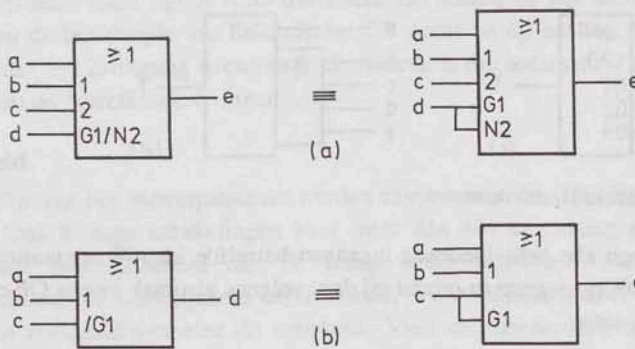
Figuur A.31. Toepassing N-afhankelijkheid.

Uit deze omschrijving volgt dat een N-ingang, die de interne logische waarde 0 heeft, de interne logische waarde van een ervan afhankelijke ingang/uitgang onberoerd laat. Heeft de

beïnvloedende ingang daarentegen de interne waarde 1, dan vormt deze de negatie van de logische waarde van de ervan afhankelijke ingangen en uitgangen. Dit verklaart ook de naam. Zie figuur A.31.

Ingangen/uitgangen met meer dan één functie

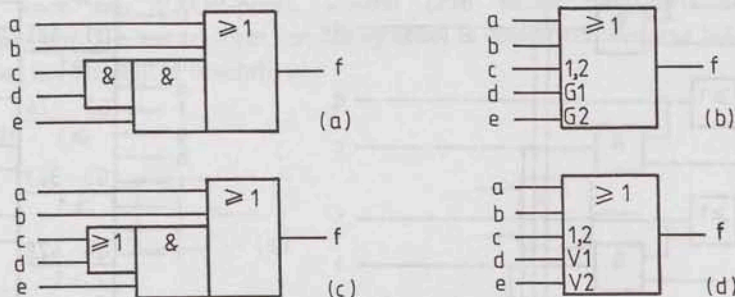
In sommige gevallen heeft een ingang/uitgang *meer dan één functie*. De voor de aanduiding van deze functies noodzakelijke toevoegsymbolen worden dan gescheiden door een *schuine streep* (solidus). Is één van de de functies die van een 'normale' ingang, dan begint het toevoegsymbool met een schuine streep. Zie figuur A.32.



Figuur A.32. Voorbeelden van ingangen met een meervoudige functie.

A.15 Enkele toepassingen

Voorbeeld

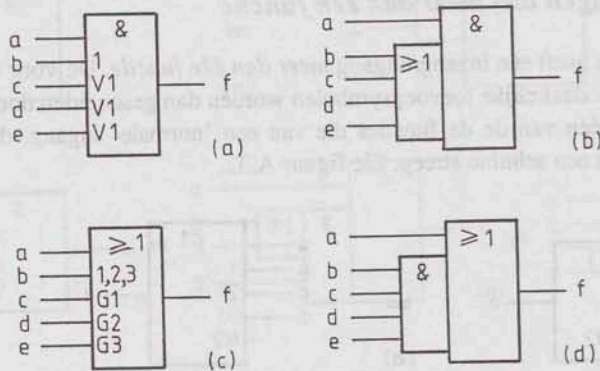


Figuur A.33. Volgorde identificatienummers.

In figuur A.33.b doet de volgorde van de identificatienummers achter ingang c er niet toe. Het normblad schrijft deze volgorde voor, gezien de opbouw van de schakeling.

In figuur A.33.d is de volgorde van de identificatienummers achter ingang c essentieel voor een juiste interpretatie van het symbool. □

Voorbeeld

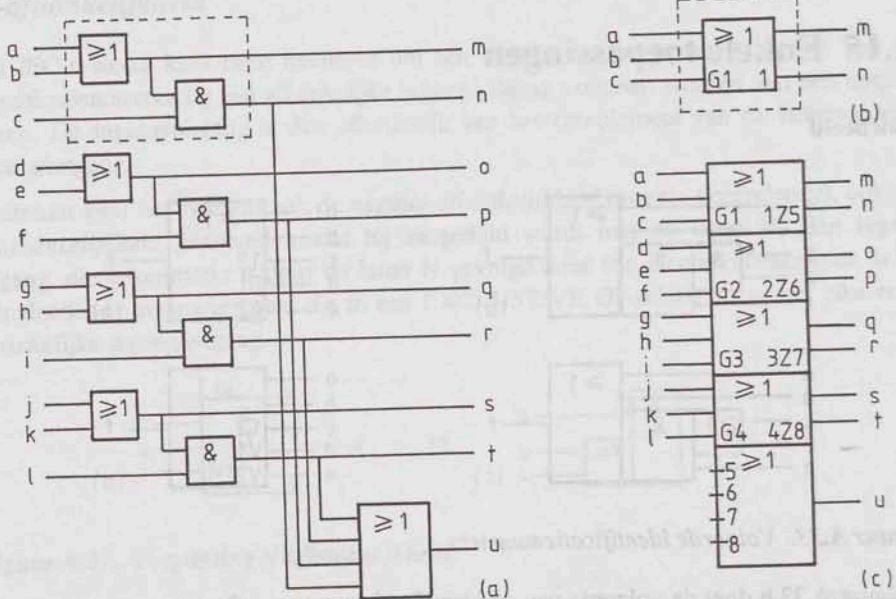


Figuur A.34. Keuze identificatienummers.

In figuur A.34.a hebben alle beïnvloedende ingangen hetzelfde identificatienummer als de afhankelijke ingang. Dit is toegestaan omdat zij dan, volgens afspraak in een OF-relatie tot de afhankelijke ingang staan.

In figuur A.34.c zijn verschillende identificatienummers noodzakelijk. □

Voorbeeld

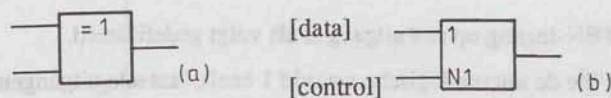


Figuur A.35. Toepassing Z-afhankelijkheid op een gemeenschappelijke uitgang.

Voor het gemeenschappelijk uitgangselement geldt de regel dat van elk element van de groep precies één uitgang als ingang van het gemeenschappelijk uitgangselement fungeert. De interne logische waarde van deze ingang is dan gelijk aan die van de betreffende uitgang. Uit deze regel volgt dat het gebruik van het gemeenschappelijk uitgangselement alleen dan is toegestaan als van *elk* element van de groep *alle* uitgangen steeds dezelfde interne logische waarde hebben. Anders zou er immers onzekerheid bestaan van welke uitgang van een element de logische waarde op de ingang van het gemeenschappelijk uitgangselement wordt overgedragen. Is aan deze eis niet voldaan of leveren niet alle elementen van de groep een bijdrage aan de gemeenschappelijke uitgang, dan mag men het gemeenschappelijk uitgangselement niet toepassen en is men aangewezen op de Z-afhankelijkheid, zoals figuur A.35 illustreert. Let hierbij op het ontbreken van de dubbele streep aan de bovenzijde van de onderste OR-poort en op het feit dat de interne logische waarde, die een Z-uitgang overdraagt, de waarde is die ontstaat na toepassing van de EN-relatie met de betreffende G-ingang. □

Voorbeeld

Afhankelijk van het ontwerp stadium worden aan symbolen in tekeningen verschillende eisen gesteld. Ook kunnen schakelingen voor meer dan één toepassing gebruikt worden. Men wordt dan geconfronteerd met de vraag welke eisen aan het symbool prevaleren, standaardisatie of duidelijkheid bijvoorbeeld. Voor documentatie van realisatieschema's verdienen standaardsymbolen de voorkeur. Voor de meeste IC's zijn standaardsymbolen voorgesteld.



Figuur A.36. Toepassingen van EXOR-poort.

Figuur A.36.a beschrijft het symbool van een EXOR-poort. Figuur A.36.b geeft een symbool voor een EXOR-poort, waarbij deze wordt toegepast als instelbare invertor/negator. De laatste vorm van het symbool is duidelijker wanneer het gaat om de functie van het element te beschrijven. □

Voorbeeld



Figuur A.37. Symbolen voor een poort van het circuit SN7400.

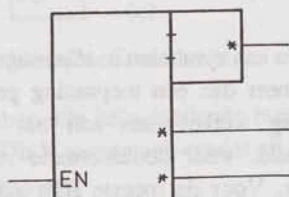
Figuur A.37 geeft twee manieren van voorstellen van het symbool voor een '2- input NAND'. Symbool A.37.a stelt de poort voor als 'AND-poort met geïnverteerde uitgang'. Symbool A.37.b beschrijft de poort als OR-poort met geïnverteerde ingangen'. Beide symbolen leiden tot dezelfde H/L tabel. □

A.16 EN-afhankelijkheid

De EN-ingang

Vele circuits hebben een *enable-ingang*, aangeduid met EN. Een EN-ingang stelt de uitgangen in op hun normale werking, met uitgangsniveau H of L, of stuurt ze naar bijvoorbeeld een toestand van zeer hoge impedantie, zoals bij 3-state uitgangen. Op de EN-ingang zijn de volgende afspraken van toepassing.

Een enable-ingang, aangeduid door het toevoegsymbool EN, heeft uitsluitend invloed op de uitgangen van het element en wel op *alle* uitgangen. Bij een samengesteld element zijn dit alle *naar buiten gevoerde* uitgangen en niet de eventuele inwendige uitgangen van de rechthoek die naar interne ingangen gaan. In figuur A.38 bijvoorbeeld staan slechts de met * gemerkte uitgangen onder besturing van de EN-ingang.



Figuur A.38. Uitgangen, beïnvloed door een EN-ingang.

Het effect van een EN-ingang op een uitgang is als volgt gedefinieerd.

- Een EN-ingang die de interne logische waarde 1 heeft, laat alle uitgangen *onberoerd*. Zij bezitten dan hun normaal gedefinieerde logische waarde en hebben het normale effect op de erdoor gestuurde elementen.
- Een EN-ingang die de interne logische waarde 0 heeft, oefent de volgende uitwerking op de erdoor beïnvloede uitgangen uit.

Normale uitgangen

Een *normale uitgang* krijgt de interne logische waarde 0. Het effect van een EN-ingang op een normale uitgang is gelijk aan dat van een G-ingang.

Open-circuit uitgangen

Een *open-circuit uitgang* wordt naar een toestand van zeer hoge impedantie gestuurd. Een open-circuit uitgang krijgt de interne logische waarde, die overeenkomt met het externe niveau met de zeer hoge impedantie.

Nemen we als voorbeeld een active low/passive pull-up uitgang. Wanneer de impedantie hoog is, correspondeert de uitgangsspanning met het hoge externe niveau. Bij afwezigheid van een polariteitsindicator leidt dit tot een interne logische waarde 1, met polariteits-

indicator tot de interne logische waarde 0. Voor een active high/passive pull-down gelden overeenkomstige redeneringen.

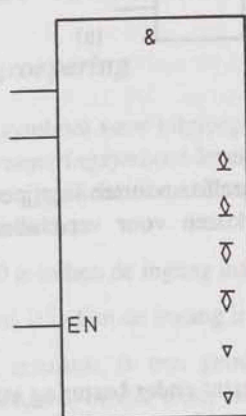
3-State uitgangen

Een 3-state uitgang wordt naar een externe toestand van zeer hoge impedantie gestuurd. Intern behoudt een 3-state uitgang zijn normale interne logische waarde, deze waarde wordt *niet* beïnvloed door de EN-ingang.

Opmerking

Open-circuit uitgangen en 3-state uitgangen verschillen dus duidelijk m.b.t. de definitie van de interne logische waarde van de uitgang. Bij open-circuit uitgangen bepaalt de uitgangsconstructie bij EN = 0 de interne logische waarde, bij 3-state uitgangen is het de logische functie van het element die de interne logische waarde bepaalt. □

Figuur A.39 licht een en ander toe. Hierbij is aangenomen dat de enable-ingang EN intern 0 is. Alle open-circuit uitgangen zijn *disabled*. Het is dan eigenlijk niet van belang welk extern logisch niveau zij aannemen, omdat dit wegens de toestand van zeer hoge impedantie toch geen effect heeft op de elementen waarmee zij verbonden zijn. Deze niveaus zijn daarom tussen haken geplaatst.

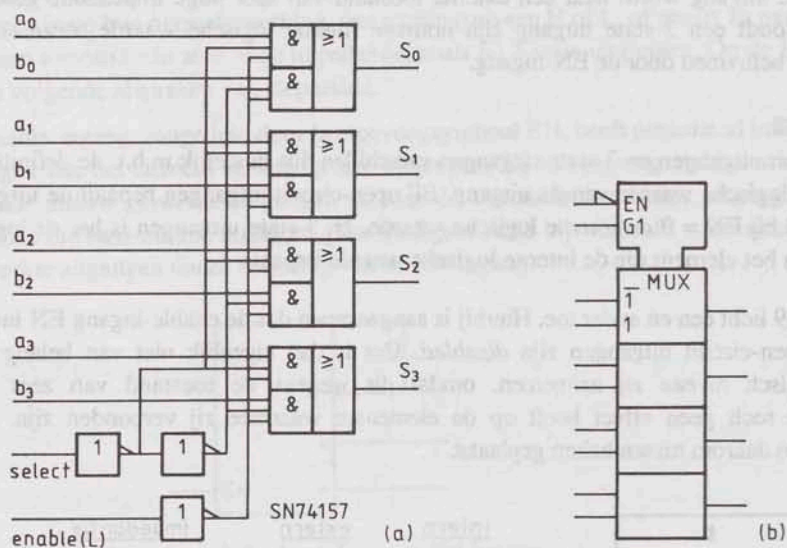
		intern	extern	impedantie
	a	0	L	laag
	b	0	H	laag
	c	1	(H)	hoog
	d	0	(H)	hoog
	e	0	(L)	hoog
	f	1	(L)	hoog
	g	0 of 1	(L of H)	hoog
	h	0 of 1	(H of L)	hoog

Figuur A.39. Invloed van een EN-ingang.

Men kan zich afvragen waarom de interne logische waarden van de uitgangen c t/m h van belang zijn. Hun uitgangsimpedantie is immers te hoog om enig effect te hebben. Men bedenke echter dat deze uitgangen eventueel ook intern als beïnvloedende uitgangen/ingangen kunnen optreden en dan is juist hun interne logische waarde bepalend voor hun invloed!

Voorbeeld

Figuur A.40 toont het uitvoeringsschema van een viervoudige $2 \rightarrow 1$ selector/multiplexer van het type SN74157. In het bijbehorende symbool is gebruik gemaakt van G-afhankelijkheid aan de ingangen en EN-afhankelijkheid aan de uitgangen.

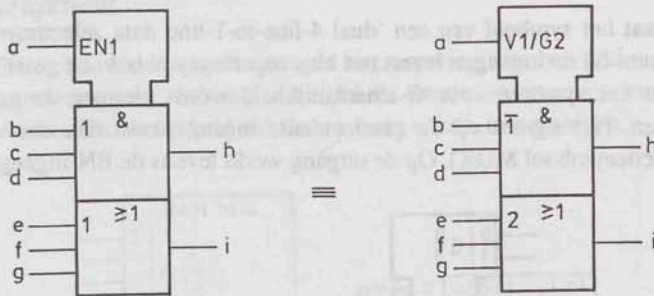


Figuur A.40. Toepassing G/EN-afhankelijkheid.

Hoewel intern beide sturingangen op hetzelfde niveau en op dezelfde poorten ingrijpen is vanwege een *gemakkelijke interpretatie* van het symbool gekozen voor verschillende afhankelijkheidsnotaties.

EN-afhankelijkheid

Uiteraard kan het voorkomen dat niet alle uitgangen van een element onder besturing staan van een EN-ingang en/of dat deze ook van invloed is op een of meer ingangen. In dat geval kan men gebruik maken van ENABLE-afhankelijkheid (EN dependency). Men voegt dan een identificatienummer aan de letters EN toe en vermeldt dit bij alle ingangen en uitgangen (eventueel ook inwendige), die door de betreffende ingang beïnvloed worden. Het effect op uitgangen is dan hetzelfde als van een normale EN-ingang, dat op ingangen wijkt echter af. Een beïnvloedende EN-ingang, die de interne logische waarde 1 heeft, laat weliswaar de ervan afhankelijke ingangen onberoerd, maar de interne logische waarde 0 van een beïnvloedende EN-ingang heeft als uitwerking dat de ervan afhankelijke ingangen als niet bestaand moeten worden beschouwd (ignored). Figuur A.41 illustreert dit.



Figuur A.41. Invloed van een EN-ingang op ingangen.

Als ingang a intern 0 is, hebben de ingangen b en e geen invloed (zij worden als niet bestaand beschouwd) op de AND- resp. OR-poort. Ingang b moet dan intern 1 zijn en ingang e intern 0. Dit verklaart het linker gedeelte van de figuur. Rechts in figuur A.41 is aangegeven hoe een en ander met behulp van V- en G-afhankelijkheid kan worden aangegeven.

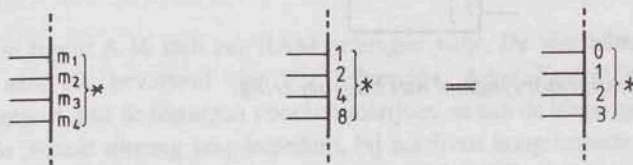
A.17 A-afhankelijkheid

Bitgroepering

Het symbool voor bitgroepering is een accolade. Ingangen die gegroepeerd zijn onder het *bitgroeperingsymbool* leveren samen een *intern getal* (internal number) op, dat de som is van de gewichten van de afzonderlijke ingangen. Elke ingang met bijschrift m_i heeft een gewicht dat

- 0 is indien de ingang intern de logische waarde 0 heeft;
- m_i is indien de ingang intern de logische waarde 1 heeft.

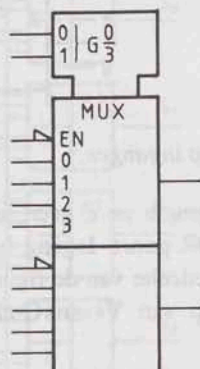
Het resultaat is een getal waarop bijvoorbeeld een mathematische bewerking wordt uitgevoerd, of dat dient als identificatienummer in bijvoorbeeld *adresafhankelijkheid*. Bij het bitgroeperingsymbool moeten de gewichten der ingangen, de m_i 's, in opklimmende of afnemende volgorde genoteerd worden. Zijn alle gewichten binair, dan kan volstaan worden met het noteren van de exponent van het grondtal 2. Anders moet het decimale equivalent van het gewicht vermeld worden. Zie figuur A.42.



Figuur A.42. Bitgroeperingsymbool.

Voorbeeld

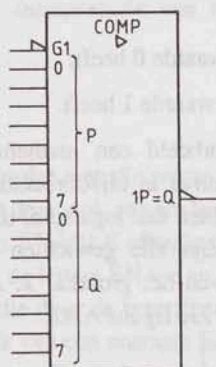
In figuur A.43 staat het symbool van een 'dual 4-line-to-1-line data selector/multiplexer'. Gezien de gewichten bij de ingangen levert het bitgroeperingsymbool een getal tussen 0 en 3 op. Samen met het symbool voor G-afhankelijkheid wordt hiermee de geselecteerde ingang aangewezen. Het signaal op de geselecteerde ingang wordt dan naar de uitgang doorgegeven (functiesymbool MUX). Op de uitgang werkt tevens de EN-ingang.



Figuur A.43. Selectorsymbool met bitgroepering. □

Voorbeeld

Figuur A.44 toont het symbool van een comparator voor twee 8-bit getallen/bitstrings. De bitgroeperingen resulteren in twee interne getallen P en Q, welke vergeleken worden. Wanneer $P = Q$, dan krijgt de uitgang de interne waarde 1 (waarop ook nog G-afhankelijkheid met G1 van toepassing is).



Figuur A.44. Comparatorsymbool met bitgroepering. □

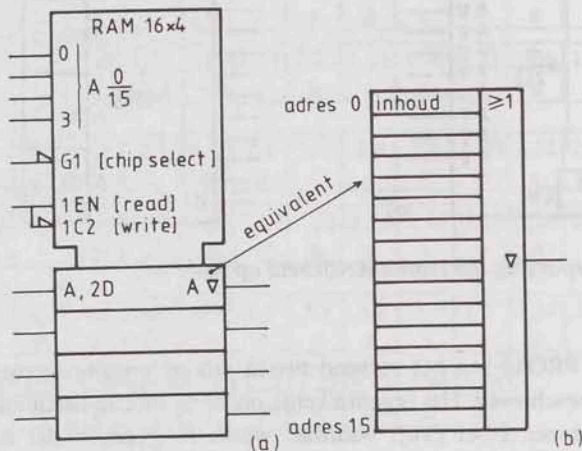
Opmerking

Ook aan uitgangen kan het bitgroeperingsymbool worden toegepast, bijvoorbeeld om de inhoud van een teller weer te geven. □

A-afhankelijkheid

Met adresafhankelijkheid in combinatie met bitgroepering kan men onder andere geheugen-arrays beknopt weergeven. *Adresafhankelijkheid* wordt weergegeven met de letter A. We introduceren adresafhankelijkheid aan de hand van enkele toepassingen.

Voorbeeld



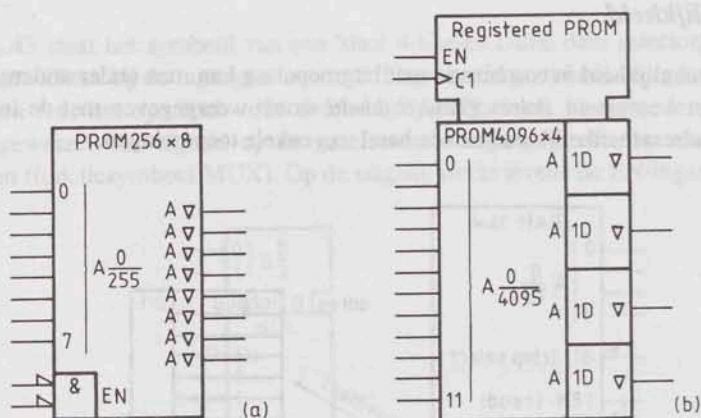
Figuur A.45. Toepassing adresafhankelijkheid op PROM's.

Figuur A.45.a toont het symbool voor een 256×8 PROM circuit. Het interne getal van het bitgroeperingssymbool wordt met adresafhankelijkheid omgezet in een adres, lopend van 0 t/m 255. De inhoud van het aangewezen adres verschijnt dan op de acht uitgangen. Wanneer adresafhankelijkheid wordt toegepast, dan behoeven niet alle 256 adressen getekend te worden. Bij toepassing van G-afhankelijkheid bijvoorbeeld zou dit wel nodig zijn.

In figuur A.45.b is het symbool van een 4096×4 registered PROM getekend. Het aangeboden adres wordt in een intern adres omgezet, waarna de 4-bit inhoud van het aangewezen adres aan de uitgang van de PROM verschijnt. Daar bevindt zich een 4-bit register, met 3-state uitgangen. Op de klok C1 wordt de inhoud van het aangewezen adres in het register geklokt. De enable-ingang EN werkt, volgens een eerdere afspraak, uitsluitend op de externe uitgangen. Wanneer de PROM disabled is, dan is het toch mogelijk het register opnieuw te laden. □

Voorbeeld

Het symbool in figuur A.46 stelt een RAM-geheugen voor. De vier adresingangen wijzen 16 adressen aan, elk bevattend vier bits informatie. Adresafhankelijkheid wordt hier tweemaal toegepast, aan de ingangen voor het schrijven en aan de uitgangen voor het lezen. Bij lezen is de 3-state uitgang laag-impedant, bij schrijven hoog-impedant. (De betekenis van de aanduiding C2 bij de schrijfgang wordt toegelicht bij C-afhankelijkheid; zie deel III van de appendix in deel 2.) □



Figuur A.46. Toepassing adresafhankelijkheid op RAM's.

Opmerking

Bij een ROM of PROM kan het verband tussen inhoud en uitgangswaarden niet in het symbool worden beschreven. Het is gebruikelijk onder de functie-aanduiding een verwijzing op te nemen naar een tabel [Tn], waarmee wordt aangegeven dat tabel n de inhoud specificeert. Alternatieven zijn

[PROM Tn] of [PAL/PLA Tn].

□

Naast de hier genoemde afhankelijkheden bestaat er ook de *C-afhankelijkheid*. Deze wordt gebruikt om de geheugenwerking te beschrijven van latches, flip-flops en andere schakelingen met geheugenwerking zoals registers en tellers. Deze afhankelijkheid bespreken we daarom in deel 1 van dit boek.

Menigeen zal na het voorafgaande verzuchten dat het hiervoor geïntroduceerde symbolen-systeem wel erg ingewikkeld is. Dit is inderdaad het geval. De lezer dient echter te bedenken dat de oorzaak hiervan niet gelegen is in de 'gramatica' van het symbolen-systeem, maar zijn oorzaak vindt in de grote diversiteit van de manieren waarop logische componenten kunnen worden gerealiseerd. Een eenvoudiger systeem, dat minder aspecten meeneemt, is automatisch ook minder relevant. Met name de hiervoor reeds genoemde command of C-afhankelijkheid is uiterst nuttig zo niet noodzakelijk voor het weergeven van de timing eigenschappen van latches en flip-flops. En deze eigenschappen hangen weer samen met ervan afgeleide eigenschappen zoals 'robuustheid' van schakelingen. Een uiterst relevant onderwerp. De opstellers van dit systeem hebben derhalve gemeend zodanige symbolen te moeten opstellen, waarmee zoveel mogelijk relevante eigenschappen gedocumenteerd kunnen worden.

Antwoorden van de opgaven

Inleiding

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
a	d	b	c	b	b	d	b

Hoofdstuk 1

1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	1.10	1.11	1.12	1.13	1.14	1.15
d	c	a	a	b	b	c	d	a	b	c	a	a	d	b

1.16	1.17	1.18	1.19	1.20	1.21	1.22	1.23	1.24	1.25	1.26	1.27	1.28	1.29	1.30
d	c	a	b	a	c	b	c	a	c	d	a	d	c	b

Hoofdstuk 2

2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	2.10	2.11	2.12	2.13	2.14	2.15
a	d	b	b	c	d	c	a	d	b	c	c	d	b	a

2.16	2.17	2.18	2.19	2.20	2.21	2.22	2.23	2.24	2.25	2.26	2.27	2.28	2.29	2.30
a	a	b	b	d	d	b	c	a	c	d	a	d	d	b

Hoofdstuk 3

3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	3.10	3.11	3.12	3.13	3.14	3.15
c	b	b	b	a	d	c	c	a	a	b	d	c	b	d

3.16	3.17	3.18	3.19	3.20	3.21	3.22	3.23	3.24	3.25
c	d	b	c	a	d	d	b	b	a

Hoofdstuk 4

4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9	4.10	4.11	4.12	4.13	4.14	4.15
b	a	a	d	b	b	d	d	d	a/b	b	d	b	b	a

4.16	4.17	4.18	4.19	4.20	4.21	4.22	4.23	4.24	4.25	4.26	4.27	4.28	4.29	4.30
a	b	d	c	c	b	a	b	a	c	c	d	c	b	c

4.31	4.32
d	d

Hoofdstuk 5

5.1	5.2	5.3	5.4	5.5	5.6	5.7	5.8	5.9	5.10	5.11	5.12	5.13	5.14	5.15
c	a	a	a	a	b	b	d	c	b	a	b	a	b	b

5.16	5.17	5.18	5.19	5.20	5.21	5.22	5.23	5.24	5.25	5.26	5.27	5.28	5.29	5.30
c	a	c	a	c	d	b	b	d	d	c	b	a	a	c

Hoofdstuk 6

6.1	6.2	6.3	6.4	6.5	6.6	6.7	6.8	6.9	6.10	6.11	6.12	6.13	6.14	6.15
d	c	d	a	b	b	d	a	d	c	c	c	d	b	d

6.16	6.17	6.18	6.19	6.20	6.21	6.22	6.23	6.24	6.25	6.26	6.27	6.28	6.29
c	b	c	b	a	c	c	c	d	b	b	a	d	a

Hoofdstuk 7

7.1	7.2	7.3	7.4	7.5	7.6	7.7	7.8	7.9	7.10
d	a	c	b	a	b	b	d	d	c

Hoofdstuk 8

8.1	8.2	8.3	8.4	8.5	8.6	8.7	8.8	8.9	8.10	8.11	8.12	8.13	8.14	8.15
b	a	c	b	d	a	d	d	b	a	c	b	a	a	b

Answers to questions

Question		Answer	
1	1.1	1.1	1.1
2	2.1	2.1	2.1
3	3.1	3.1	3.1
4	4.1	4.1	4.1
5	5.1	5.1	5.1
6	6.1	6.1	6.1
7	7.1	7.1	7.1
8	8.1	8.1	8.1
9	9.1	9.1	9.1
10	10.1	10.1	10.1
11	11.1	11.1	11.1
12	12.1	12.1	12.1
13	13.1	13.1	13.1
14	14.1	14.1	14.1
15	15.1	15.1	15.1
16	16.1	16.1	16.1
17	17.1	17.1	17.1
18	18.1	18.1	18.1
19	19.1	19.1	19.1
20	20.1	20.1	20.1
21	21.1	21.1	21.1
22	22.1	22.1	22.1
23	23.1	23.1	23.1
24	24.1	24.1	24.1
25	25.1	25.1	25.1
26	26.1	26.1	26.1
27	27.1	27.1	27.1
28	28.1	28.1	28.1
29	29.1	29.1	29.1
30	30.1	30.1	30.1
31	31.1	31.1	31.1
32	32.1	32.1	32.1
33	33.1	33.1	33.1
34	34.1	34.1	34.1
35	35.1	35.1	35.1
36	36.1	36.1	36.1
37	37.1	37.1	37.1
38	38.1	38.1	38.1
39	39.1	39.1	39.1
40	40.1	40.1	40.1
41	41.1	41.1	41.1
42	42.1	42.1	42.1
43	43.1	43.1	43.1
44	44.1	44.1	44.1
45	45.1	45.1	45.1
46	46.1	46.1	46.1
47	47.1	47.1	47.1
48	48.1	48.1	48.1
49	49.1	49.1	49.1
50	50.1	50.1	50.1
51	51.1	51.1	51.1
52	52.1	52.1	52.1
53	53.1	53.1	53.1
54	54.1	54.1	54.1
55	55.1	55.1	55.1
56	56.1	56.1	56.1
57	57.1	57.1	57.1
58	58.1	58.1	58.1
59	59.1	59.1	59.1
60	60.1	60.1	60.1
61	61.1	61.1	61.1
62	62.1	62.1	62.1
63	63.1	63.1	63.1
64	64.1	64.1	64.1
65	65.1	65.1	65.1
66	66.1	66.1	66.1
67	67.1	67.1	67.1
68	68.1	68.1	68.1
69	69.1	69.1	69.1
70	70.1	70.1	70.1
71	71.1	71.1	71.1
72	72.1	72.1	72.1
73	73.1	73.1	73.1
74	74.1	74.1	74.1
75	75.1	75.1	75.1
76	76.1	76.1	76.1
77	77.1	77.1	77.1
78	78.1	78.1	78.1
79	79.1	79.1	79.1
80	80.1	80.1	80.1
81	81.1	81.1	81.1
82	82.1	82.1	82.1
83	83.1	83.1	83.1
84	84.1	84.1	84.1
85	85.1	85.1	85.1
86	86.1	86.1	86.1
87	87.1	87.1	87.1
88	88.1	88.1	88.1
89	89.1	89.1	89.1
90	90.1	90.1	90.1
91	91.1	91.1	91.1
92	92.1	92.1	92.1
93	93.1	93.1	93.1
94	94.1	94.1	94.1
95	95.1	95.1	95.1
96	96.1	96.1	96.1
97	97.1	97.1	97.1
98	98.1	98.1	98.1
99	99.1	99.1	99.1
100	100.1	100.1	100.1

Studie-advies

Voor een eerste cursus bevat het boek te veel informatie. Veel informatie is dan ook toegevoegd om het boek ook geschikt te maken als naslagwerk in een omgeving waar Digitale Techniek bedrijfsmatig wordt beoefend. Voor een introductiecursus op HBO of universitair niveau staat in de onderstaande tabel een voorstel. Bij het samenstellen van het boek is er naar gestreefd om deze selectie consistent te maken, zodat bij de bestudering geen manco's ten aanzien van de niet-behandelde delen ontstaan.

[illegible]

Index

- 3-state buslijn 255
- 3-state uitgang 253, 254
- 4-bit full adder 282
- 7-segment code 386

- aanduiden van negaties 209
- aantal cijfers per getal 27
- aantal operanden 373
- absorptiewetten 80
- AC noise margins 266
- active-low wiring 255, 259
- adresdecodering 382, 384
- afbeeldbaar logisch model 181, 191, 215
- afbeelding AND poort 203
- afbeelding in negatieve logica 203
- afbeelding in positieve logica 203
- afbeelding op hardware 201
- afhankelijkheid van logische variabelen 101, 102
- afleiding permissible functions 223
- afmetingen van het kanaal 243
- afrondding 27
- aktie 67
- alfanumerieke karakterverzameling 35
- algoritmekeuze 371
- alternatieven voor deelschakeling 199
- ALU 314
- analoge signaalbewerking 19
- AND array 391
- AND poort 177, 194, 196, 219, 247
- AND-OR-INVERT poort 2, 49, 339
- AND-OR-NOT poort 249
- ANSI/IEEE floating-point standaard 58
- applicatie-omgeving 215
- architectuur 169, 192, 193
- architectuur van de opteller 170
- arithmetic logic unit 314
- array structuur 385
- ASCII code 35
- associatieve wetten 40, 75, 78
- average worst-case carry 361
- axioma's van Peano 27

- BCD code 100, 386
- BCD representatie 28
- BCD tellersectie 100
- BCD tellersectie, transportfunctie 122
- BCD-to-7-segment decoder 386
- BDD 332
- BDD, don't cares in 157
- BDD, structuur van 157
- BDT, geactiveerd pad in 159
- BDT, knooppunt in 156
- BDT, volgorde variabele in 156
- belastbaarheid van een uitgang 240, 253
- belastingsafhankelijkheid 278
- bemonstertheorema 15
- bemonstertijdstippen 15
- bereik 57, 370
- bereik in teken-en-modulus 44
- best case condities 272
- betrouwbaarheid 335
- bewerking, duur van een 373
- bewerkingen op bit strings 314
- bewerkingstijd 357, 372
- bewijzen van schakelwetten 81
- binair rekenen 40
- binair code 38
- binair codeschijf 37
- binair signaalbewerking 17, 19
- binair talstelsel 19, 26
- binair variabele 70
- binair voorstelling van de mantisse 58
- Binary Decision Diagrams 96, 152
- Binary Decision Tree 96, 152
- binary decision graph 96
- binary digit 27
- bipolaire transistor 257
- bipolaire transistorsymbolen 258
- bit 27
- bit partitioning 397, 398
- bit rate 19
- bit string 29, 194
- bitgeoriënteerde aanpak 371
- bitgeoriënteerde opteller 320
- black box parameters 281
- block check 337
- blokschema 4-bit opteller 173
- Boole algebra 71
- Boolean variable 70
- bridging defect 287, 289
- buffer 278
- burst fouten 339
- buslijn 255, 270
- byte 28

- canonieke vormen logische functie 88
- capaciteit 173
- capaciteitsoverschrijding 43, 53, 54, 55, 174, 176
- capaciteitsoverschrijding, detectie 177
- capacitieve belasting 271
- carry 360
- carry generate functie 308, 312, 366
- carry propagate functie 308, 312, 366
- carry propagatie 282, 308, 311, 359, 362
- carry-completion opteller 362
- carry-save adder 322, 359, 364
- carry-skip adder 322
- carry bit 50
- Cartesisch produkt 193
- CMOS invertor 245
- CMOS logica 183
- CMOS logische families 262
- CMOS poorten, opbouw van 246
- CMOS technologie 271
- codelineaal 37
- codering van tussenresultaten 374
- codeschijf 37
- codewoorden 340
- collector 257
- combinatorische logica 201, 389
- combinatorische logica in ROM 382
- combinatorische schakeling 169, 193, 382
- combinatorische schakeling, multilevel 198
- combinatorische schakeling, optimaliseren van 198
- commutatieve wet 40, 41, 75, 93, 97
- comparator 373
- compatibel 264, 272
- complement codes 364
- complementair 88
- complementaire functies 339
- complementaire MOS schakeling 240
- complementaire schakelaar 239
- complementaire schakelaar, rusttoestand 240
- complementaire werking 68
- conditionele don't cares 100
- consensus theorema 147
- consensuswetten 80
- conversie somvorm → produktvorm 129
- conversie van binair naar hexadecimaal 34
- conversie van binair naar octaal 34
- correct-by-design 216
- critical area 291
- cross-over frequency 247

cube representation 137
cyclustijd 38

daaltijd 273

data-overdracht 266, 326
DC noise margins 265, 273
DC ruismarges 270
DC set 99, 100
decimaal talstelsel 26
decimale vermenigvuldiger 193
decoder 328, 384
decompositie 189, 197
decompositie met wisselcontacten 94
decompositie van logische functies 92, 138, 180, 330
decompositie van waarheidstabel 93

decompositie, gereduceerde 96
decompositieprobleem 191
deeln bewerking 173
deeln bewerking, repeterende 174
defect 216, 217, 287
defect modellering 237
defect statistics 290
dekking van een functie 142
dekking, bepalen van een 142, 146
dekkingstabel 143
dekkingstabel, dominante rij in 144
dekkingstabel, multiple-output 150
demultiplexer 328
dependence set 97

depletion transistor 241, 243
design-for-testability 289
detecteerbaarheid verhoogde stroom 288
detectie capaciteitsoverschrijding 53, 54, 177
digitaal signaal 16
digitale techniek 22
dissipatie 239, 241, 246, 253, 283, 288, 357, 381
distributieve wetten 40, 78, 78, 82
domein van getallen 369
domein, oorspronkelijk 49
dominante rij 145
don't care functiewaarden 122, 123, 124

don't care set 100
don't cares 130, 157, 192, 194, 200, 224, 388
doorslag 243
drain 242
drempelspanning 242, 258
drijvende komma 57
druktoets 20
duale vorm formule 206
dubbele representant van 0 56
dynamisch gedrag van poorten

221, 285, 322
dynamische toestand 70, 246, 265, 267, 274, 288

EBCDIC code 36
economische haalbaarheid 293
eenvoudigste produkt-van-sommenvorm 129
eenvoudigste som-van-produktenvorm 86
eenvoudigste somvorm 89
effectieve propagatietijd 360
emitter 257
enable ingang 253
end-around carry 324, 325
enhancement transistor 241, 243
equidistante code 37, 40
essentiële priemimplicant 125, 137, 144
excess-3 representatie 30
exclusieve OF 74, 337
EXOR bewerking 184, 337
EXOR met NAND's 185
EXOR met NOR's 185
EXOR operator 92
EXOR poort 249
exponent 57
externe informatie-overdracht 212
externe ingangen 198
externe ingangscombinatie 210
externe representatie van data 373
externe uitgang 211

FALSE 19
Fanin 271, 308, 328
fanin aanpassing 180
fanin beperking 186
fanin don't care set 199
fanin van poort 179
fanout 271
fanout don't care set 199, 201
fanout, maximale 192
fault class 219
fault coverage 292
field programmable 393
fixed-point representatie 57
flank jitter 270
flanksteilheid 241, 276
floating-point format 58
floating-point representatie 57
formule, eenvoudigste vorm 138
formuleconversie 138, 187
fout 217, 291
fout-gevoeligheid 292
foutafhandeling 101
foutdekking 217
foutencorrectie 335, 340
foutendetectie 31, 36, 335, 336, 388

foutkans 17, 217, 266
foutklasse 217
foutmodel 218, 290
foutoorzaken 214
foutsimulatie 291
fractie 27
frame check 337
frequentiespectrum 14
full adder 176, 281, 309
full adder tabel 173
full adder, 4-bit 282
functie, multiple output 149
functie, single output 149
functies, vergelijken van 157
functiesymbool 209, 210
functietabel ALU 316
functiewaarde 83, 94
functioneel ontwerp 21
fysisch ontwerpniveau 237, 246, 273
fysische eigenschappen 273
fysische poort 204
fysische werking 203

gate 242
gate array 396
gate-substraat capaciteit 246
gedetailleerd logisch ontwerp 201
geheugenlussen 397
geleidend kanaal 242
geleidingsvermogen 243
gelijkheidswetten 77
genereren van priemimplicanten 141
geordend talstelsel 26
geordende rij 26
gerealiseerde schakeling 101
gereduceerde decompositie 96
gespiegeld binaire code 38
getalbegrip 25
getalbereik 42
getalrepresentaties 25, 27, 29, 43, 370
gewicht 26, 28, 317
gewicht van de bits 173
gewogen code 28
glitch 280
globaal defect 217
grafische voorstelling van functies 152
Gray code 38
grensvlak poortsymbolen 212
grondtal 26
grondtal conversie 31
grondvorm van een symbool 208
ground bounce 266, 268

H/L component 203
H/L tabel 206

- H/L verwisselingen 103
- haakjes 75, 180, 206
- haken, vierkante 211
- half adder 176
- Hammingafstand 340
- hard defect 217
- herhaalde decompositie 94
- hexadecimaal talstelsel 34
- hexadecimale representatie 29
- high-order digit 26
- hulpfunctie 101, 189, 192, 197, 223, 309, 366, 368
- hypercube 136
- ideale schakelaar 238
- identieke subfuncties 99
- implementatieconflict 171
- implementatieverschillen 170, 171, 192, 193, 271
- implicant 124, 137
- incidentele storingen 217
- inclusieve OF 74
- incomplete specificatie 131
- indeling van het ontwerpproces 170
- individuele logica 207
- inductive fault analysis 291
- informatie-inhoud 14
- informatie-overdracht 373
- informatiedragend signaal 13
- informatieverwerkend systeem 13
- ingangsalfabet 193
- ingangscombinatie 83
- input-output transfer 211
- instelbaarheid 221
- instelsignalen 314
- instructiedecodering 391
- integer 27
- intensiteit 13
- intern datatransport 374
- interne ingang(scombinatie) 198, 210
- interne logische waarde 212, 213
- interne representatie 371, 373
- interne structuur 355
- interne uitgang 211
- interne uitgangswaarde 199, 210, 212
- interne vergrendeling 220
- interpretatie van formule 75
- introductie two's complement 49
- inversie 242
- invertor 245, 246, 287
- iteratieve consensus 146
- kader poortsymbolen 208
- kanaal 243
- Karnaughdiagram 38, 40, 115
- Karnaughdiagram, opbouw van 118
- Karnaughdiagram, plaatsing functie in 117, 129
- Karnaughdiagram, spiegellijnen in 118
- Karnaughdiagram, systematisch uitlezen 123
- Karnaughdiagram, toepassingen 130, 134
- Karnaughdiagram, volgorde van randschriften 133
- kenmerken logisch signaal 273
- keuze propositie 69, 102
- knooppunt 156
- kolomdecodering 384
- kontakt 20
- kontaktschakeling 97
- kop waarheidstabel, informatie in 84
- kostenfunctie 142, 144
- kruisende lijnen 248
- kubus 136
- kwaliteit van een test 218
- kwantisatieuis 16
- kwantiseren van signaalwaarden 19
- layout 215
- layout optimalisatie 292
- least significant digit 26
- lekstroom 258
- letter/cijfer wissel 31
- logisch abstractieniveau 71, 77
- logisch array 391
- logisch equivalent 78
- logisch identiek 382
- logisch identieke schakelingen 77
- logisch model 202
- logisch model, afbeeldbaar 181
- logisch model, theoretisch 180
- logisch neutraal 213
- logisch niet gedefinieerd 201
- logisch ontwerp 20, 201
- logisch ontwerpniveau 246, 277, 307, 342
- logisch signaal 273
- logisch uitwisselbaar 194
- logische bewerking 90
- logische bewerkingen op functies 92
- logische conventie 203
- logische familie 264, 272
- logische functie 84, 99, 334
- logische functie, complement van 127
- logische functie, cube representatie 136
- logische functie, multi-output 132
- logische functie, negatie van 127
- logische functie, onvolledig gespecificeerde 99, 131
- logische functie, volledig gespecificeerde 99, 157
- logische interpretatie 204
- logische minimalisatie 101
- logische niveaus 262
- logische nullen en enen 70, 174
- logische operator EN 71, 72
- logische operator NIET 73
- logische operator OF 73
- logische operator 212
- logische schakeling 20
- logische schema's 251
- logische spanningsniveaus 69
- logische specificatie 103, 174
- logische variabelen 70, 101
- logische werking 67
- lokaal defect 217
- look-ahead carry generator 366, 367
- low-order digit 26
- m-uit-n codes 31
- maakkontakt 67, 68, 74
- mantisse 57
- marges voor ruis 241
- mask programmable 384
- maximale fanout 192
- maximale omranding 124
- maximale propagatietijd 277, 278, 280
- maximum likelihood decoding 341
- maxterm 87
- maxtermvorm 87, 397
- mechanische schakelaar 102
- meerderheidsleidingdragers 250
- meerwaardige logica 71
- minimale BDD vorm 332
- minimale dekking 142
- minimale Hammingafstand 341
- minimale omranding 122
- minimale set van testpatronen 218
- minimale som-van-produktenvorm 121, 125, 127
- minimaliseren 194
- minimum propagatietijd 277, 278, 280
- minterm 115, 124, 392
- mintermen, samennemen van 120
- minterm signaal 383
- mintermvorm 83, 84, 132, 137, 146, 151, 159, 331
- modulaire structuur 371, 320
- modulaire vermenigvuldigers 320, 321
- module-omvang 357, 373
- modulus 44
- moduluswetten 79
- momentele waarde 85

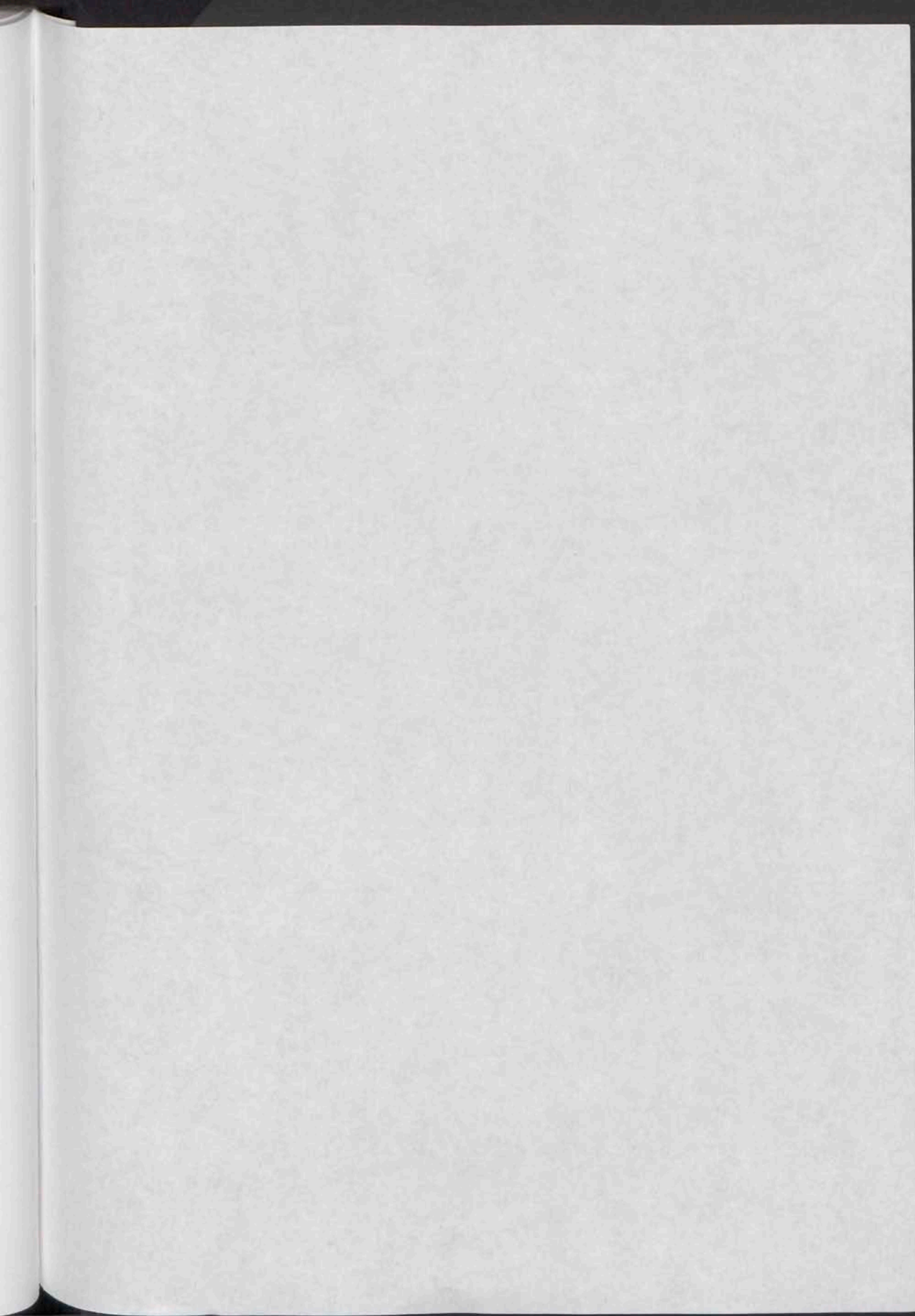
- MOS geheugen 386
- MOS geheugenarray 385
- MOS transistor 238, 242, 244
- MOS transistor als schakelaar 241
- most significant digit 26
- multi-emitter transistor 259
- multilevel combinatoriek 283
- multilevel combinatoriek, testen van 220
- multilevel combinatorische schakeling 198
- multilevel decompositie 188, 190
- multilevel poortnetwerk 281
- multiplexer 327
- n-bit full adder 173
- n-dimensionale kubus 136
- n-input AND 187
- n-input OR 187
- n-type halfgeleider 241
- n-well 246
- naburige mintermen 117
- NAND poort 182, 195, 247, 259, 260
- NAND-NAND schakelingen 183
- natuurlijke getallen 27
- nauwkeurigheid 18, 27, 57, 370
- negatie 209
- negatie-indicator 209, 212, 213, 240, 244
- negatief gewicht 49
- negaties in grensvlakken 212
- negatiestelling 90
- negatiestreep 74
- negatieve logica 202, 204, 205
- negatiwetten 79, 129
- negatiwetten van De Morgan 129
- negator 246
- netlist 215
- netwerk 215
- nibbles 30
- niet-catastrofaal defect 288
- niet-gedefinieerd gebied 202, 273, 280
- niet-gegenereerde secundaire ingangscombinaties 200
- niet-genormaliseerde informatie 211
- nMOS technologie 247
- nMOS transistor 240, 242
- noise margins 266
- NOR poort 182, 248, 251, 260
- NOR-NAND schakeling 186
- NOR-NOR schakeling 184
- NOT poort 177, 247
- NOT-AND poort 182
- NOT-AND-OR schakeling 177
- NOT-OR-AND schakeling 178
- notatie maxtermvorm 89
- notatie mintermvorm 89
- nnp transistor 257
- numerieke nullen en enen 174
- numerieke specificatie 175
- observeerbaarheid 221
- octale representatie 29
- OFF set 99
- omgevingscondities 284
- omkeereffect 253
- omranding, maximale 124
- omslagdrempel 240
- ON set 99
- ON/OFF verhouding 238
- onderhoudschema's 207
- one's complement optelling 324
- one's complement representatie 56, 361
- ontwerpniveaus 215, 342
- ontwerptraject 21
- ontwerptraject opteller 170
- ontwerpverificatie 171, 214
- ontwerpvrijheid 192, 194, 197
- onvolledig gespecificeerde functie 124
- oorspronkelijk domein 49
- opbouw van CMOS poorten 246
- open defect 287
- open ingang 181, 218
- open-collector uitgang 255, 396
- open-drain uitgang 255, 256
- opgestelde specificatie 138
- oplossingsruimte 355
- opstellen van formules 104
- optellen en aftrekken 51
- opteller met snel transport 313
- optelling 358, 369
- optelling in het one's complement 56
- optelling in het two's complement 49
- optelschakeling 170
- optelsectie 42, 175
- opteltabel 42
- optimalisatie array logica 396
- optimalisatiecriterium 334, 342
- optimalisatiefactoren 191
- optimaliseren van combinatorische schakelingen 198
- OR array 392
- OR poort 177, 195, 248
- ordered BDT 96
- orderrelatie 374
- output enable 255
- overdrachtskarakteristiek, corrigeren 390
- overgangstoestand 240
- overgangverschijnsel 37, 240, 324, 360
- overspraak 15
- p-type halfgeleider 241
- pad naar de uitgang 200
- pakkingsdichtheid 381
- PAL 394
- parallel processing 366
- parallelschakeling 73, 247, 284
- parallelvermenigvuldiger 317, 319
- parameters 217
- parameterveranderingen 17
- pariteitsbepaling 338
- parity bit 36
- parity check 336
- parity checker 337
- parity functie, BDD van 153
- parity generator 337, 357
- partieel produkt 317
- pass transistor 251, 328
- passingsprobleem 179
- path-sensitization 221
- permissible function 194, 220
- permissible functions, aantal 199
- permissible functions, afleiding van de 200, 223
- Petrickfunctie 134, 144
- pipeline opteller 364
- PLA 393
- pMOS transistor 240, 243
- pnp transistor 257
- polariteitsindicator 212, 213
- poorten per package 179
- poortequivalent 389
- poortschakeling 196, 218
- poortsymbolen 208
- positieve logica 202, 204, 248
- positioneel talstelsel 26
- potlood-en-papiermethode
- optelling 173
- priemimplicant 150
- priemimplicanten, genereren van 139, 141, 147
- primaire ingang 198, 222
- primaire ingangscombinaties 198, 200
- primaire ingangssignalen 102
- primaire ingangsvariabelen 200
- primaire uitgang 222
- prime implicant 137
- prioriteit 326
- prioriteitsvolgorde 41, 342
- priority encoder 329
- probleemaanpak 355
- produkt-van-sommenvorm 87, 128, 129, 178
- produktietesten 216
- produktieverificatie 214, 216
- produktstelling 91
- produktterm 124

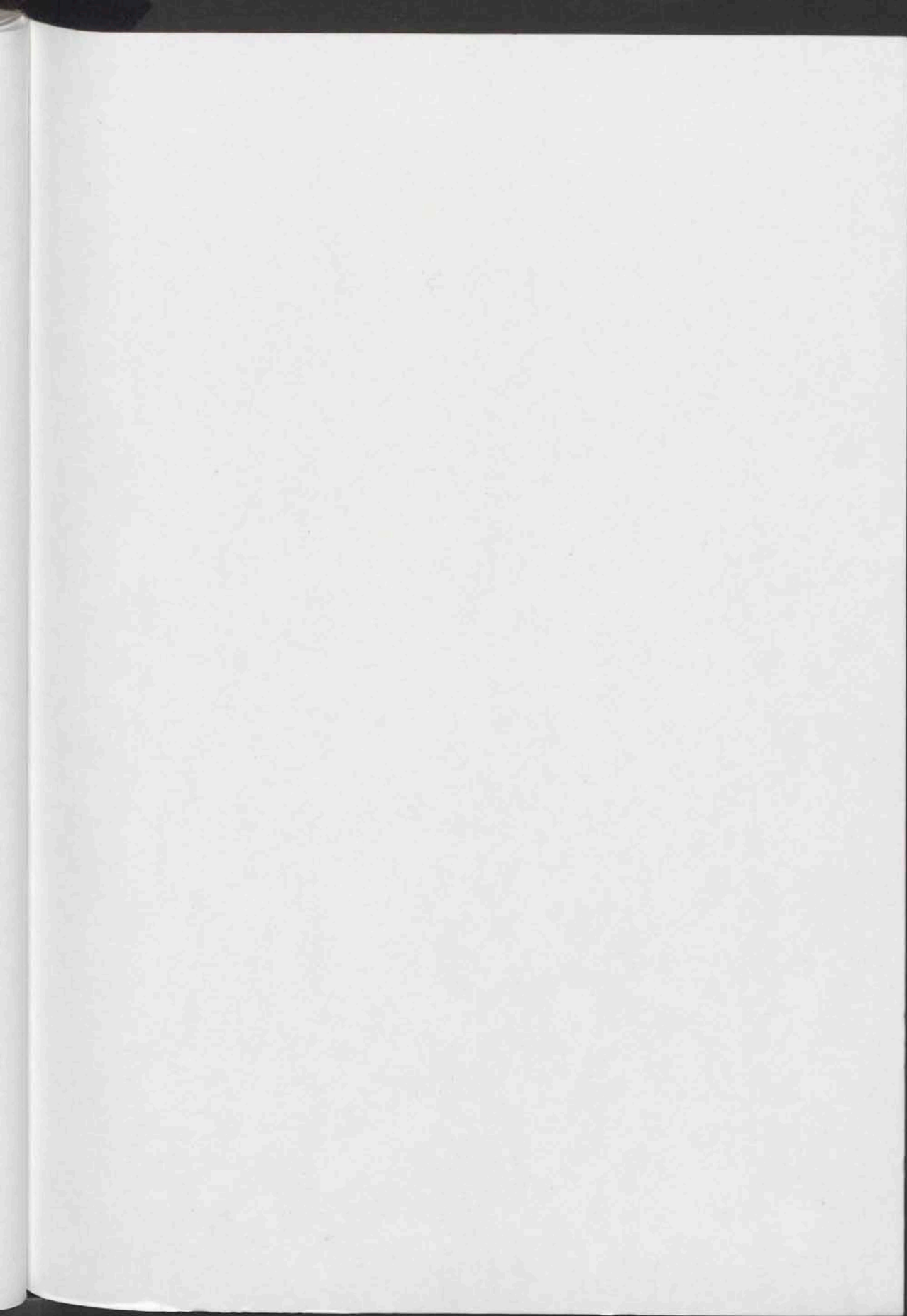
produktterm, gemeenschappelijke 150
 produktvorm 127
 programma van eisen 169
 programmable array logic 393
 programmable logic array 393
 programmeerbare logica 381
 progressieve code 37
 PROM 392
 propagatietijd 247, 277, 288, 384
 propagatietijd van poort 202, 279
 propagatietijd verbindingen 279
 propagatietijd, effectieve 360
 propagatietijd, gemiddelde 360
 propositie 19, 68, 70, 81, 102, 103
 propositie, bevestigende formulering 102
 proposities, ondubbelzinnigheid 103
 qualifying symbol 209
 Quine-McCluskey algoritme 138
 Quine-McCluskey algoritme, uitbreiding van 146
 quotiënt 32
 radix 26
 random logica 389
 randschriften bij Karnaughdiagrammen 117
 randschriften, volgorde variabelen 117, 133
 read only memory 382
 realisatie 170, 192, 193
 realisatie van opteller 171
 realisatiefase 138
 realisatieniveau 201
 rechthoekig kader 208
 reductie van logische functie 120
 redundante poorten 194, 224
 redundante schakeling 220
 redundantie 224
 Reed-Muller decompositie 334
 referentieniveaus voor de timing 273, 274
 rekenen in teken-en-modulus 44
 rekenkundige bewerkingen 41
 rekenregels 76
 rekenregels voor constanten 76
 rekenregels voor logische variabelen 77
 repeterende deelbewerking 174
 repeterende structuur 358
 representant 47
 representant tegengestelde getal 52
 representatie van de getallen 369
 reproduceerbaarheid 267
 requirements specification 169
 residu 92

rest 32
 restklasse 55
 restklassen modulo 55
 restoring logic 201, 250
 richting informatie-overdracht 208
 rijdecodering 384
 ripple carry timing 311
 robuustheid 18, 172, 267, 270, 274, 289
 ROM geheugen 382
 ruismarges 17, 263, 265, 266, 273, 288
 ruisniveau, toelaatbaar 17
 ruststand 102
 samenvatting two's complement 55
 schakelaar 2, 38, 239
 schakelaar, ideale 238
 schakelalgebra 71, 75, 76
 schakelconditie, worst case 285
 schakelsnelheid CMOS schakeling 253
 Schmitt-trigger ingang 269
 secundaire ingang 198, 222
 secundaire ingangscombinaties 198, 200
 secundaire ingangssignalen 102
 secundaire ingangsvariabelen 200
 secundaire uitgang 222
 selectiemechanisme 253, 255
 selector 97, 327, 331, 333
 sequentiële fouten 292
 serieschakeling 71, 247, 284
 Shannon decompositie 92, 152, 331, 332
 short 287
 signaal, frequentiespectrum 14
 signaal, informatiedragend 13
 signaal, kwaliteit van 17
 signaal, opbouw van 14
 signaal, tijdcontinu/intensiteitcontinu 14
 signaal, tijddiscreet/intensiteitcontinu 15
 signaalmodel 201, 202
 signaalniveau 17
 signaaloeverdracht op fysisch niveau 212
 signaaloevergang 285
 signaaloevergang, tijdstip 17
 signaalwaarde, beschikbaarheid 276
 signaalwaarde, nieuwe 278
 signaalwaarde, oude 278
 soft defect 217, 288
 som-van-produktenvorm 86, 103, 120, 125, 139, 151, 158
 sombit 50
 sommatie modulo n 50

somstelling 91
 somvorm 127
 source 242
 specificatie, onderzoek van 131
 specificatiefase 138
 specifieke kenmerken 373
 spikes 280
 spot defect 217
 standaardbelasting 278
 standaardcomponent 307
 standaardingang 271
 standaardrecept voor NAND's 188
 standaardrecept voor NOR's 188
 standaardvorm 84
 statische toestand 70, 194, 241, 246, 259, 267
 stijgtijd 273
 stimulus 67
 storingen 255
 storingsgevoeligheid 181, 217
 stroomverandering in CMOS 267
 stroomversterkingsfactor 258
 structureel ontwerpniveau 355, 358
 structurele aspecten 374
 structuur 314, 355, 356
 structuur van een ROM 382
 stuck-at fault model 218, 289
 stuck-at foutmodel 289
 stuck-at instelbaar 221
 stuck-at testbaar 220, 222, 223
 stuck-at-0 fout 218
 stuck-at-1 fout 218
 stuck-at-1 testbaar 224
 stuck-open fouten 218
 subcube 137
 subfuncties 92
 substraat 242
 superpositie 14
 symbolen voor poorten 207, 210
 symbolische logica 71
 symbool 182, 207
 symbool AND 183
 symbool NAND 183
 symboolkader 208, 212
 tabelmethode 372
 table lookup methode 33
 talstelsel 26
 technologie-mapping 179
 tegengestelde van een getal 48
 teken 44
 teken-en-modulus 43
 tekenbit 43, 48, 54
 ternaire notatie 139
 test, kwaliteit van een 218
 testbaarheid 101, 237
 testen 214, 216
 testen multilevel combinatoriek 220

- testen van IC's 289
- testingangen 221
- testpatroon 218
- testvectoren, set van 290
- theoretisch logisch model 180, 181, 215
- threshold drop 252
- threshold interval 266, 273
- threshold voltage 242
- tijdcontinu/intensiteitcontinu signaal 14, 19
- tijddiagram 204, 205
- tijddiscreet 15
- tijddiscreet/intensiteitcontinu signaal 15
- tijddiscreet/intensiteitdiscreet signaal 16
- tijdmodel 273
- tijdmodel poort 277
- tijdmodel verbinding 275
- tijdmodellen, toepassing van 283
- tijdmultiplexing 255
- timing model, statisch 285
- timing multilevel combinatoriek 279
- timing parameters 271, 284
- timing verificatie 22, 278
- toelaatbaar ruisniveau 17
- toestand 67
- top-down benadering 22, 283, 342
- top-down ontwerpstyl 215
- topologie van gegeven schakeling 197
- totem pole uitgang 253, 259
- transformatieregels 46
- transformatieregels one's complement 56
- transformatieregels two's complement 44, 46
- transmissielijn 275, 276
- transmission gate 251, 253, 328
- TRUE 19
- TTL AND-OR-NOT poort 260
- TTL logica 183
- TTL logische families 256, 261
- TTL poortschema's 260
- TTL technologie 259
- tweewaardige componenten 68
- tweewaardige signalen 67
- two's complement 46, 51, 361
- two's complement bereik 46
- two's complement cirkel 55
- two's complement representatie 46
- two-pattern testvectoren 292
- typenummer 211
- typical conditions 265, 275
- uitbreidbaarheid van schakelwetten 81
- uitgangsalphabet 193
- uitgangsbuffer 253, 271
- uitgangsinvertor 247
- uitgangsniveaus 238
- uitgangspolariteit 395
- uitgangsschakeling 253
- uitgangswaarde 199
- uitslingerverschijnselen 276
- uitvoeringsschema 251
- universele logische componenten 182, 184
- universum 86
- unsigned integers 28
- vals pad 282, 283
- variable fold 400
- veldeffect transistor 241
- Vennndiagram 86
- verarming 241
- verband mintervorm en maxtermvorm 87
- verbrekkontakt 67, 68, 74
- vereenvoudigingsregels BDT's 155
- vereenvoudigingswetten 80
- vermenigvuldigalgoritme 318
- vermenigvuldigers 320
- vermindering levensduur 271
- verrijking 241
- versterker 272
- vervangen van random logica 389
- vervorming 14, 17
- verzadigde transistor 258
- verzameling van de permissible functions 195
- vierkante haken 211
- voedingsspanning 267
- volgorde bij decompositie 96
- volgorde van bewerkingen 75, 76
- volgorde van variabelen 84
- volgorderegels schakelalgebra 177
- volledige inductie 82
- vormsymbolen 207, 208, 209
- vrije elektronen 241
- W/L verhouding 250
- waarheidstabel 68, 69, 83, 115, 206, 213
- waarheidswaarde 19, 70
- werkgebied van schakeling 284
- werkstand 102
- wetten van de Morgan 80
- wired-AND 255
- wisselcontacten 93, 97
- woordgeoriënteerde aanpak 371
- woordgeoriënteerde opteller 320
- worst-case condities 272, 284
- worst-case propagatietijd 358
- zoekruimte 355
- zwevende lijnen 255





Digitale techniek, van probleemstelling tot realisatie, deel 1
5e druk

Een geheel vernieuwde druk van Digitale Techniek is noodzakelijk geworden door diverse ontwikkelingen. Allereerst is er de technologische ontwikkeling. De CMOS-technologie is de grote winnaar gebleken. Daarnaast zijn de prestaties van de logische schakelingen drastisch groter geworden, zowel wat betreft de schakelfrequentie als de complexiteit van de schakelingen op één chip. Dit betekent bijvoorbeeld dat de strikt logische beradering zijn langste tijd heeft gehad. Steeds meer moet er op het logisch ontwerpniveau ingespeeld worden op de onderliggende ontwerp niveaus. Een onderwerp als 'timing' moet meer aandacht krijgen om de gewenste betrouwbaarheid en reproduceerbaarheid van snelle CMOS logica te kunnen garanderen. Testbaarheid en testbaar ontwerpen van IC's zijn andere voorbeelden van een vakgebied dat recent tot ontwikkeling is gekomen.

Uitgegeven in opdracht van de Vereniging voor Studie- en Studentenbelangen te Delft door de Delft University Press.

ISBN 90-407-1793-1

ISBN 90-407-1793-1



9 789040 717932

EO25