Logisch ontwerp van meerlaags programmable logic arrays

Nijenhuis, M.R.M.

Award date:
1990

Faculteit der Electrotechniek
Technische Universiteit Eindhoven
Vakgroep Digitale Systemen

LOGISCH ONTWERP VAN MEERLAAGS PROGRAMMABLE LOGIC ARRAYS

door M.R.M. Nijenhuis

Verslag van het afstudeerwerk
uitgevoerd van 01-11-1987 tot 25-08-1988
begeleider : Ir. M.J.M. van Weert
afstudeerhoogleraar : Prof. ir. M.P.J. Stevens.

Samenvatting

Twee laags (AND-OR) programmable logic arrays (PLA's) zijn een implementatievorm van twee laags logica die zich bij uitstek leent voor automatisch ontwerp. Bij deze PLA's hoeft een ingangssignaal maar door twee gates te propageren. Daarom wordt verwacht dat twee laags PLA's snel zijn in vergelijk met drie- en meer laags PLA's. Twee laags PLA's hebben echter vaak een groot aantal gates nodig om een functie te realiseren. Het benodigde oppervlak kan daarom groot zijn. De vraag rijst of het niet voordeliger is drie- of meer laags PLA's te gebruiken. Het aantal gates voor bijvoorbeeld een drie laags PLA kan veel kleiner zijn dan voor een twee laags PLA. Het benodigde oppervlak wordt dan kleiner. Als het electrisch vermogen constant blijft, stijgt het vermogen per gate. De gates hebben dan een kleinere delay. De drie laags (OR-AND-OR) PLA kan daardoor kleiner en sneller zijn dan een twee laags PLA.
Bij het vergelijk van PLA's met een verschillend aantal lagen is er een afweging tussen het aantal gates en de snelheid. Deze afweging is moeilijk te maken. In dit verslag wordt om deze reden alleen gekeken naar het aantal gates. Daarbij wordt een model gebruikt voor PLA's. Het wiskundige Σ_k-circuit is een model voor een k-laags PLA. Deze circuits zijn in het verslag gedefinieerd. Het verslag geeft verder de resultaten van een literatuurstudie naar het logisch ontwerp van Σ_k-circuits. Daarbij ligt de nadruk op een zogenaamd beperkt Σ_3-circuit. Dit is een speciaal soort Σ_3-circuit. Er wordt een partitie van de ingangsvariabelen gemaakt in disjuncte groepen. Van iedere groep worden alle maxtermen berekend in de eerste OR laag van het beperkte Σ_3-circuit.
Voor het logisch ontwerp van beperkte Σ_3-circuits worden gegeneraliseerde Booleaanse expressies (GBE's) gebruikt. GBE's zijn een veralgemenisering van de bekende Booleaanse expressies

die bij het logisch ontwerp van Σ_2-circuits (twee laags PLA's) gebruikt worden. Bestaande algoritmes voor het minimaliseren van Booleaanse expressies kunnen worden uitgebreid voor GBE's. De voor de uitvoering van de algoritmes benodigde rekentijd is groter in het geval van GBE's. Beperkte Σ_3-circuits zijn nooit groter (meer gates) dan Σ_2-circuits. In het algemeen zijn ze zelfs aanzienlijk kleiner. De grootte van de groepen en de verdeling van de ingangsvariabelen over de groepen van de partitie blijkt de afmetingen van het beperkte Σ_3-circuit te beïnvloeden. Het verslag bespreekt enkele algoritmes en manieren om de grootte van de groepen en de verdeling van de variabelen erover zo te kiezen dat het beperkte Σ_3-circuit zo klein mogelijk wordt.

Inhoudsopgave

pagina

1 Inleiding

De huidige i.c. techniek maakt het mogelijk grote, complexe (Very Large Scale Intergration) bouwstenen te realiseren ten behoeve van logische schakelingen. Door de hoge integratiegraad zijn automatische ontwerphulpmiddelen onontbeerlijk geworden. Een voorwaarde voor doelmatige, automatische ontwerpmethodes is het gebruik van regelmatige structuren. Programmable logic arrays (PLA's) hebben zo'n regelmatige structuur. Met PLA's kan men combinatorische logische functies realiseren en met een kleine uitbreiding ook sequencers. De meeste PLA's bestaan uit twee velden, namelijk een veld met logische AND poorten gevolgd door een veld met OR poorten. Om zoveel mogelijk schakelingen in hetzelfde i.c. onder te brengen optimaliseert (ook wel minimaliseert) men de logische expressies van de te realiseren logische functies. Over de minimalisatie van expressies ten behoeve van PLA's met twee velden is reeds veel bekend. De grenzen van minimalisatie worden bereikt. Een manier om de combinatorische functies op een nog kleiner i.c.-oppervlak te realiseren, is het toepassen van PLA's die bestaan uit drie velden. Deze velden bestaan achtereenvolgens uit logische OR-, AND- en OR poorten. Over de minimalisatie van deze PLA's is veel minder bekend. Dit afstudeerverslag geeft meer inzicht in de expressies ten behoeve van PLA's met drie velden en de minimalisatie ervan. Het afstudeerproject is verricht bij de vakgroep digitale systemen (EB) van de Technische Universiteit Eindhoven in het kader van een onderzoek binnen de vakgroep naar logische schakelingen, optimalisatie en simulatie.
In het verslag komen veel Engelse termen voor. Deze zijn niet vertaald vanwege een betere aansluiting met de over het algemeen engelse literatuur. Hoofdstuk 2 van het verslag introduceert PLA's en schetst het ontwerpproces met betrekking tot PLA's. In hoofdstuk 3 wordt een model gevormd van PLA's. Hiertoe worden

eerst een aantal definities gegeven over bijvoorbeeld functies, expressies en circuits. In hoofdstuk 4 is de minimalisatie van PLA's met twee velden beschreven. Hoofdstuk 5 beschrijft de minimalisatie en de problemen die zich hierbij voordoen voor een speciale soort PLA's met drie velden. Bij deze speciale PLA zijn voorwaarden opgelegd aan de verbindingen in het eerste veld. Tot slot bevat hoofdstuk 7 de conclusies en aanbevelingen. De aanbevelingen gaan onder andere over het logisch ontwerp voor PLA's met meer dan drie velden.

2 Programmable logic arrays en het ontwerpen van logica

In dit verslag komt een groot aantal keren het begrip programmable logic array (PLA) voor. Een PLA is een manier om een logische functie te realiseren. Paragraaf 2.1 geeft een antwoord op de vraag waarom juist PLA's gebruikt worden om functies te realiseren. Daarna belicht paragraaf 2.2 de ontwerpstappen van logica in het algemeen en met betrekking tot PLA's in het bijzonder.

2.1 Introductie van programmable logic arrays (PLA's)

Kosten spelen een belangrijke rol bij het realiseren van logische schakelingen (zie kostenfuncties in [7] en [12]). Alhoewel er altijd een vraag zal zijn naar logica waarbij alleen de prestaties van de logica van belang zijn en niet de kosten, bijvoorbeeld in een laboratoriumomgeving, zal er in de meeste gevallen sprake zijn van een afweging tussen kosten en prestaties. Ten gevolge van deze afweging zijn er twee verschillende methodes ontstaan om logische schakelingen te realiseren :

- Printed circuit boards.
- Geïntegreerde schakelingen (i.c.'s).

Bij allebei de methodes kunnen PLA's gebruikt worden. De methodes worden hieronder toegelicht.

2.1.1 Programmable logic arrays en printed circuit boards

Bij de eerste methode realiseert men de logische schakeling met behulp van losse kant en klare standaard bouwstenen en onderdelen. De bouwstenen en onderdelen zijn geplaatst op een board waarop ook de verbindingen zijn aangebracht. De bouwstenen kunnen SSI (small scale integration), MSI (medium scale integration) of (V)LSI ((very) large scale integration) integrated circuits (i.c.'s) zijn, bijvoorbeeld de TTL 74-serie bouwstenen. De kosten bij deze methode zijn onder andere afhankelijk van de ontwerptijd en het aantal gebruikte bouwstenen. Vanwege dit laatste is het zaak de bouwstenen zoveel mogelijk te benutten. Dit kan echter een belemmering vormen bij een gestructureerde ontwerpmethode.
Om de kosten laag te houden moet men dus enerzijds zorgen dat de gebruikte bouwstenen goedkoop zijn. Anderzijds moeten ze "op maat gemaakt zijn" voor een specifieke toepassing. Dit heeft geleid tot de zogenaamde programmable logic devices (PLD's). PLD's zijn regelmatige, geheugenachtige structuren die gebruikt worden om logische, combinatorische functies te realiseren. Vanwege hun universele karakter kan men ze in grote hoeveelheden produceren waardoor de kosten per bouwsteen laag zijn. Daarnaast levert het programmeerbare aspect de mogelijkheid om de bouwsteen (optimaal) aan te passen aan de wensen van de gebruiker.

PLD's bestaan in het algemeen uit twee velden (arrays) te weten een veld met logische AND gates en een veld met logische OR gates. De velden bestaan uit horizontale en verticale verbindingen. De horizontale lijnen zijn allemaal ingangen en de verticale lijnen zijn uitgangen of vice versa. De horizontale en verticale lijnen kruisen elkaar maar zijn electrisch gescheiden. Op een kruispunt van twee lijnen kan een transistor aanwezig zijn. Als er een transistor aanwezig is, hebben ingangen invloed op uitgangen en worden AND- en OR gates gerealiseerd. Met PLD's

kan men een of meer gewenste functie(s) realiseren als "som van produkten" van de ingangssignalen van de PLD. Het AND veld realiseert de produkten. Bij het programmeren geeft men aan welke ingangssignalen met welke AND gates en welke AND gates met welke OR gates verbonden zijn. De functies zijn de uitgangen van de OR gates. Pas na dit programmeren ligt de functie die de PLD verricht vast. Men kan een onderscheid maken [5] tussen drie soorten PLD al naar gelang de programmeermogelijkheden van de velden :

- Programmable read only memory (PROM), ook wel programmable logic element (PLE) genoemd.
- Programmable array logic (PAL).
- Programmable logic array (PLA).

In het onderstaande zijn deze drie soorten beschreven.

PROM

Bij de PROM is alleen het OR veld te programmeren. Het AND veld is vast ingedeeld. Het AND veld bevat een gate voor iedere mogelijke combinatie van ingangssignalen. Het AND veld kan dan opgevat worden als een adresdecoder. In deze context is het OR veld het gedeelte waar de data bits liggen opgeslagen. In de PROM is de te realiseren functie dus opgeslagen in de vorm van een tabel (look-up-table). Een voordeel van de PROM is dat iedere ingangscombinatie verwerkt kan worden i.e. per functie (uitgang) is precies een kolom in het OR veld nodig. Een nadeel is echter dat er geen minimalisatie van de functie mogelijk is omdat het AND veld een vaste indeling heeft.

PAL

In tegenstelling tot bij de PROM is bij de PAL alleen het AND veld te programmeren. Het OR veld is vast ingedeeld. Hierdoor ligt het aantal termen per functie vast. Een term is een in het AND veld gevormd produkt van ingangssignalen. Ook kan men elke term slechts voor een functie (uitgang) gebruiken. Een OR gate is dus verbonden met een vast aantal AND gates en een AND gate is

slechts met een OR gate verbonden. Met een PAL is een beperkte logische minimalisatie mogelijk.

PLA

Van de PLA kan men zowel het AND als het OR veld programmeren. PLA's zijn daardoor het meest flexibel. Het is nu wel mogelijk een AND gate met meerdere OR gates te verbinden. Logische minimalisatie leidt nu tot een minimaal aantal gates. Het AND veld van PLA's is in het algemeen kleiner dan dat van PROM's omdat niet alle combinaties van ingangssignalen gevormd worden.

De flexibiliteit van de verschillende soorten PLD's neemt in de hierboven gebruikte volgorde, toe. De prijs die men hiervoor moet betalen, is een toename van het chipoppervlak en/of een afname van de snelheid. Een programmeerbare gate neemt immers meer ruimte in beslag dan een "vast ingedeelde" gate.

2.1.2 Programmable logic arrays en geïntegreerde circuits

Naast realisatie op een printed circuit board bestaat de mogelijkheid de hele logische schakeling in een of enkele LSI of VLSI ((very) large scale integration) i.c.'s te integreren. Het chipoppervlak en de ontwerpkosten bepalen nu de kosten. Om wat prijs-prestatie verhouding betreft te kunnen concurreren met losse bouwstenen schakelingen, moeten er grote aantallen van een en hetzelfde i.c. geproduceerd en verkocht worden. Het minimale aantal i.c.'s dat verkocht moet worden om uit de kosten te komen, neemt steeds verder af.

Bij full custom designed i.c.'s is iedere overgang tussen de verschillende ontwerpstadia geöptimaliseerd met betrekking tot de specifieke gebruikerstoepassing van het i.c.. Grote prestatie, lange ontwerptijd en hoge ontwerpkosten zijn daarom kenmerken van

deze i.c.'s. De beperkte aanwezigheid van geautomatiseerde ontwerphulpmiddelen voor full custom designed i.c.'s is in hoofdzaak verantwoordelijk voor de hoge ontwerpkosten.

Het gebruik van standaard ontwerpprocedures voor i.c. ontwerp kan de ontwerpkosten verlagen. Al deze procedures maken gebruik van reeds ontworpen en geteste circuitcomponenten i.e. standaards met betrekking tot de layout. Hierdoor bespaart men op de kosten voor het layout ontwerp. Standaard procedures kenmerken zich door een kleinere kans op (layout) ontwerpfouten en een kortere ontwerptijd. Dit verslag beperkt zich tot een korte beschrijving van de volgende standaard layout methodes :

- Gate array.
- Standardcell.
- Macrocell.

Gate array

Bij gate arrays maakt men gebruik van een regelmatige structuur van basiscellen met daartussen bedradingskanalen. De basiscellen bestaan ieder uit twee tot acht transistoren. Door middel van in bibliotheken opgeslagen bedradingspatronen genereert men logische cellen (gates, flipflops, etc.) uit deze basiscellen. De logische cellen worden daarna verbonden volgens het logische schema.

Standardcell

De cellen bij de standardcell methode hebben een uniforme hoogte en een variabele breedte. De cellen liggen in rijen naast elkaar. De rijen zijn gescheiden door bedradingskanalen van variabele hoogte. De cellen zijn ontworpen volgens een standaard schema. De voedingslijnen zitten in alle cellen op dezelfde plaats en alle aansluitingen voor intercel connnectie liggen op vaste plaatsen.

Macrocell

Bij macrocell is de uniforme hoogte eis van de standardcell methode vervallen. Macrocells maken een veel efficiënter gebruik

van het chipoppervlak dan standardcells omdat de hoogte niet voorgeschreven is. Daarnaast zijn er complexe, door de gebruiker gespecificeerde functies beschikbaar. Een macrocell kan een blok met standardcells zijn. Daarnaast kunnen ze geheugens (RAM's, ROM's), PLA's, microprocessor kernen en ALU's bevatten. Deze kunnen vanwege hun regelmatige structuur met behulp van zogenaamde building block generators automatisch ontworpen worden. Vooral PLA macro's zijn [3] erg efficiënt voor het ontwerpen van zowel combinatorische als sequentiële schakelingen. Ten opzichte van random logic heeft een PLA het voordeel dat er makkelijk een andere functie mee te realiseren is, zonder de noodzaak van ingrijpende veranderingen in de layout [16]. De PLA behoudt de twee velden. Het aantal ingangen en het aantal gerealiseerde functies blijven gelijk. Het enige dat verandert zijn het aantal transistoren en/of de posities van de transistoren in de velden. Men spreekt wel van het veranderen van de "personality" van de PLA. Alleen de inhoud van de PLA verandert, het uiterlijk blijft hetzelfde. Ongetwijfeld zou een ontwerp met random logic dezelfde functies op een kleiner oppervlak realiseren. Random logic bestaat uit losse gates die met elkaar verbonden zijn. Er gaat daarbij geen ruimte verloren door een array structuur van de verbindingen. Men zit niet vast aan de volgorde van AND gates gevolgd door OR gates. Het aantal lagen van random logic kan groter zijn dan twee. Maar zelfs als het aantal lagen bij random logic twee is dan zullen [8] realisaties met random logic minder gates vergen dan een PLA realisatie. Bovendien kan men bij random logic van hetzelfde type gate verschillende uitvoeringen hebben al naar gelang het vermogen dat de uitgang van een gate moet leveren. De grootte van een gate is evenredig met het te leveren vermogen. Bij PLA's is men gebonden aan een electrisch vermogen dat voor alle gates gelijk is. Bij complexe (V)LSI chips offert men echter liever wat chipoppervlak op voor een regelmatige PLA structuur. Het verlies aan chipoppervlak wordt ruimschoots gecompenseerd door een kortere ontwerptijd, een kleinere kans op ontwerpfouten en de

mogelijkheid van eenvoudige modificatie. De trend bij (V)LSI is dan ook [8] minimalisatie te vervangen door standaarditeit (beperkt aantal standaardcomponenten, bijvoorbeeld RAM's, PLA's, ALU's, etc.) en regulariteit (componenten verbonden door een simpele geometrische structuur, bijvoorbeeld PLA's). PLA macro's zijn bijvoorbeeld al in gebruik in de Intel 8086, de Motorola 68000 en de Hewlett-Packard 32-bit microprocessors.

2.2 Het ontwerpproces van logica

In de vorige paragraaf is aangegeven welke plaats PLA's innemen in de techniek op dit moment. PLA's zijn een implementatievorm voor combinatorische logica. PLA's zijn geschikt voor de implementatie van zogenaamde structuurloze, Booleaanse functies. Deze functies komen [38] in praktijk voor in de hardware van computers. Voor functies met een vorm van structuur, bijvoorbeeld optellings- of vermenigvuldigingsfuncties, kunnen hele efficiënte circuits ontworpen worden. Bij dit ontwerp gebruikt men de kennis van de structuur van de functie. In dit verslag richt men zich op het ontwerpen van PLA's zonder naar de structuur van de functies te kijken. Bij het ontwerp van logische schakelingen in het algemeen, doet men er echter verstandig aan te kijken of er geen andere, meer efficiënte realisaties mogelijk zijn die gebruik maken van de structuur van de functie.
Combinatorische logica is logica waarbij de waarden van de uitgangen alleen afhankelijk zijn van de huidige ingangswaarden. Het verleden van de in- en uitgangen heeft geen enkele invloed. Sequentiële schakelingen kunnen worden opgebouwd uit een deel combinatorische logica en een aantal geheugenelementen. De combinatorische logica van de sequentiële schakeling kan heel goed met een PLA gerealiseerd worden.
De meeste PLA's bestaan uit twee lagen namelijk een veld met AND

gates gevolgd door een veld met OR gates. De gates zijn in een array structuur gerangschikt ten behoeve van automatisch ontwerp. Deze paragraaf behandelt het ontwerpproces van combinatorische logica. Het uitgangspunt hierbij is dat de schakeling uiteindelijk met een PLA wordt gerealiseerd. Het ontwerpproces kan men grofweg in de volgende drie stappen verdelen [3] :

- Functioneel ontwerp.
- Logisch ontwerp.
- Fysisch ontwerp.

Deze drie ontwerpstappen worden hieronder nader toegelicht. Tot slot wordt in paragraaf 2.2.4 nog aandacht besteed aan de testbaarheid van PLA's. Alhoewel de PLA implementatie feitelijk pas bij het fysisch ontwerp aan de orde komt, beïnvloedt de keuze voor PLA implementatie ook het logisch ontwerp. Het is daarom moeilijk een strikte scheiding te maken tussen logisch- en fysisch ontwerp omdat het belang van bepaalde factoren tijdens het logisch ontwerp afhankelijk is van de implementatie in het fysisch ontwerp. In het hier beschreven logisch (en fysisch) ontwerp is er van uitgegaan dat voor de uiteindelijke implementatie PLA's gebruikt worden. Daarom wordt in het vervolg ook wel gesproken over het logisch ontwerp van of voor PLA's. Bij de bespreking komen een aantal begrippen als (Booleaanse) functie, expressie, twee-, drie laags PLA, literal, circuit, etc. voor. In het volgende hoofdstuk worden deze begrippen formeel omschreven. Het zwaartepunt van het verslag ligt bij het logisch ontwerp ten behoeve van PLA's. De besprekingen van het functioneel-, fysisch ontwerp en het testen van PLA's zijn daarom beknopt. Er zijn echter een aantal artikelen vermeld waarin nadere informatie over deze onderwerpen staat.

2.2.1 Functioneel ontwerp

De eerste ontwerpstap is het functioneel ontwerp. Hierbij stelt men om te beginnen de functionele specificaties van de combinatorische schakeling op. Dit kan geschieden met behulp van een beschrijvingstaal. De functionele specificaties worden vervolgens vertaald. Dit resulteert in een Booleaanse functie of expressie die het van de schakeling gewenste gedrag beschrijft. Het functioneel ontwerp wordt in dit verslag niet nader toegelicht. In het vervolg van het verslag is de aanwezigheid van een Booleaanse functie of expressie verondersteld.

2.2.2 Logisch ontwerp

De tweede ontwerpstap is het logisch ontwerp. Bij het logisch ontwerp maakt men altijd gebruik van Booleaanse expressies. Het kan zijn dat in het functioneel ontwerp alleen een functie-beschrijving is gevormd. In dat geval maakt men een expressie die bij deze functie past. Zoals later zal blijken zijn er vele mogelijke expressies te geven die een en dezelfde functie representeren.
Bij het logisch ontwerp transformeert men een expressie in een andere expressie. Beide expressies representeren echter dezelfde functie. Het zal duidelijk zijn dat dit transformeren niet willekeurig gebeurt. Men streeft naar een optimaal logisch ontwerp. Optimaal wil zeggen dat de uiteindelijke expressie het best bruikbaar is bij implementatie. Voor de optimalisatie zijn criteria nodig. De belangrijkste hiervoor zijn [15], [8] de reeds in paragraaf 2.1 genoemde kosten en prestaties. Deze criteria worden hieronder nader toegelicht.

Er zijn verschillende factoren aan te wijzen waarmee men een kostenfunctie kan opstellen. De meeste van deze factoren hebben direct of indirect te maken met oppervlak. Het belang van een factor voor de kostenfunctie is afhankelijk van de uiteindelijke implementatievorm van de logica. Enkele factoren zijn :

- Het aantal gebruikte gates.
- Het aantal verschillende types gates.
- Het aantal ingangen per gate.
- Het aantal gebruikte i.c.'s.
- Het aantal verbindingen.
- De structuur van de verbindingen.

De prestatie van een schakeling wordt voornamelijk bepaald door de vertragingstijden en het opgenomen vermogen. Vertragingstijden hangen direct samen met het aantal lagen waaruit de logica bestaat.
Merk op dat hier een koppeling is gemaakt tussen het aantal lagen van logica in het logisch ontwerp en de gevolgen ervan bij implementatie. Een algemene regel is dat de vertragingstijd toeneemt als het aantal lagen stijgt. De volgende redenering laat echter zien dat deze regel slechts onder bepaalde voorwaarden geldig is. Door toename van het aantal lagen neemt het aantal gates af. Ook het aantal ingangen per gate neemt af. Dit zijn ervaringsfeiten die naast de prestatie ook de kosten beïnvloeden. Over het verband tussen het aantal lagen en het aantal gates volgt later meer. Bij implementatie kan dit betekenen dat er minder en kortere verbindingen nodig zijn. Bij een gelijkblijvend electrisch vermogen kan dit leiden tot snellere gates. De totale totale schakeling kan hierdoor even snel blijven of zelfs sneller worden. De algemene regel geldt wel in het geval dat het aantal gates en het aantal ingangen per gate ongeveer gelijk blijven.

Hieronder wordt het logisch ontwerp toegelicht voor een uiteindelijke implementatie in twee of drie laags PLA's.
De invloed van de structuur van de verbindingen op het logisch ontwerp verdient enige aandacht. Een twee laags PLA is een

mogelijke implementatievorm voor logica in het algemeen en twee laags AND-OR logica in het bijzonder. De simpele geometrische matrix- of arraystructuur van PLA's is, in tegenstelling tot de structuur van random logic, bij uitstek geschikt voor automatische bewerking. Dit draagt bij tot een verlaging van de ontwerpkosten. De PLA realiseert functies als som van produkten van ingangssignalen. De structuur van de verbindingen is een AND veld gevolgd door een OR veld. In de PLA komen dus slechts twee gate types voor namelijk AND- en OR gates. Een expressie van de te realiseren functie moet dan uiteindelijk ook in een zogenaamde "disjunctive form" staan. De functie is dan geschreven als som van produkten van literals. In deze expressies komen alleen logische AND en OR functies voor. Op deze manier is er een één op één afbeelding tussen de expressie en de implementatie in een twee laags PLA.
Andere kosten factoren die in rekening zijn gebracht, zijn het aantal gates en het aantal ingangen per gate.
In hoofdstuk 4 is beschreven hoe een expressie geminimaliseerd kan worden. Bij minimalisatie herschrijft men de expressie met de kostenfactoren "aantal gebruikte gates" en "aantal ingangen per gate" als uitgangspunt.

Bij twee laags PLA's heeft een hoge snelheid de hoogste prioriteit. Men verwacht immers dat twee laags PLA's sneller zijn dan drie of meer laags omdat ingangssignalen door minder gates hoeven te propageren om aan de uitgang te verschijnen. Zoals hierboven al is aangegeven hoeft dit echter helemaal niet zo te zijn. Bij twee laags PLA's is het logisch ontwerp in de eerste plaats optimaal met betrekking tot de prestatie, meer in het bijzonder tot het aantal lagen. Zoals zo vaak het geval is, is een ontwerp dat optimaal is met betrekking tot een criterium verre van optimaal voor een of meerdere andere criteria. Dit leidt [8] in feite tot een suboptimaal totaal ontwerp. Ook bij twee laags PLA's is dit het geval. Het aantal gates en het aantal ingangen per gate is bij twee laags PLA's erg hoog. In deze

context is het aardig de uitspraak van Fleisher en Maissel [7] uit 1975 te vermelden. Fleisher en Maissel beweren dat schakelingen met gemiddeld drie tot zes lagen ook het kleinste aantal gates bevatten. Minder lagen ("grotere prestatie") leidt tot een significante toename van het aantal gates en dus ook van de kosten. Meer lagen levert geen of slechts een geringe afname van het aantal gates op. Om een optimaler totaal ontwerp te krijgen ligt het dus voor de hand naar alternatieven te zoeken. Een schakeling met drie (meer dan twee) lagen is zo'n alternatief.
Dit verslag houdt zich daarom voor een belangrijk deel bezig met het logisch ontwerp voor drie laags PLA's. Ook deze PLA heeft een regelmatige arraystructuur. De velden bestaan ook hier weer uit AND- of OR gates. Het aantal verschillende types van gates is hier dus ook twee. Net als bij twee laags PLA's leidt dit tot beperkingen voor de vorm van de expressie. Met een geschikte notatie is ook bij drie laags PLA's een één op één afbeelding van expressie op implementatie mogelijk. De drie laags is gekozen omdat deze vorm (hopelijk) leidt tot schakelingen met veel minder gates en drie laags het meest op twee laags lijkt. Over het logisch ontwerp van drie laags PLA's bestaan veel minder artikelen dan over het logisch ontwerp van twee laags PLA's. Dit laatste ontwerp is volledig bekend en er bestaan automatische ontwerpmethodes voor [30]. Van drie of meer laags circuits is niet zoveel bekend. Van voor 1980 [38] zijn er op dit gebied bijna geen resultaten bekend.

Hoofdstuk 5 richt zich op het logisch ontwerp van zogenaamde decoded PLA's. Dit zijn drie laags PLA's waarvan de eerste laag op een speciale manier gebruikt wordt om alle mogelijke maxtermen van groepen van ingangssignalen te vormen. Deze maxtermen dienen daarna als ingangen voor de volgende twee lagen. De eis van het realiseren van maxtermen van groepen van ingangssignalen betekent een beperking van de mogelijkheden van de eerste laag. Toch is op deze manier in een aantal gevallen al een aardige reductie (10%-

20% [29]) van het aantal gates te bereiken. Ten opzichte van twee laags PLA's leveren decoded PLA's [3], [7] in termen van logische minimalisatie van het aantal gates nooit een verslechtering op. De prijs die men voor de decoder laag betaalt, is de extra vertraging en het extra oppervlak van de eerste (decoder) laag. Zoals al eerder verduidelijkt is, hoeft dit noch tot prestatievemindering noch tot kostenverhoging van de totale PLA te leiden. Het zal blijken dat de logische ontwerpmethode voor twee laags PLA's zodanig uit te breiden is [11] dat er ook decoded PLA's mee te ontwerpen zijn. Bestaande twee laags minimalisatie programma's (MINI, EXPRESSO) zijn geschikt (gemaakt) voor het ontwerp van decoded PLA's.

Voor het algemene geval van drie laags (en in de toekomst misschien meer laags) PLA's is deze methode (uitbreiding methode van twee lagen) niet geschikt. Het is niet bekend of er expressies zijn en hoe deze eruit zien die één op één af te beelden zijn op drie- of meer laags PLA's.
Een andere methode voor het logisch ontwerpen van drie- en meer laags PLA's kan bestaan uit het afsplitsen van subfuncties van een te realiseren Booleaanse functie. Deze subfuncties worden dan in enkele lagen gerealiseerd. De resterende lagen vormen dan de oorspronkelijke functie uit de subfuncties. De subfuncties moeten bij voorkeur functies zijn die in enkele lagen "niet te veel" gates nodig hebben. In hoofdstuk 7 vindt men een stukje theorie over de samenhang tussen het aantal gates en het aantal lagen dat gebruikt kan worden voor een "afsplits methode".

2.2.3 Fysisch ontwerp

De derde ontwerpstap van het PLA ontwerpproces is het fysisch ontwerp. In het fysisch ontwerp komt de plaatsing en de verbinding van de gates aan de orde. Tijdens het logisch ontwerp gebruikt men een symbolische vorm om de logica te beschrijven. In het fysisch ontwerp wordt deze symbolische vorm omgezet (afgebeeld) in een i.c. layout.
Het fysisch ontwerp bij i.c.'s is afhankelijk van de gebruikte layout methode. Als een PLA in een gate array ondergebracht wordt, moet men er in het fysisch ontwerp voor zorgen dat de PLA in de gate array cellen past. Fysische optimalisatie is in dit geval van groot belang. Als de PLA in een macrocell wordt ondergebracht, moet de PLA zo ontworpen worden dat de externe aansluitingen van het "PLA-blok" op gunstige plaatsen liggen. De grootte van de PLA is nu van secundair belang.
Tijdens het fysisch ontwerp streeft men naar een optimaal resultaat. Optimaal wil zeggen dat men de array structuur van de PLA zodanig verandert dat de beschikbare ruimte (chipoppervlak) zo goed mogelijk gebruikt wordt. Als men de arraystructuur verandert, moet men bedenken dat de mogelijkheid om een PLA eenvoudig te veranderen, voor een belangrijk deel verloren gaat. In praktijk kan men daarom besluiten pas na enkele produktie-series de PLA fysisch te optimaliseren. Er is hier uitgegaan van de veronderstelling dat de PLA een onderdeel is van een chip (bijvoorbeeld een macrocell). Bij de eerste produktieseries heeft men dan nog de mogelijkheid de PLA functies eenvoudig te veranderen. Na enkele series zijn de (meeste) fouten ontdekt en verbetert. Fysische optimalisatie leidt dan tot een kleiner chip-oppervlak waardoor de yield vergroot wordt. Er zijn twee gangbare, topologische technieken om de array structuur te veranderen :
- Array partitioning.
- Array folding.

Het fysisch ontwerp wordt niet verder toegelicht. Hieronder vindt men een korte uitleg van de begrippen array folding en array partitioning. In de beschrijvingen staan wel literatuur-verwijzingen naar artikelen die verder uitwijden over de details van de technieken.

Array partitioning

Bij array partitioning splitst men een PLA in meerdere kleinere PLA's. Dit gebeurt op een zodanige manier dat het totale gebruikte chipoppervlak kleiner is. In [9] staat, naar de titel van het artikel aangeeft, een samenvatting van partitioning van PLA's. Een tweede artikel is [24]. In [3] wordt gesproken over het programma "SPAM". SPAM is een versie van MINI waarmee partitioning mogelijk is om grote PLA's te minimaliseren. MINI is een programma dat heuristische methodes gebruikt voor het ontwerpen van PLA's. MINI is in het midden van de jaren 70 door IBM ontwikkeld. Een vierde artikel over PLA partitioning is [37]. Dit artikel gaat onder andere in op de partitioning problemen die ontstaan als een multiple-output functie meer uitgangen heeft dan een PLA bouwsteen toelaat.

Array folding

Array folding is de tweede techniek om de array structuur te veranderen. Folding is een techniek die tracht het gebruikte oppervlak te reduceren door gebruik te maken van het feit dat PLA arrays ijl zijn. Ijl wil zeggen dat het aantal transistoren in de AND- en OR velden gering is. Het aantal ingangen van een veld dat invloed heeft op een uitgang (gate) van een veld is relatief klein ten opzichte van het totale aantal ingangen dat invloed kan hebben. Als men geen folding toepast, wordt een aanzienlijk deel van het chipoppervlak verspild. Verspild opppervlak is [18] chipoppervlak dat dient voor verbindingen en dat niet direct bijdraagt aan de implementatie van logische functies. Verspild oppervlak zal zowel de yield als de snelheid nadelig beïnvloeden. Folding heeft het meeste effect bij ijle PLA arrays. Door de

kostenfactor van het aantal ingangen per gate te laten meetellen tijdens het logisch ontwerp, kan men ervoor zorgen dat de arrays relatief ijl worden. In [18] worden verschillende folding methodes besproken. Tevens toont men aan dat folding een NP compleet probleem is. Dit rechtvaardigt het ontwikkelen van heuristische folding algoritmes. In [17] zijn grenzen afgeleid voor de oppervlaktebesparing bij toepassing van folding. Naast deze theoretische artikelen over folding zijn er artikelen die programma's of algoritmes voor folding beschrijven. Enkele van deze artikelen zijn [1], [14], [19], [39], [40] en [41].

2.2.4 Testen van programmable logic arrays

Tot slot van dit hoofdstuk enkele regels over het testen en de testbaarheid van PLA's. Alhoewel dit verslag niet verder ingaat op het testen van PLA's, is het onderwerp belangrijk genoeg om te noemen. Testen van een i.c. gaat een steeds belangrijkere plaats innemen in het i.c. ontwerpproces. In [16] betoogt men dat de foutmodellen van de gangbare testmethodes niet volledig lijken. De in PLA's optredende fouten zouden moeilijk te beschrijven zijn met bestaande foutmodellen. Een goed foutenmodel voor PLA's bestaat uit de volgende vier foutklassen :

- Multiple stuck-at fouten.
- Multiple bridging fouten i.e. fouten tengevolge van kortsluitingen tussen naast elkaar gelegen verbindingen.
- Multiple missing- of extra crosspoint fouten i.e. fouten tengevolge van het wel of niet aanwezig zijn van een transistor in de arrays. Dit komt overeen met een extra of weggevallen verbinding tussen twee gates.
- Multiple onderbrekingen in verbindingen.

Een van de conclusies is dat een test voor multiple missing crosspoint fouten ook alle multiple stuck-at en multiple bridging fouten ontdekt. In [23] staat een overzicht van het testen van

programmeerbare logica. Andere artikelen zijn [2], [33], [22], [25] en [32] over "design for testability".

3 Formele beschrijving van programmable logic arrays

In hoofdstuk 2 zijn PLA's geïntroduceerd. Zowel als losse bouwsteen als deel van een i.c. waren PLA's een redelijk compromis tussen kosten en prestaties. Twee en drie laags PLA's bleken een vorm te zijn om twee of drie laags logica te realiseren. Het aantal lagen van logica is gelijk aan het maximale aantal gates waardoor een ingangssignaal gaat voor het een uitgang van de schakeling bereikt. In paragraaf 3.2 vindt men een uitvoerige beschrijving en indeling van de in praktijk bestaande PLA's.
Daarnaast is in hoofdstuk 2 de logische ontwerpstap van logica, in het bijzonder met betrekking tot PLA's, besproken. Tijdens het logisch ontwerp wordt een expressie, behorende bij een functie, herschreven in een andere vorm. In paragraaf 3.1 vindt men definities van de begrippen logische functie en expressie. Daarnaast staat er een definitie van een circuit. Een circuit berekent een logische functie. Een speciaal soort circuit, het Σ_k-circuit, blijkt, zoals in paragraaf 3.3 is beschreven, een goed model te zijn voor de PLA's van paragraaf 3.2. Alle Σ_k-circuits kunnen fysisch met PLA's gerealiseerd worden. Door te abstraheren van de fysische PLA naar het wiskundige Σ_k-circuit kan het logisch ontwerp van twee-, drie laags of decoded PLA's vervangen worden door het logisch ontwerp van Σ_k-circuits. Σ_k-circuits kunnen bijvoorbeeld makkelijk onderling vergeleken worden. Men heeft niets meer te maken met de prestaties en de invloed van het fysisch ontwerp ten gevolge van array folding en array partitioning. De definities en modellen van PLA's in dit hoofdstuk vereenvoudigen de beschrijving van het logisch ontwerp voor twee laags, drie laags en decoded PLA's in de volgende hoofdstukken.

3.1 Definities

In deze eerste paragraaf staan de definities van begrippen zoals functie, expressie en circuit. Er zijn ook verbanden gegeven tussen het uiterlijk van een expressie en een eigenschap van een functie. Met de begrippen wordt het onder andere mogelijk een model te vormen van PLA's en het logisch ontwerp in de volgende hoofdstukken effectief te beschrijven.

3.1.1 Definities met betrekking tot Booleaanse functies

Het verslag handelt over Booleaanse-, logische- of schakelfuncties. Tenzij expliciet anders vermeld is, wordt in het vervolg met "een functie" gerefereerd aan een Booleaanse functie.

$B_{n,m}$ [38] is de set van Booleaanse functies $f : \{0,1\}^n \longrightarrow \{0,1\}^m$.

In plaats van $B_{n,1}$ schrijft men B_n. De set B_n bestaat uit 2^a met $a=2^n$ functies. $B^*_{n,m}$ is de set van partieel gedefinieerde Booleaanse functies $f : \{0,1\}^n \longrightarrow \{0,1,?\}^m$. Men spreekt ook wel van niet volledig gespecificeerde Booleaanse functies. Analoog aan B_n staat B^*_n voor $B^*_{n,1}$. Het zal duidelijk zijn dat een volledig gespecificeerde functie uit de set $B_{n,m}$ ook in de set $B^*_{n,m}$ zit. Het omgekeerde van deze bewering is niet waar. De set $B_{n,m}$ is een deelverzameling van $B^*_{n,m}$: $B_{n,m} \subsetneq B^*_{n,m}$.

Bij PLA's zegt men dat de PLA een aantal functies realiseert. Hiermee wordt bedoeld dat de PLA meerdere uitgangen heeft die

allemaal op een andere manier afhangen van de ingangen. In het kader van bovenstaande definitie van Booleaanse functies, worden de functies van zo'n PLA beschreven met één functie uit de set $B_{n,m}$.
Een Booleaanse variabele is een discrete variabele die twee waarden (0 of 1) aan kan nemen. Een functie $f \in B_{n,m}$ is dan te schrijven als functie van n (Booleaanse) variabelen :
$f=f(x_1,\ldots,x_n)$.

Hieronder volgen nog een aantal definities met betrekking tot functies met een speciale eigenschap. Deze functies worden bij de bespreking van het logisch ontwerp in de volgende hoofdstukken gebruikt. Deze definities zijn voor het merendeel overgenomen uit de literatuur. Achter het definitienummer is de oorsprong van de definitie aangegeven. Omdat de definities uit verschillende artikelen en boeken afkomstig zijn, overlappen ze elkaar soms (gedeeltelijk).

Definitie 1 [3]
Een functie f heet monotoon stijgend (monotone increasing) in een variabele x_j indien geldt : Als x_j verandert van 0 naar 1 en de functiewaarde verandert hierdoor dan stijgt de functiewaarde van f van 0 naar 1.

Definitie 2 [3]
Een functie f heet monotoon dalend (monotone decreasing) in een variabele x_j indien geldt : Als x_j verandert van 0 naar 1 en de functiewaarde verandert hierdoor dan daalt de functiewaarde van f van 1 naar 0.

Definitie 3 [3]
Een functie heet unate in een variabele x_j als de functie monotoon dalend of monotoon stijgend is in de variabele x_j.

Definitie 4 [3]
Een functie is unate als de functie unate is in al zijn variabelen.

Definitie 5 [38]
Gegeven $a=(a_1,\ldots,a_n)$, $b=(b_1,\ldots,b_n) \in \{0,1\}^n$. Men gaat uit van de gebruikelijke ordening i.e. $a \leq b$ dan en slechts dan als $a_i \leq b_i$ voor alle i met $0 \leq 1$. Een functie f is monotoon dan en slechts dan als uit $a \leq b$ volgt $f(a) \leq f(b)$.

Een monotone functie kan ook gedefinieerd worden met behulp van definitie 1. Een monotone functie is dan een functie die monotoon stijgend is in al zijn variabelen. Beide definities zijn hier genoemd omdat beide notaties in de literatuur voorkomen.
Het zal duidelijk zijn dat alle monotone functies unate zijn. Omgekeerd is het niet zo dat een unate functie monotoon is.

Booleaanse functies kunnen gedefinieerd worden met behulp van een tabel $x \longrightarrow f(x)$ ter lengte 2^n. Als $f \in B_n$ dan volstaat het $f^{-1}(1)$ en $f^{-1}(0)$ te specificeren. $f^{-1}(y)$ is de inverse van de functie f. De inverse levert de set die bestaat uit alle $a=(a_1,\ldots,a_n) \in \{0,1\}^n$ met $f(a)=y$. De sets staan respectievelijk bekend onder de naam ON-set en OFF-set van f. Bij niet volledig gespecificeerde functies $f \in B_n^*$ komt daar nog een DC-set (don't care set) $f^{-1}(?)$ bij. Vaak is het echter eenvoudiger een functie te beschrijven door zijn gedrag vast te leggen. Een functie f kan echter ook beschreven worden met een algebraïsche expressie f. In dit verslag wordt hetzelfde symbool f gebruikt om zowel de functie als de bijbehorende expressie aan te geven.

3.1.2 Definities met betrekking tot expressies

Hieronder volgen enkele definities met betrekking tot expressies. Voordat in definitie 10 beschreven wordt wat een expressie is, volgen eerst enkele inleidende definities.

Definitie 6 [27]
Laat $a=(a_1,\ldots,a_n)$ een constante en X een variabele zijn in B^n $(B=\{0,1\})$.
X^a staat voor de afbeelding $X^a : B^n \longrightarrow B$
met $\begin{cases} X^a=0 \text{ als } X\neq a \\ X^a=1 \text{ als } X=a. \end{cases}$
X^S staat voor de functie $X^S = \bigvee_{b\in S} X^b$ met $S \subseteq B^n$.

X^S heet een literal.

De hier gebruikte logische conjunctie (OR) $x \vee y$ (ook wel $x+y$) van x en y levert de waarde 0 dan en slechts dan als $x=y=0$. De logische disjunctie (AND) $x \wedge y$ (ook wel $x.y$ of xy) levert de waarde 1 dan en slechts dan als $x=y=1$. Naast disjunctie en conjunctie bestaat er nog een logische ontkenning (NOT) $\neg$ van x. Deze ontkenning levert de waarde 1 als $x=0$ en de waarde 0 als $x=1$.

Voor literals gelden [27] de volgende rekenregels :

- $X^{S_1} \cdot X^{S_2} = X^{S_1 \cap S_2}$.

- $X^{S_1} + X^{S_2} = X^{S_1 \cup S_2}$.

- $\neg (X^{S_1}) = X^{\overline{S_1}} = X^{I-S_1}$.

- $X^I = 1$.

$- X^{\phi} = 0$.

Hierbij is X een variabele uit B^n. De verzamelingen S_1 en S_2 zijn deelverzamelingen van B^n ($S_1, S_2 \in B^n$). De verzameling I is gelijk aan B^n ($I=B^n$). De verzameling $(I-S_1)$ bestaat uit alle $x \in B^n$ die niet in S_1 zitten.

Alle expressies zijn opgebouwd met literals. Definitie 6 mag op het eerste oog omslachtig lijken. Samen met definitie 12 uit deze paragraaf zal het echter een handige manier leveren om de expressies bij het logisch ontwerp van decoded PLA's in hoofdstuk 5, weer te geven.

Voorbeeld 1

Neem n=1 in definitie 6. Er zijn nu twee afbeeldingen X^0 en X^1 waarvoor geldt :

$$X^0 = \begin{cases} 0 \text{ als } X \neq 0 \text{ dus als } X=1 \text{ (NB n=1)} \\ 1 \text{ als } X=0 \end{cases}$$

$$X^1 = \begin{cases} 0 \text{ als } X \neq 1 \text{ dus als } X=0 \text{ (NB n=1)} \\ 1 \text{ als } X=1. \end{cases}$$

Men noteert X^0 vaak als $\bar{x}$ en X^1 als x. De functies x en $\bar{x}$ vormen samen met de constante functies 0 en 1 de vier mogelijke functies in B_1. De functie x heeft dezelfde waarde als de variabele x en $\bar{x}$ is de ontkenning van de variabele X.
(einde voorbeeld)

Voorbeeld 2 [27]

In dit tweede voorbeeld om definitie 6 te verduidelijken is n=2. Neem a=(0,1) en b=(1,0). Twee mogelijke literals zijn :

$$X^a = \begin{cases} 1 \text{ als } X=(0,1) \\ 0 \text{ als } X \neq (0,1) \text{ dus als } X=(0,0) \text{ of } X=(1,0) \text{ of } X=(1,1) \end{cases}$$

$$X^{(a,b)} = \begin{cases} 1 \text{ als } X=(0,1) \text{ of } X=(1,0) \\ 0 \text{ als } X=(0,0) \text{ of } X=(1,1). \end{cases}$$

Hierbij bevat de set $S \subseteq B_n$ in het eerste geval alleen a en in

het tweede geval a en b.
(einde voorbeeld)

Definitie 7
Een term is een produkt (disjunctie) van literals. Een clause is een som (conjunctie) van literals.

Voorbeeld 3
X en Y zijn literals. (XY) is een term en (X+Y) is een clause. (X) is zowel een clause als een term.
(einde voorbeeld)

Definitie 8
Een minterm m_a voor $a=(a_1,\ldots,a_n) \in \{0,1\}^n$ is een term

$$m_a(x) = x_1^{a_1} \wedge \ldots \wedge x_n^{a_n} .$$

Definitie 9
Een maxterm M_a voor $a=(a_1,\ldots,a_n) \in \{0,1\}^n$ is een clause

$$M_a(x) = x_1^{\neg a_1} \vee \ldots \vee x_n^{\neg a_n} .$$ (NB $\neg$ is de NOT operator.)

Definitie 10 [3]
Een Booleaanse expressie f is een algebraïsche representatie van een functie f dan en slechts dan als de expressie de waarde 1 levert voor alle "inputs" uit de ON-set van f en de waarde 0 voor alle "inputs" uit de OFF-set. Voor "inputs" uit een eventuele DC-set levert de expressie 0 of 1 .

In plaats van Booleaanse expressie spreekt men in dit verslag ook wel over de expressie f.

Voorbeeld 4
Gegeven een functie $f(x,y,z) \in B_3$. Deze functie levert de waarde 1 dan en slechts dan als één van de variabelen x of y de waarde

1 heeft terwijl z de waarde 1 heeft. Enkele expressies voor f zijn : (1) $x\bar{y}z+\bar{x}yz$

(2) $(x\bar{y}+\bar{x}y)z$

(3) $(x+y)(\bar{x}+\bar{y})z$

(4) $(xz+yz)(\bar{x}+\bar{y})$

(5) $(x+y+\bar{z})(\bar{x}+\bar{y}+\bar{z})(x+y+z)(x+\bar{y}+z)(\bar{x}+y+z)(\bar{x}+\bar{y}+z)$.

Er zijn nog veel meer expressies te bedenken die f representeren. Deze zijn via de regels van de Booleaanse algebra te construeren. Voor $x\bar{y}+\bar{x}y$ schrijft men ook wel $x \oplus y$.
(einde voorbeeld)

Definitie 11

Een expressie heeft de conjunctive form (CF) of conjunctive normal form (CNF) als de expressie geschreven is als respectievelijk produkt van clauses en produkt van maxtermen. Een expressie heeft de disjunctive form (DF) of disjunctive normal form (DNF) als de expressie geschreven is als respectievelijk som van termen en som van mintermen.

Merk op dat een expressie met de CNF of DNF een speciaal geval is van respectievelijk de expressie in de CF en DF. In theorema 1 zal blijken dat er maar een expressie is met de CNF en DNF. Er zijn talloze expressies met de CF en DF voor een en dezelfde functie te geven. Deze zijn ten alle tijden in elkaar over te voeren. De CF en DF zijn makkelijk voor het beschrijven van respectievelijk Σ_k- en Π_k-circuits (zie paragraaf 3.1.3). Er is een één op één afbeelding mogelijk tussen de expressie in DF en Σ_k-circuits, welke zoals later zal blijken, een model zijn van PLA's.

Voorbeeld 5

In het vorige voorbeeld heeft expressie (1) de DNF, expressie (5) de CNF en expressie (3) de CF. De expressies (2) en (4) staan noch in de DF noch in de CF.
(einde voorbeeld)

Theorema 1 [38]

Iedere Booleaanse functie f is te schrijven als som van mintermen of als produkt van maxtermen :

$$f(x) = \bigvee_{a \in f^{-1}(1)} m_a(x) = \bigwedge_{b \in f^{-1}(0)} M_b(x).$$

Deze schrijfwijzen zijn uniek. Er is maar een expressie van een functie die als produkt van maxtermen geschreven is. En er is maar een expressie die als som van mintermen geschreven is.

Bewijs

Het bewijs dat f te schrijven is als som van mintermen luidt als volgt. Uit de definitie volgt $m_a(x)=1$ dan en slechts dan als $x=a$. De functie $f(x)$ is gelijk aan 1 dan en slecht dan als $x \in f^{-1}(1)$. Dit geldt echter dan en slechts dan als een van de mintermen $m_a(x)$ met $a \in f^{-1}(1)$ de waarde 1 oplevert.

Het tweede bewijs dat f als produkt van maxtermen te schrijven is, verloopt als volgt. Uit de definitie volgt dat $M_a(x)=0$ dan en slechts dan als $x=a$. De functie $f(x)$ is gelijk aan 0 dan en slechts dan als $x \in f^{-1}(0)$. Dit geldt dan en slechts dan als een van de maxtermen $M_a(x)$ met $a \in f^{-1}(0)$ de waarde 0 oplevert.

Q.E.D.

Met de tabel $x \rightarrow f(x)$ ter lengte 2^n is een expressie in de CNF of DNF voor f eenvoudig af te leiden. Theorema 1 geeft al aan dat er minstens twee verschillende expressies zijn die dezelfde functie representeren. Vaak zijn er meer dan deze twee expressies die een en dezelfde functie representeren. Van deze eigenschap wordt bij het logisch ontwerp gebruik gemaakt. Van een willekeurige expressie kan eenvoudig een expressie in de CNF of DNF gemaakt worden. Zo'n willekeurige expressie wordt daartoe om te beginnen in de CF of DF geschreven. Dit kan door gebruik te maken van de regels $(x+y)z=xz+yz$ en $(xy)+z=(x+y)(x+z)$. Daarna kan iedere term of clause geschreven worden als respectievelijk som

van mintermen en produkt van maxtermen. Voorbeeld 6 geeft aan hoe een term geschreven wordt als som van mintermen.

Voorbeeld 6

Gegeven een term t waarin de variabele x niet voorkomt. De term is dan te schrijven als som van twee termen waarin x wel voorkomt namelijk $t = t.1 = t\ (x + \bar{x}) = t.x + t.\bar{x}$. Dit kan herhaald toegepast worden tot in iedere term iedere variabele een keer voorkomt.

(einde voorbeeld)

In paragraaf 3.1.1 is gesproken over unate en monotone functies. Uit de volgende twee lemma's blijkt dat het aan de hand van de expressie van een functie eenvoudig te bepalen is of een functie unate of monotoon is.

Lemma 1

Een functie is monotoon dan en slechts dan als van iedere variabele x_i van de functie f slechts de literal x_i in de expressie voorkomt.

Bewijs

Uitgangspunt is een expressie in de DF. Het bewijs valt uiteen in twee delen. In het eerste deel wordt aangetoond dat als de literal x_i voorkomt de functie monotoon is in de variabele x_i. Dit kan zonder meer herhaald worden voor alle variabelen.

Stel dat een expressie wel een term $t=t'\bar{x}_i$ bevat. Zonder verlies van algemeenheid kan dan aangenomen worden dat $t''=t'x_i$ en/of een willekeurige subterm t''' niet in de expressie voorkomt. Als t'' wel voorkomt, kunnen t en t'' vervangen worden door een term $t'x_i+t'\bar{x}_i=t'$. (Dit is de omgekeerde procedure als in voorbeeld 6 gebruikt is.) Een subterm t''' van t is een term die bestaat uit een aantal literals van t. Als t''' wel voorkomt, kunnen t en t''' vervangen worden door t+t'''=t'''.

Er is nu een a met $t(a)=1$ en $a_i=0$. Neem een b met $b_j=a_j$ voor alle $j\neq i$ en $b_i=1$. Dan is $f(a)=1$ en $f(b)=0$. Dus $a<b$ en $f(a)>f(b)$. De functie f is nu niet monotoon in de variabele x_i.
In het tweede deel van het bewijs wordt aangetoond dat als f monotoon is in een variabele x_i alleen de literal x_i voorkomt in een expressie voor f. Dit kan herhaald worden voor alle variabelen.
Neem een a en b met $a_j=b_j$ voor alle $j\neq i$, $a_i=0$ en $b_i=1$. Er geldt nu $a\leq b$. Als f monotoon is zijn er drie mogelijkheden :

- $f(a)=0$ en $f(b)=0$. In dit geval komt noch de term $t'x_i$ noch de term $t'\bar{x}_i$ voor in een expressie voor f. De functie is niet afhankelijk van de variabele x_i.
- $f(a)=0$ en $f(b)=1$. Nu komt de term $t'x_i$ voor. Deze term bevat alleen de literal x_i.
- $f(a)=1$ en $f(b)=1$. In dit geval komen zowel $t'x_i$ als $t'\bar{x}_i$ voor. Deze termen zijn samen te voegen tot t' waarin geen van de literals van x_i voorkomt.

Q.E.D.

Lemma 2

Een functie is unate dan en slechts dan als van iedere variabele x_i van de functie f slechts één van de twee mogelijke literals (x_i of $\bar{x}_i$) in de expressie voorkomt.

Het bewijs wordt hier niet gegeven. Er wordt volstaan met de opmerking dat Hideo Fujiwara in [10] een expressie unate noemt als er van een variabele x_i alleen de literal x_i of $\bar{x}_i$ voorkomt.

Met behulp van lemma 1 en de definities 8 en 9 is eenvoudig in te zien dat ale maxtermen en mintermen unate functies zijn. Over sommen en produkten van unate en monotone functies kan het volgende gezegd worden. Laat f een monotone (unate) functie zijn op de set van variabelen $\{x_1,\ldots,x_n\}$ en g een monotone (unate) functie op de set $\{x_{n+1},\ldots,x_{n+m}\}$. Dan zijn het produkt f.g en de som f+g ook monotoon (unate) op de set $\{x_1,\ldots,x_{n+m}\}$.

Definitie 12 [27] [11]

Gegeven een variabele $X=(x_1,\ldots,x_n)$ in $\{0,1\}^n$ waarin x_i voor $1\leq i\leq n$ een binaire variabele is. De set van variabelen in X wordt genoteerd als $\{X\}$. Verder heet $(X_1,\ldots,X_r)$ een partitie van X dan en slechts dan als $\{X_1\}\cup\{X_2\}\cup\ldots\cup\{X_r\} = \{X\}$, $\{X_i\}\cap\{X_j\}=\phi$ $(i\neq j)$ en $\{X_i\}\neq\phi$ voor alle $i,j\in\{1,\ldots,r\}$. Het aantal variabelen in $\{X_i\}$ is $d(X_i)$. X_i wordt een partitie-element van de partitie $(X_1,\ldots X_r)$ genoemd. De partitie $(X_1,\ldots X_r)$ met $d(X_i)=1$ voor $i=1,\ldots,r$ heet de triviale partitie.

Het nut van definitie 12 wordt in hoofdstuk 5 duidelijk.

Voorbeeld 7

Neem $n=3$ en $X=(a,b,c)$ met $a,b,c\in\{0,1\}$. Als $X_1=(a,b)$ en $X_2=(c)$ dan is (X_1,X_2) een partitie van X. (X_1,X_2) met $X_1=(a,c)$ en $X_2=(b)$ is een andere partitie.
(einde voorbeeld)

3.1.3 Definities met betrekking tot circuits

Tot nog toe is gesproken over verschillende soorten Booleaanse functies en expressies die hiervan een algebraïsche representatie vormen. In het nu volgende wordt het begrip circuit geïntroduceerd. Men gebruikt hiervoor de definitie van Wegener [38]. Circuits worden gebruikt voor de berekening van Booleaanse functies. Dit kan ook met expressies in bijvoorbeeld de CNF of DNF. Maar vaak hebben simpele functies een CNF of DNF van exponentiële lengte. Daarom wordt een ander berekeningsmodel geïntroduceerd.
In het in dit verslag gebruikte berekeningsmodel voor functies simuleert men de manier waarop berekeningen met lange getallen worden uitgevoerd. Hierbij gaat men uit van een kleine set bekende operaties zoals de optelling van cijfers, het gebruik van

vermenigvuldigingstabellen, vergelijking van cijfers en "als-tests". Alle berekeningen met getallen zijn op deze basisoperaties gebaseerd.
Een eindige set Ω van Booleaanse functies met één uitgang dient als basis voor de berekening van een willekeurige Booleaanse functie. De inputs voor de berekening zijn de variabelen $x_1,\ldots,x_n$ en de constanten 0 en 1. Een stap in de berekening komt neer op het toepassen van een basisoperatie $\omega \in \Omega$ op enkele inputs en/of reeds berekende data. Een berekeningsmodel dat volgens bovenstaand principe werkt heet een circuit. In definitie 13 volgt een formele definitie van een circuit.

Definitie 13 [38]
Een Ω-circuit werkt op een vast aantal (n) Booleaanse ingangsvariabelen $x_1,\ldots,x_n$. Het circuit bestaat uit een eindig aantal (b) gates $G(1),\ldots,G(b)$. Een gate G(i) is gedefinieerd door het type $\omega_i \in \Omega$ en, indien $\omega_i \in B_{n(i)}$, een n(i)-tuple $(P(1),\ldots,P(n(i)))$ van voorgangers. Elke P(i) mag een element zijn uit $\{0,1,x_1,\ldots,x_n,G(1),\ldots,G(i-1)\}$. De Booleaanse functie die door G(i) berekend wordt heet $res_{G(i)}$. res is dus inductief gedefinieerd. Voor een input I is res_I gelijk aan I.
Als $G(i) = (\omega_i,P(1),\ldots,P(n(i)))$
dan $res_{G(i)}(x) = \omega_i\,(res_{P(1)}(x),\ldots,res_{P(n(i))}(x))$.
De outputvector $y=(y_1,\ldots,y_m)$ waar y_i een output of de uitgang van een gate is, geeft aan wat het circuit berekend, namelijk $f \in B_{n,m}$ met $f=(f_1,\ldots,f_m)$ en f_i is de functie die door y_i berekend wordt. Ook onvolledig gespecificeerde functies uit $B^*_{n,m}$ kunnen met het circuit berekend worden. Het circuit berekent $f \in B^*_n$ aan de uitgang van gate dan en slechts dan als $f(x)=res_G(x)$ voor alle $x \in f^{-1}(\{0,1\})$.

Met het circuit zijn [8] alleen combinatorische functies en geen sequentiële functies te berekenen.
De set Ω heet een complete set als iedere willekeurige functie $f \in B^*_{n,m}$ met een Ω-circuit berekend kan worden.

Voor het gemak representeert men een circuit vaak door middel van een gerichte acyclische graaf. De bronnen van de graaf zijn de ingangsvariabelen en de putten zijn de berekende functies. De knooppunten zijn de gates en knooppunt G(i) wordt gekarakteriseerd door het type ω_i en n(i) inkomende takken van voorgangers. Bovenstaande begrippen worden verduidelijkt met het volgende voorbeeld.

Voorbeeld 8 [38]

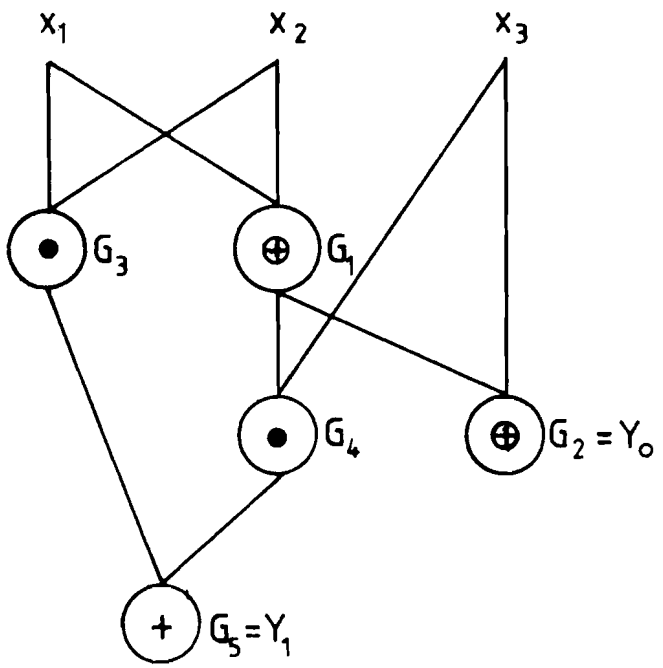
Gegeven de functie f voor een full-adder : $f(x_1,x_2,x_3)=(y_1,y_0)$. De x_1,x_2,x_3 zijn de carry-in en de twee op te tellen bits, y_1 is de carry-out en y_0 is het resultaat. Op deze manier vormen y_1 en y_0 een binaire representatie van $x_1+x_2+x_3$ of te wel $x_1+x_2+x_3=y_0+2y_1$. De functies y_1 en y_0 zijn als volgt te beschrijven : $y_0=x_1 \oplus x_2 \oplus x_3$ en $y_1=((x_1 \oplus x_2)x_3+x_1x_2)$. Een B_2 circuit dat y_0 en y_1 berekent, ziet eruit als in figuur 1 getekend is.
(einde voorbeeld)

Het zal duidelijk zijn dat er algemeen meerdere circuits zijn die dezelfde functie berekenen. De graafrepresentatie van een circuit kan zonder meer gebruikt worden als schema van een electrische realisatie van de functie. Het wiskundige begrip circuit heeft dus een electrisch equivalent.
Om circuits onderling te kunnen vergelijken en karakteriseren zijn nog een aantal begrippen nodig. Deze zijn hieronder gedefinieerd.

Definitie 14 [38]

De grootte of complexiteit (complexity) van een circuit S is C(S) welke gelijk is aan het aantal gates in S. De diepte (depth) D(S) van circuit S is het aantal gates in het langste pad van de graaf van S. De complexiteit en diepte van een functie f met betrekking tot een basis Ω zijn $C_\Omega(f)$ en $D_\Omega(f)$. $C_\Omega(f)$ en $D_\Omega(f)$ zijn respectievelijk gelijk aan het kleinste aantal gates en de minimale diepte van een Ω-circuit dat f

berekend. (NB Het circuit dat de minimale grootte oplevert hoeft niet tegelijkertijd het circuit te zijn dat een minimale diepte oplevert.)



$G_1=(\oplus,x_1,x_2)$ $G_3=(.,x_1,x_2)$ $G_5=(+,G_3,G_4)$
$G_2=(\oplus,G_1,x_3)$ $G_4=(.,G_1,x_3)$
$(Y_1,Y_0)=(G_5,G_2)$

Figuur 1 : Circuit definities (onder) van een circuit voor een full-adder en een graafrepresentatie (boven)

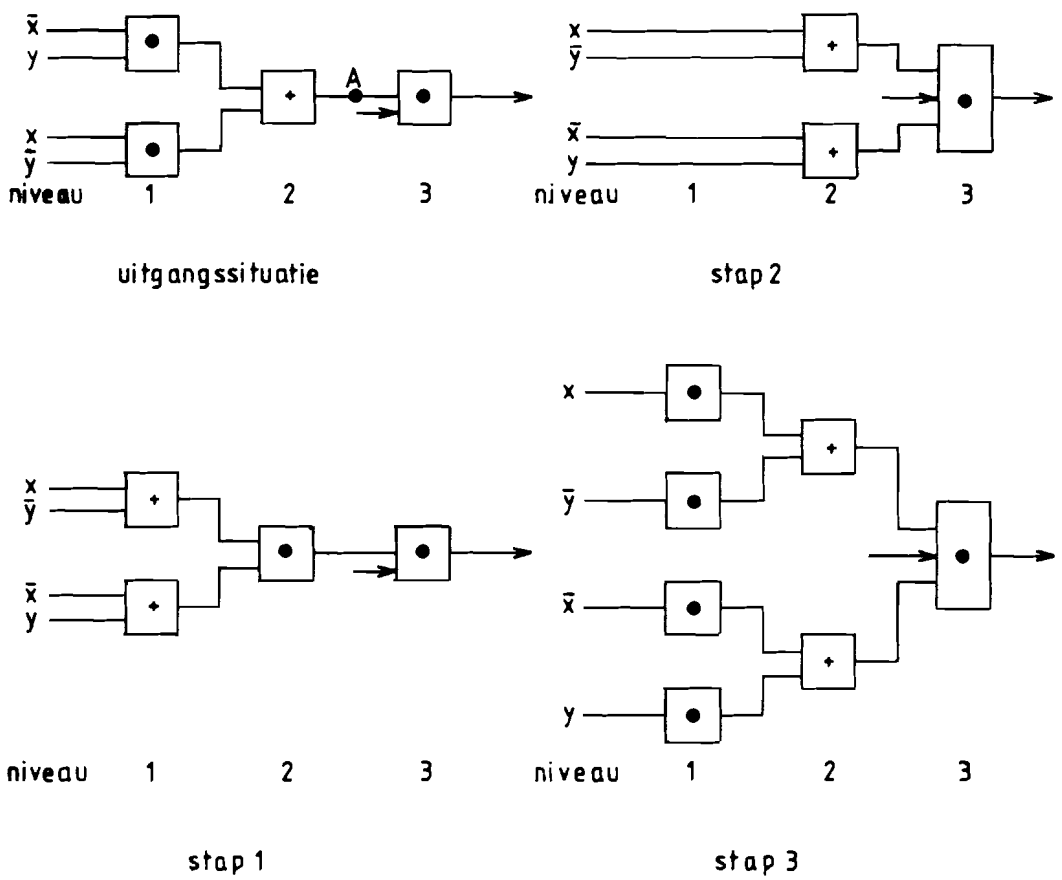
Definitie 15 [38]
De fan-out van een gate is het maximale aantal keren dat het resultaat gebruikt mag worden. Als de fan-out van een circuit gegeven is, geldt deze voor alle gates van het circuit.
De fan-in van de basis Ω van een Ω-circuit is het maximale aantal inputs voor een functie $\omega \in \Omega$.

De complexiteit van een circuit neemt [38] niet significant toe als de fan-out beperkt wordt.

Tot slot van deze paragraaf volgt de definitie van een speciaal soort circuit. Dit circuit heeft een begrensde diepte van k logische niveaus (levels).

Definitie 16 [38]
Een Σ_k-circuit (π_k-circuit) is een circuit dat bestaat uit k logische niveaus. Alle ingangen van gates op niveau 1 zijn uitgangen van gates op niveau 1-1. De literals $(x_1,\bar{x}_1,\ldots,x_n,\bar{x}_n)$ vormen niveau 0 en zijn dus de ingangen van gates op niveau 1. De niveaus k,k-2,k-4,... bestaan uit OR gates (AND gates) en de overblijvende niveaus uit AND gates (OR gates). De AND gates en OR gates hebbben een onbegrensde fan-in.

Het heeft geen zin invertors (NOT operatoren) op te nemen [38] in of tussen de niveaus in het model van Σ_k- en π_k-circuits. Als op een bepaald punt een ontkenning gewenst is dan resulteert dat uiteindelijk slechts in het nemen van ontkenningen van de ingangsvariabelen op niveau 1. De procedure waarmee dit bereikt kan worden bestaat uit het herhaald toepassen van de regels van de Morgan. Figuur 2 illustreert de procedure. Men wenst op punt A in figuur 2 in plaats van $x \oplus y$ de ontkenning $x \odot y$. In stap 1 worden de regels van de Morgan toegepast. In stap 2 comprimeert men de twee AND gates tot een nieuwe. Tenslotte herstelt men in stap 3 de AND-OR-AND structuur van de eerste drie lagen. Door deze procedure verdubbelt het aantal gates van de niveaus 1 en 2 die bij de berekening van A betrokken zijn. In het voorbeeld van figuur 2 kan het aantal gates in het nieuwe circuit terugggebracht worden tot het aantal in de uitgangssituatie.



Figuur 2 : Procedure voor het realiseren van een ontkenning

Σ_k-circuits en Π_k-circuits zijn bovendien zogenaamde synchrone circuits. Dit zijn circuits waarbij de lengte van alle paden naar een bepaalde gate even lang zijn. De Σ_k-circuits blijken een uitstekend model te zijn van de PLA schakelingen in dit verslag.

Voor alle Σ_k-circuits geldt dat een circuit met meer niveaus altijd in een circuit met minder niveaus kan worden omgezet.
Uit ieder Σ_k-circuit is altijd een Σ_i-circuit met $1 \leq i < k$ te maken. Dit kan door de niveaus 1 tot en met k-i van het Σ_k-circuit ieder te laten bestaan uit een aantal gates dat gelijk is aan het aantal ingangssignalen (meestal n). De gates op deze niveaus

hebben slechts een ingang en er zijn geen twee gates in een niveau met dezelfde ingang. Van deze eigenschap wordt in hoofdstuk 5 gebruik gemaakt bij het vergelijken van Σ_2- en Σ_3-circuits.

3.2 Classificatie van programmable logic arrays

In de volgende paragraaf wordt met behulp van het in de vorige paragraaf gedefinieerde circuit een model gevormd van PLA's. Alvorens een model te vormen van PLA's geeft deze paragraaf een nauwkeurige beschrijving van de PLA's waarvan een model gevormd wordt. Er zijn verschillende punten waarop PLA's verschillen. In dit verslag onderscheidt men de volgende punten :

- De soort van de gebruikte logica.
- Het aantal lagen (levels) waaruit de logica bestaat.
- De gebruikte set van basisfuncties. Als deze set een zogenaamde complete set is dan kan iedere willekeurige functie met functies uit deze set gerealiseerd worden.
- Beperkingen van de fan-in en fan-out van de gates.

3.2.1 Soort logica

Er zijn verschillende soorten logica. Dit verslag maakt een onderscheid tussen :

- Two-valued logic.
- Multiple-valued logic.

Hierbij staat "multiple" voor "meer dan twee".

Two-valued logic

Two-valued logic is logica die precies twee toestanden kent. Er zijn verschillende technologieën waarin de two-valued logic

geïmplementeerd kan worden. Zo bestaan er technologieën, zoals bijvoorbeeld transistor-transistor logic (TTL), emitter coupled logic (ECL) en diode-transistor logic (DTL), die zijn opgebouwd met zogenaamde bipolaire transistoren. Enkele technologieën die gebruik maken van unipolaire transistoren zijn n-channel metal oxide semiconductor (NMOS) en complementary metal oxide semiconductor (CMOS). Daarnaast zijn er nog andere technologieën [12]. De keuze voor een bepaalde technologie hangt af van verschillende factoren zoals :

- Integratie dichtheid (aantal transistoren per chip).
- Vertragingstijden.
- Aantal mogelijke bedradingslagen.
- Etcetera.

Multiple-valued logic

Bij multiple-valued logic zijn er meer dan twee toestanden. CCD (charge coupled device) is een technologie [12] waarmee dit mogelijk is. Andere technologieën voor multiple-valued logic zijn de bipolaire integrated injection logic (I^2L), ECL en de unipolaire CMOS [20]. Bij CCD wordt er gewerkt met ladingsniveaus. Het is in deze technologie mogelijk te werken met drie of meer ladingsniveaus en hier operaties op uit te voeren zoals bijvoorbeeld het modulo optellen van twee ladingen. Met CCD kan men op deze manier multiple-valued functies realiseren van multiple-valued variabelen i.e. discrete functies en variabelen die meer dan twee waarden kunnen aannemen.

Schakelingen opgebouwd met multiple-valued logic worden nog niet op grote schaal toegepast. Hiervoor zijn verschillende redenen aan te geven. Zo zijn er eigenlijk nog steeds geen echte multiple-valued logische schakelingen. In de toekomst kan [12] de geïntegreerde optoëlectronica hier een oplossing bieden. Een ander probleem vormen de synthese methodes voor multiple-valued logic. Het gebruik hierbij van bepaalde gangbare algebra's levert problemen bijvoorbeeld bij de minimalisatie. Desondanks maken de voordelen multiple-valued logic aantrekkelijk. Op dit moment zijn

verbindingen [12], [30] een van de grootste problemen bij two-valued logic. Bij integrated circuits nemen verbindingen op de chip al ongeveer 70% van het beschikbare opppervlak in beslag. Ook het toenemend aantal verbindingen tussen chips leidt tot problemen bij de behuizing. Multiple-valued logic kan gebruikt worden om de capaciteit van een verbinding te vergroten. Ook kan multiple-valued logic wiskundige voordelen opleveren bij numerieke representaties en rekenkundige bewerkingen.
PLA's kunnen zowel met two-valued als met multiple-valued logic gerealiseerd zijn. Dit verslag beperkt zich echter tot PLA's gerealiseerd met two-valued logic. In [35] en [12] is nader ingegaan op de realisatie van multiple-valued functies met behulp van multiple-valued logic CCD PLA's. De opbouw van CCD PLA's verschilt van de opbouw van two-valued logic PLA's. De CCD PLA's bestaan nog wel uit velden maar hierin vindt men geen AND- en OR gates maar gates met andere functies. Om iedere willekeurige multiple-valued functie te kunnen realiseren dient een set van deze functies compleet te zijn. In [35] wordt een complete set bestaande uit vier functies geïntroduceerd waarmee men iedere willekeurige 4-valued functie van 4-valued variabelen kan realiseren. Deze complete set bestaat uit de in dit verslag niet nader omschreven functies addition, constant, fixed overflow en inhibit.

3.2.2 Aantal lagen van programmable logic arrays

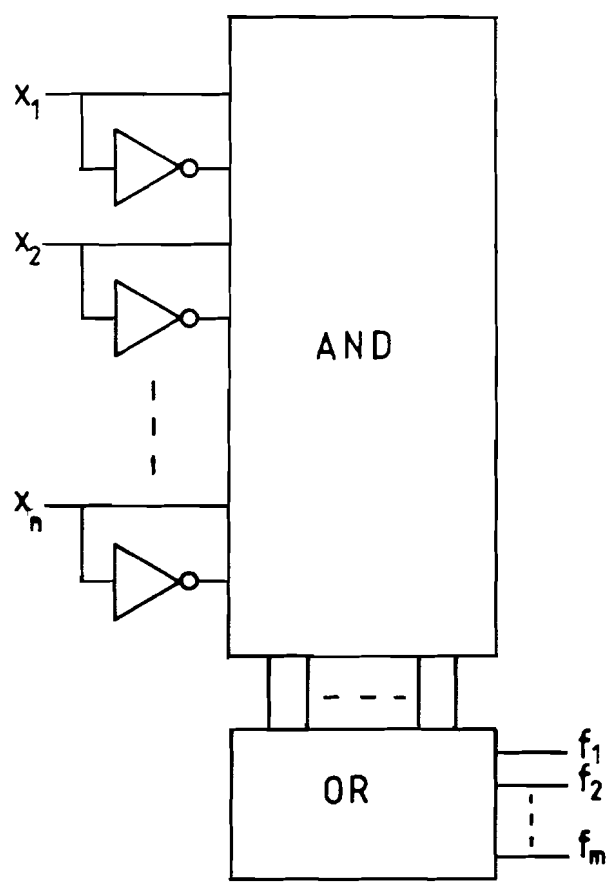
De PLA's in dit verslag zijn opgebouwd met two-valued logic. Een PLA bestaat (zoals in hoofdstuk 2 al is aangegeven) uit velden (arrays) waarin zich AND- of OR gates bevinden. PLA's kunnen verschillen in het aantal en de opbouw van de velden. Dit verslag beschrijft de volgende drie soorten PLA's :

- Two-level PLA.
- Three-level PLA.
- Decoded PLA.

De term "level" wordt hier gebruikt in plaats van "laag". Iedere laag correspondeert met een veld. Alle PLA's realiseren fysisch m binaire functies $(f_1,\ldots,f_m)$ van n binaire ingangsvariabelen $(x_1,\ldots,x_n)$.

Two-level PLA

In figuur 3 is een two-level PLA weergegeven. De PLA bestaat uit een veld met AND gates gevolgd door een veld met OR gates. De ingangen van het AND veld zijn de ingangssignalen $(x_1,\ldots,x_n)$ en de bijbehorende ontkenningen, aangegeven met $(\bar{x}_1,\ldots,\bar{x}_n)$. De

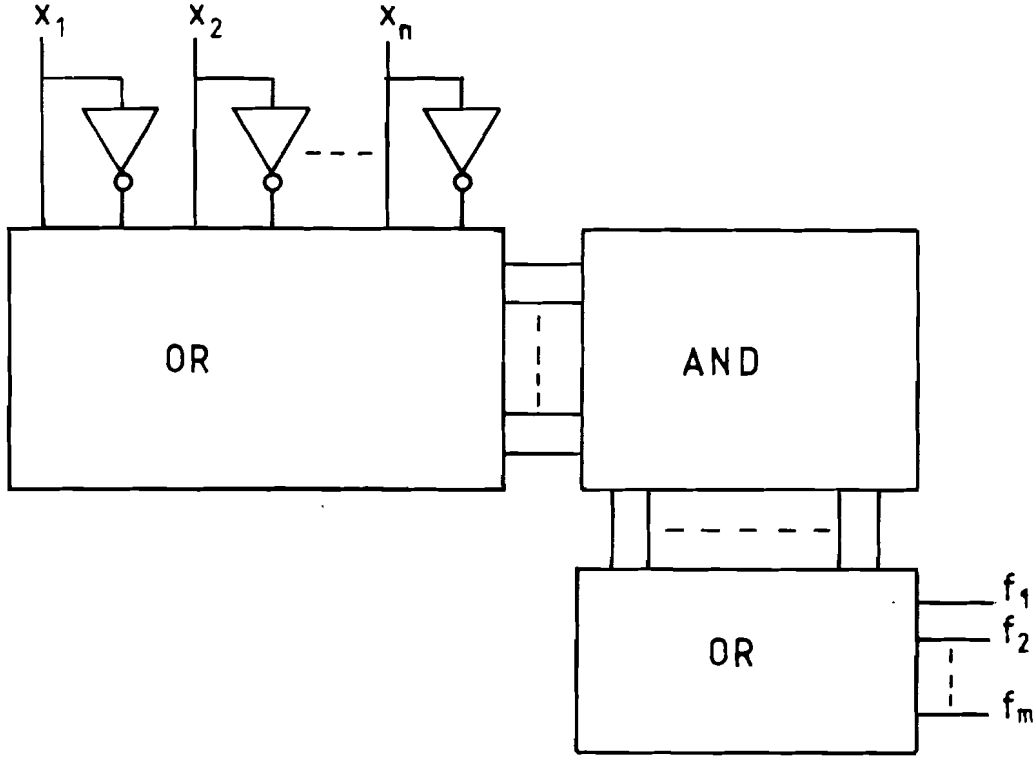


Figuur 3 : Two-level PLA

ontkenningen worden gevormd in zogeheten true-complement generatoren of phase-splitters. De ingang van een true-complement generator is een ingangssignaal x_i ($1 \leq i \leq n$) en de uitgangen zijn de signalen x_i en $\bar{x}_i$. De true-complement generatoren bevatten drivers om ervoor te zorgen dat de fan-out van de signalen x_i en $\bar{x}_i$ groot genoeg is. De signalen $(x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n)$ zijn dus de ingangen van het AND veld. Elke uitgang van het AND veld is een logische AND (produkt) van één of meer ingangen. Naast het AND veld heeft de PLA een OR veld. Ingangen van het OR veld zijn de uitgangen van het AND veld. Een uitgang is de logische OR (som) van één of meer ingangen. Uitgangen van het OR veld zijn de te realiseren functies.

Three-level PLA

In figuur 4 is een three-level PLA afgebeeld. Het AND veld en het

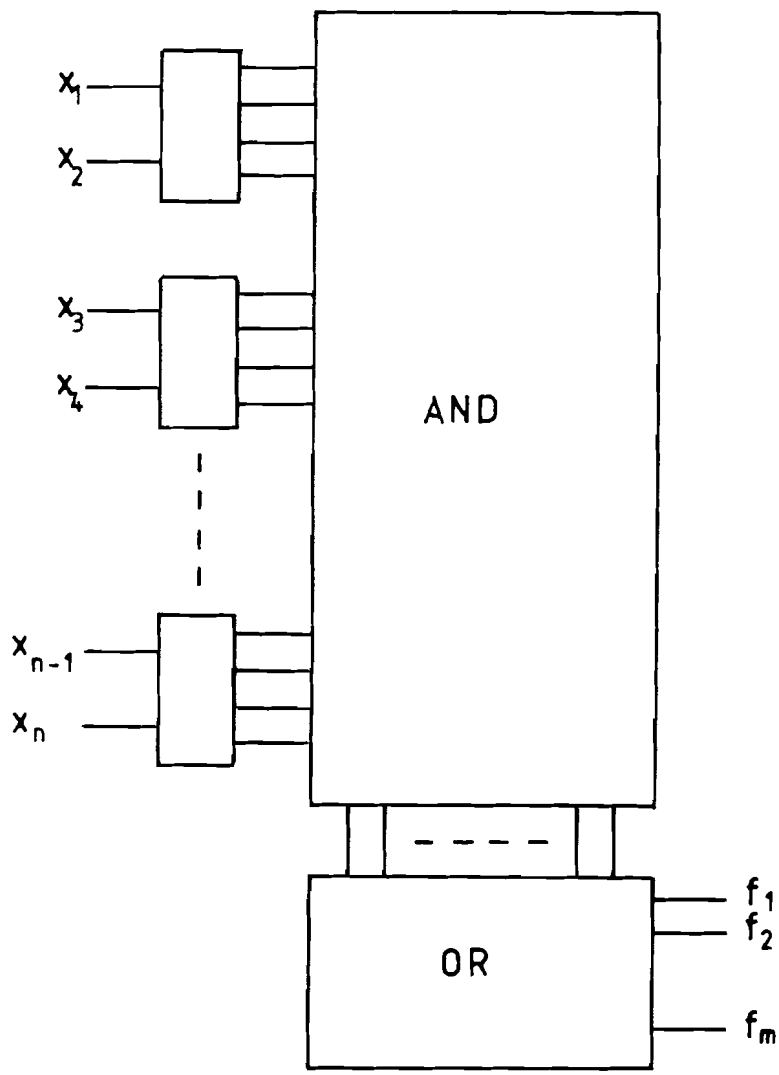


Figuur 4 : Three-level PLA

tweede OR veld zijn hetzelfde als bij de two-level PLA. Het verschil tussen two- en three-level PLA's is het extra OR veld voor het AND veld. Dit eerste OR veld heeft als ingangen de ingangssignalen $(x_1,\ldots,x_n)$ en de bijbehorende ontkenningen $(\bar{x}_1,\ldots,\bar{x}_n)$. Een uitgang van het eerste OR veld is de logische OR (som) van een of meer ingangssignalen.

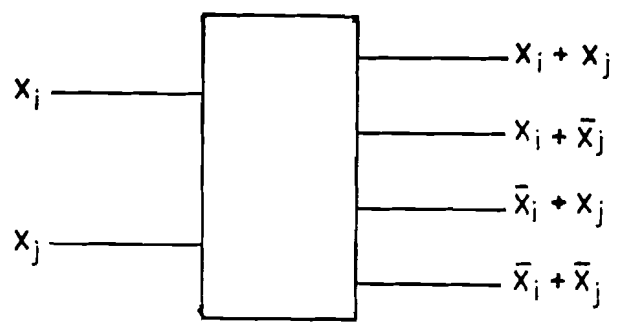
Decoded PLA

In figuur 5 is een PLA met two-bit decoders getekend. Het AND en



Figuur 5 : Decoded PLA met two-bit decoders

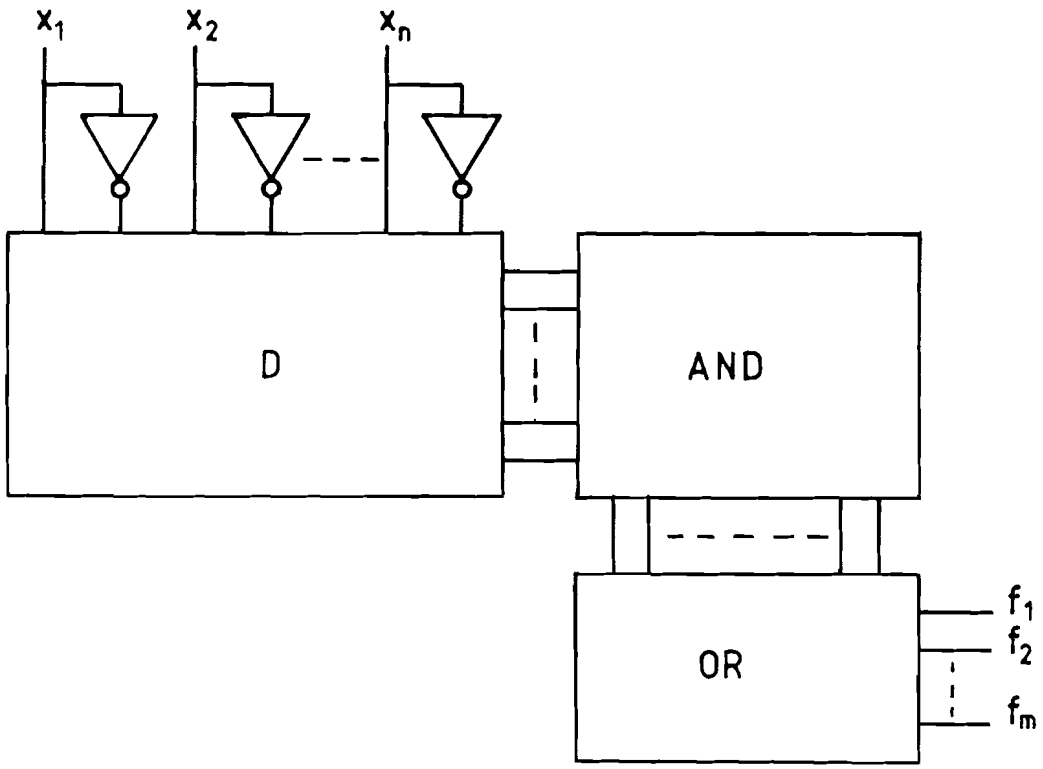
OR veld van de decoded PLA zijn identiek aan die van de two-level PLA. Het verschil tussen decoded PLA en two-level PLA zit (net als bij three-level PLA en two-level PLA) in de ingangen van het AND veld. Uit figuur 5 volgt dat de ingangssignalen $(x_1,\ldots,x_n)$ paarsgewijs aan zogenaamde two-bit decoders worden toegevoerd. Een two-bit decoder vormt alle vier de maxtermen van de twee ingaande variabelen. Dit is in figuur 6 weergegeven. De maxtermen zijn de ingangen van het AND veld. In de decoders zitten ook de drivers die zorgen voor een voldoende grote fan-out van de decoders.



Figuur 6 : Two-bit decoder

Naast two-bit decoders zijn er ook decoders met een ander aantal ingangen. Zo zijn er three-bit, four-bit, ... decoders. Deze decoders met k ingangen leveren ieder 2^k maxtermen als output. De one-bit decoder is identiek aan een true-complement generator of phase-splitter. Een decoded PLA met one-bit decoders is dan ook een two-level PLA. Een decoded PLA met één grote n-bit decoder is op te vatten als een read only memory (ROM). Het OR veld heeft dan geen functie.
De decoders in figuur 5 zijn allemaal van hetzelfde type i.e. ze hebben allemaal evenveel in- en uitgangen. Natuurlijk is het ook mogelijk om in dezelfde PLA verschillende types decoders naast elkaar te gebruiken. Men kan dit op een manier analoog aan figuur

5 realiseren. Een andere manier is in figuur 7 getekend. Het



Figuur 7 : Decoded PLA met D veld

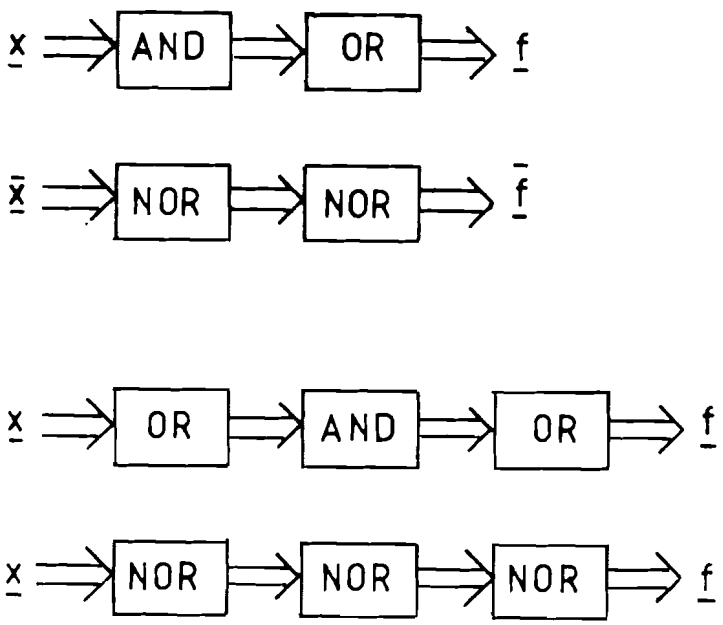
D(ecoder) veld is een OR veld waarin verschillende types decoders gerealiseerd zijn. De decoded PLA is dan een speciale vorm van een three-level PLA. Er zijn twee redenen om een D veld te gebruiken in plaats van losse decoders. Ten eerste kan men met dezelfde structuur ten alle tijden die types decoders gebruiken die voor een optimaal ontwerp nodig zijn. Ten tweede hoeven de ingangssignalen niet in een bepaalde volgorde aangeboden te worden. Dit kan voordelig zijn voor een routing algoritme dat zorgt voor de verbindingen tussen de PLA en andere delen op een chip. Een D veld is ook makkelijker te veranderen dan de losse decoders. Dit is een van de, reeds in hoofdstuk 2 genoemde, redenen om regelmatige verbindingsstructuren te gebruiken in

plaats van random logic. Losse decoders hebben het voordeel minder ruimte in te nemen dan een D veld. Dit komt omdat de drivers relatief gezien veel chip-oppervlak beslaan. De extra logica in een decoder om de maxtermen te realiseren neemt, relatief gezien weinig ruimte in.
Decoded PLA's uit figuur 5 en figuur 7 worden [7] ook wel PLA's met respectievelijk "fixed" en "variable" decoders genoemd. Een decoded PLA met D veld volgens figuur 7 wordt in de literatuur, bijvoorbeeld [27], ook wel three-level PLA genoemd. Deze naamvoering is in dit verslag niet overgenomen. Het D veld is welliswaar een veld met OR gates maar er zijn beperkingen opgelegd aan de verbindingen in dit eerste OR veld. Decoded PLA's met D veld of losse decoders zijn een speciale vorm van three-level PLA's. In dit verslag is de term three-level PLA dan ook gereserveerd voor PLA's met drie velden waarbij vooraf geen voorwaarden gesteld zijn aan de verbindingen in de velden.

3.2.3 Set van basisfuncties

Alle hierboven beschreven PLA's (two-level, three-level en decoded) bestaan uit functies uit de set {AND,OR,NOT}. Deze set is compleet. De NOT functie of ontkenning die gerealiseerd wordt door een true-complement generator, is tot nog toe niet meegeteld als type gate noch als laag bij de PLA's. De ontkenning moet echter wel opgenomen worden om de set compleet te maken. Andere verwante in PLA's gebruikte complete sets zijn {AND,NOT}, {OR,NOT} en {EXOR,NOT}.
De tot nog toe besproken PLA velden waren AND velden en OR velden. Als de PLA's in MOS technolgie ontworpen zijn dan staan vaak alleen NOR velden ter beschikking. Alle AND-OR PLA's zijn als NOR-NOR PLA's te realiseren. De NOR, samenstelling van OR en NOT, levert immers een complete set. Aan de hand van figuur 8 is dit eenvoudig in te zien. Hierbij is gebruik gemaakt van de

Morgan's regel $a.b = \overline{\bar{a}+\bar{b}}$. De ingangs- en uitgangsvectoren $\underline{x}$ en $\underline{f}$ staan voor $\underline{x}=(x_1,\bar{x}_1,\ldots,x_n,\bar{x}_n)$ en $\underline{f}=(f_1,\ldots,f_m)$. De vectoren $\underline{\bar{x}}$ en $\underline{\bar{f}}$ staan respectievelijk voor $(\bar{x}_1,x_1,\ldots,\bar{x}_n,x_n)$ en $(\bar{f}_1,\ldots,\bar{f}_m)$. Bij de two-level NOR-NOR PLA moet men dus ten eerste de uitgangen inverteren om de oorspronkelijke functies te krijgen. Daarnaast moeten de ingangen geïnverteerd worden. Dit kan eenvoudig geschieden door signalen van de true-complement generatoren paarsgewijs verwisseld in het NOR veld in te voeren.



Figuur 8 : Omzetting van two-level AND-OR in two-level NOR-NOR (boven) en omzetting van three-level OR-AND-OR in three-level NOR-NOR-NOR (onder)

De in hoofdstuk 2 genoemde functionele- en logische ontwerpstappen zijn in hoge mate onafhankelijk van de manier waarop de PLA gerealiseerd is (AND-OR, NOR-NOR, etcetera). Voor de eenvoud wordt daarom in de rest van dit verslag gewerkt met AND en OR velden.

Wat is het gevolg van het gebruik van een andere complete set van basisfuncties? De vraag rijst of de complexiteit en diepte van een circuit in het algemeen afhangen van de gebruikte set. Het antwoord hierop luidt ja. De diepte en complexiteit veranderen echter slechts met een constante factor [38] bij het overstappen van de ene complete set naar de andere. Gegeven zijn twee complete sets Ω en Ω'. De getallen c en d zijn als volgt gedefinieerd :

$c=\max\{C_{\Omega}(g) \mid g \in \Omega'\}$

$d=\max\{D_{\Omega}(g) \mid g \in \Omega'\}$.

In dit geval geldt [38] :

$C_{\Omega}(f) \leq c.C_{\Omega'}(f)$ en

$D_{\Omega}(f) \leq d.D_{\Omega'}(f)$ voor alle $f \in B_n$.

Het volgende voorbeeld verduidelijkt deze formules. In hoofdstuk 7 wordt in het kort ingegaan op PLA's met een andere complete set.

Voorbeeld 9

Gegeven zijn de sets $\Omega=\{\wedge,v,\neg\}$ en $\Omega'=\{\oplus,\wedge,v,\neg\}$. De set Ω bevat dus de AND, OR en NOT functies. De set Ω' is een uitbreiding van Ω. De set Ω' bevat namelijk de exclusive or (EXOR) functie. Omdat alle functies uit Ω ook in Ω' zitten zal de diepte en complexiteit nooit toenemen als men Ω' in plaats van Ω gebruikt. Wat zijn nu de waarden van c en d? Het is eenvoudig in te zien dat de functie g uit bovenstaande definitie de functie "$\oplus$" is. In onderstaande figuur 9 is een Ω-circuit getekend dat de EXOR functie realiseert.

Met behulp van deze figuur vindt men dan :

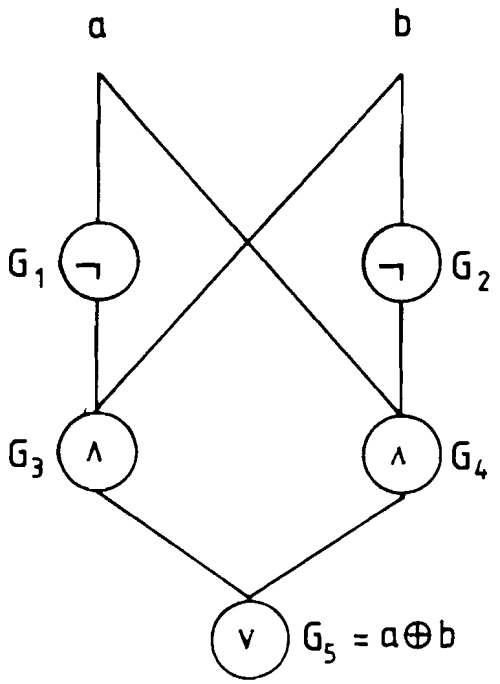
$c=C_{\Omega}(\oplus)=5$ en

$d=D_{\Omega}(\oplus)=3$.

Voor diepte en complexiteit geldt dan :

$$C_{\{\oplus,\wedge,v,\neg\}}(f) \leq C_{\{\wedge,v,\neg\}} \leq 5.C_{\{\oplus,\wedge,v,\neg\}}$$
$$D_{\{\oplus,\wedge,v,\neg\}}(f) \leq D_{\{\wedge,v,\neg\}} \leq 3.D_{\{\oplus,\wedge,v,\neg\}}.$$

Een circuit met Ω als complete set van basisfuncties heeft maximaal een vijf maal grotere complexiteit en een driemaal grotere diepte dan een overeenkomstig circuit met Ω' als complete set van basisfuncties.



Figuur 9 : Realisatie van $\oplus$ uit Ω' met behulp van Ω
(einde voorbeeld)

3.2.4 Fan-in en fan-out

De in PLA's gebruikte gates hebben in principe een onbeperkte fan-in en fan-out. In praktijk is het echter voldoende dat een gate van level l een fan-in heeft die gelijk is aan het aantal uitgangen van level l-1. De fan-out van zo'n gate hoeft niet groter te zijn dan het aantal gates op level l+1.

3.3 Modelvorming van programmable logic arrays

Hieronder wordt met behulp van definitie 16 van elk van de drie soorten PLA's een model gevormd.

Een Σ_2-circuit is een model voor een two-level PLA. Niveau 2 bestaat uit OR gates en niveau 1 uit AND gates. De ingangen van gates op niveau 1 zijn de literals $(x_1,\bar{x}_1,\ldots,x_n,\bar{x}_n)$. Alle ingangen van gates op niveau 2 zijn uitgangen van gates op niveau 1.
Een Σ_3-circuit vormt een model voor een three-level PLA. De niveaus 1 en 3 bestaan uit OR gates en niveau 2 uit AND gates. De ingangen van gates op niveau 1 zijn de literals $(x_1,\bar{x}_1,\ldots,x_n,\bar{x}_n)$. De ingangen van gates op niveau 2 en 3 zijn respectievelijk uitgangen van gates op niveau 1 en 2.
Het Σ_3-circuit is ook een model voor een decoded PLA. Uit het voorafgaande volgt dat decoded PLA's volgens figuur 7 gerealiseerd kunnen worden. Het in figuur 7 gebruikte D veld is immers een OR veld dat op een speciale manier gebruikt is. Ook in de verbindingen in het eerste niveau van het Σ_3-circuit is de speciale indeling van het eerste OR veld in te voeren. Een Σ_3-circuit dat dient als model voor een decoded PLA wordt een beperkt Σ_3-circuit genoemd.
Alle Σ_2- en Σ_3-circuits zijn fysisch te realiseren met two-valued two-level, three-level of decoded PLA's. Er blijkt een één op één afbeelding mogelijk te zijn tussen expressies in de DF en Σ_2- en beperkte Σ_3-circuits. Er kunnen hiermee efficiënte Σ_2- en beperkte Σ_3-circuits geconstrueerd worden door expressies te manipuleren (zie de hoofdstukken 4 en 5).

4 Logisch ontwerp voor two-level programmable logic arrays

In de vorige hoofdstukken zijn PLA's geïntroduceerd als een implementatievorm van (twee laags) logica. PLA's hebben regelmatige structuren die uitermate geschikt zijn voor automatisch ontwerp. Dit hoofdstuk beschrijft de logische ontwerpstap van het ontwerpproces voor two-level PLA's. Het doel van de logische ontwerpstap is in paragraaf 2.2.2 besproken.
Het logisch ontwerp wordt uitgelegd voor Σ_2-circuits. Een Σ_2-circuit is een model van een two-level PLA (zie paragraaf 3.3). Resultaten afgeleid voor het Σ_2-circuit, kunnen fysisch gerealiseerd worden met een two-level PLA.
Het logisch ontwerp bestaat uit het transformeren van een expressie in een andere. Beide expressies zijn een algebraïsche representatie van de Booleaanse functie die met het Σ_2-circuit berekend moet worden. De vorm van de expressie waarmee men in het logisch ontwerp begint, kan willekeurig zijn. De som van mintermen vorm (DNF) is hier een voorbeeld van. Deze zal vaak voorkomen als de Booleaanse functie met behulp van een tabel gedefinieerd is. De vorm van de uiteindelijke expressie is de disjunctieve vorm (DF). Voor een expressie die in deze vorm staat kan men direct een Σ_2-circuit construeren. Er is een één op één afbeelding mogelijke tussen de sommen en produkten in de expressie en respectievelijk de OR- en AND gates in het Σ_2-circuit.
Zoals gezegd, staat de uiteindelijke expressie in de DF. Dit hoofdstuk beschrijft de manier waarop men een expressie in de DF met een minimaal aantal termen en literals per term kan bepalen. Zo'n expressie wordt de minimale expressie genoemd en leidt tot Σ_2-circuits met een minimale complexiteit. Het transformeren van de originele expressie in de minimale wordt daarom ook wel aangeduid met "minimalisatie van de (originele) expressie". In het algemeen zal er meer dan een minimale expressie zijn. Het is

dan ook niet juist te spreken van "de minimale expressie". Beter is het te spreken van "een minimale expressie".
Verschilende methodes voor het minimaliseren van expressies ten behoeve van Σ_2-circuits, zijn reeds in vele boeken [38], [15], [8] en artikelen uitvoerig besproken. Het principe van de methodes wordt hier, zij het beknopt, toch beschreven. Het dient ter opfrissing van het geheugen en zorgt voor een betere aansluiting met hoofdstuk 5.
Methodes voor minimalisatie zijn altijd in twee delen te splitsen. Als eerste wordt een verzameling van zogenaamde prime implicants bepaald. Wat hieronder verstaan wordt en hoe dit in zijn werk gaat, staat in paragraaf 4.1. Daarna moet uit deze verzameling een zogenaamd cover gekozen worden. Zo'n cover bestaat uit een of meerdere prime implicants. In paragraaf 4.2 is beschreven hoe men een cover kan vormen. Minimalisatie volgens de methode uit de paragrafen 4.1 en 4.2 is voor bepaalde functies eenvoudiger dan voor andere. In paragraaf 4.3 blijkt dat het relatief makkelijk is een cover te vinden voor monotone en unate functies. De methodes in de paragrafen 4.1, 4.2 en 4.3 zijn uitgelegd voor functies uit de set B_n^*. Dit zijn functies "met een uitgang". In paragraaf 4.4 wordt in het kort besproken hoe men minimale expressies kan vinden voor functies in $B_{n,m}^*$ i.e. functies "met meerdere uitgangen". Het blijkt dat het principe voor het construeren van minimale expressies hetzelfde blijft.

4.1 Verzameling van prime implicants

In deze paragraaf is beschreven hoe men een verzameling van prime implicants kan bepalen. Alvorens daarmee te beginnen, wordt eerst antwoord gegeven op de vragen "wat is een prime implicant" en "wat is het nut ervan".
Er zijn twee kostenfactoren (zie paragraaf 2.2.2) die in rekening gebracht worden bij het minimaliseren van expressies van Σ_2-

circuits. In de resulterende expressie streeft men naar :
- Zo min mogelijk literals per term.
- Zo min mogelijke termen.

De tweede kostenfactor leidt direct tot kleinere Σ_2-circuits. De eerste leidt tot zogenaamde ijle PLA arrays bij implementatie. Deze arrays lenen zich bij uitstek voor folding technieken waarmee het gebruikte chipoppervlak gereduceerd wordt.
Een term is een produkt van literals. De literals van een binaire variabele x zijn hier x en $\bar{x}$ (zie voorbeeld 1 bij definitie 6 in paragraaf 3.1.2). De kosten van een term zijn gelijk aan het aantal literals in de term. Een expressie in de DF is een som van termen. De totale kosten van zo'n expressie zijn gelijk aan de som van de kosten van de termen waaruit de expressie bestaat. Er wordt nu gezocht naar een expressie met minimale kosten. Deze expressie levert de functiewaarde f(x) van een functie $f \in B_n^*$ voor alle x uit de ON- en OFF-set van f. Voor x in de DC-set doet de waarde van de expressie er niet toe.
De gedachte achter minimalisatie is de volgende. Als twee termen eenzelfde subterm bevatten dan is het waarschijnlijk goedkoper de subterm maar een keer te berekenen. Een subterm P van een term Q is een term waarin een of meer literals uit de term Q niet voorkomen maar die voor de rest identiek is.

Voorbeeld 10
Gegeven de termen $X=\bar{a}bc$ en $Y=\bar{a}bd$. De term bc is een subterm van X. De subterm $\bar{a}b$ is zowel een subterm van X als van Y.
(einde voorbeeld)

Een subterm zoals die hierboven is ingevoerd, wordt officieel een implicant genoemd. De formele definitie van een implicant luidt als volgt. Een term t is [38] een "implicant" van een functie $f \in B_n^*$ als

$t^{-1}(1) \subseteq f^{-1}(\{1,?\})$ en $t^{-1}(1) \not\subseteq f^{-1}(?)$.

De uitdrukking $t^{-1}(1)$ staat voor de verzameling van alle ingangscombinaties uit $\{0,1\}^n$ waarvoor de term t de waarde 1 levert. De uitdrukking $f^{-1}(\{1,?\})$ staat (zie paragraaf 3.1.1) voor de set van inputs voor de functie f, welke als functiewaarde 1 opleveren of waarvoor de functiewaarde niet gespecificeerd is. De verzameling van alle implicants van f is I(f).
Een term in een expressie die in de DF staat, zorgt ervoor dat de expressie 1 gemaakt wordt. Als een term alleen de waarde 1 levert voor inputs uit de DC-set dan is het voordeliger om deze term weg te laten. Dit is de reden voor de tweede voorwaarde voor een term om implicant te zijn.

Uit het bovenstaande volgt dat een minimale expressie in de DF alleen uit implicants zal bestaan. In het algemeen zijn er vele implicants voor een functie. Deze zijn echter lang niet allemaal nodig in een minimale expressie. Implicants die bestaan uit zo min mogelijk literals zijn het goedkoopst volgens de eerder gegeven definitie van kosten. Een implicant met een minimaal aantal literals heet een prime implicant. De formele definitie van een prime implicant ziet er als volgt uit.
Een implicant is een "prime implicant" als er geen echte subterm van de prime implicant bestaat die ook een implicant is. Een echte subterm is een subterm die niet identiek is aan de term zelf. De set van alle prime implicants van de functie f wordt weergegeven als PI(f).
De prime implicants zijn dus de kleinste implicants i.e. implicants met het minste aantal literals. Een minimale expressie voor f bestaat, volgens theorema's in [38] en [8], alleen uit prime implicants. Voor een minimale expressie zijn meestal niet alle prime implicants uit PI(f) nodig.

Hierboven is een beschrijving gegeven van het begrip prime implicant. Het belang van prime implicants is dat minimale expressie uit prime implicants bestaan. Nu rijst de vraag hoe de set PI(f) bepaald moet worden.

Er zijn verschillende methodieken om PI(f) te vinden. Deze zijn in te delen in twee groepen :
- Grafische methodieken (Karnaugh).
- Tabulaire methodieken (bijvoorbeeld Quine-McCluskey).

Hieronder staat een korte toelichting bij deze methodieken. Het valt buiten het bestek van dit verslag om deze methodes uitvoerig te beschrijven. Voor een uitvoerige beschrijving wordt naar de literatuur, bijvoorbeeld [15], [8] en [38], verwezen.

De methode van Karnaugh is een grafische methode die geschikt is voor het minimaliseren van expressies van functies van hooguit zes variabelen. De Karnaugh methode genereert niet expliciet alle prime implicants. Ze zijn er echter wel mee te bepalen.

De tabulaire methode van Quine-McCluskey bepaalt wel expliciet alle prime implicants. Uit de verzameling van alle implicants ter lengte q worden alle implicants ter lengte q-1 en prime implicants ter lengte q bepaald. Voor een functie van n binaire variabelen bepaalt men zo in ten hoogste n stappen de set van alle prime implicants. Startpunt bij Quine-McCluskey is de verzameling van alle mintermen uit de ON- en DC-set. Het is eenvoudig in te zien dat iedere minterm een implicant is.

Het aantal mintermen en het aantal implicants dat door een Quine-McCluskey algoritme gegenereerd wordt kan aanzienlijk zijn. Deze aantallen kunnen voor functies van veel binaire variabelen zo groot worden dat het Quine-McCluskey algoritme in praktijk te veel geheugenruimte en rekentijd vergt. Er zijn daarom andere tabulaire methodes ontwikkeld die niet starten met een som van mintermen expressie maar met een willekeurige expressie in de DF. Zulke methodes maken bijvoorbeeld gebruik van de hier niet verder "sharp" functie [15] of consensus [15], [11].

4.2 Vinden van een cover

In de vorige paragraaf is aangegeven wat een prime implicant is en hoe PI(f), de set van alle prime implicants van f, geconstrueerd kan worden. Zoals reeds eerder vermeld is, bestaat ([8], [38]) een minimale expressie uit prime implicants. De conjunctie van alle prime implicants realiseert de gewenste functie. Niet iedere prime implicant hoeft echter in de expressie voor te komen. Ook met een subset van alle prime implicants is f te realiseren. Deze paragraaf geeft in het kort aan hoe men uit de set PI(f) een cover i.e. een aantal prime implicants die samen een minimale expressie leveren, selecteert. Het cover wordt ook wel "minimal cover" genoemd omdat het bestaat uit een zo klein mogelijk aantal prime implicants dat een goedkoopste oplossing levert.
Een aantal methodes voor het bepalen van een minimal cover werkt met tabellen. In zulke tabellen is er een kolom voor iedere term uit de originele expressie in de DF. Voor elke (uit de originele expressie bepaalde) prime implicant is er een rij. Daarnaast is er een kolom waarin de kosten voor de prime implicants staan. Met behulp van deze tabel wordt een minimal cover gevormd. In de literatuur, bijvoorbeeld [15], [8] en [38], is beschreven hoe dit in zijn werk gaat. Voor functies van "veel" variabelen worden de tabellen al snel te groot om mee te werken. De door de tabel gebruikte geheugenruimte wordt te groot en de algoritmes worden te langzaam. Dit is vooral het geval als de originele expressie in de DNF staat. Naast de methodes die een tabel gebruiken zijn er die gebruik maken van de al eerder genoemde "sharp" functie. Voor een beschrijving wordt verwezen naar bijvoorbeeld [15].

Het vinden van een "goede" groep prime implicants (minimal cover) is meestal [38], [3] een NP compleet probleem. Dit is onafhankelijk van de methode die men gebruikt voor het vinden van een minimal cover. Voor problemen uit de klasse van NP complete

problemen heeft men tot nu toe geen algoritmes gevonden waarvan het aantal stappen (rekentijd) en/of de benodigde geheugenruimte, begrensd is door een polynoom.
Omdat het bepalen van een cover een NP compleet probleem is, is de ontwikkeling van heuristische algoritmes gerechtvaardigd. Deze leveren een (sub) optimaal cover in een aanvaardbare rekentijd. In dit verslag wordt niet verder ingegaan op de werking van zulke heuristische algoritmes.

Met de hierboven beschreven methodes kan men, uitgaande van de set van prime implicants, een cover van prime implicants vinden. De som (conjunctie) van deze prime implicants is een expressie in de DF. Deze expressie heeft minimale kosten, volgens de definitie van kosten in paragraaf 4.1, en kan direct omgezet worden in een Σ_2-circuit. Omdat de expressie minimale kosten heeft zal de complexiteit van het Σ_2-circuit in het algemeen zo klein mogelijk zijn. In de vorige uitspraak staat "in het algemeen". Dit is nodig omdat in de kosten niet alleen het aantal termen van de expressie maar ook het aantal literals per term tot uitdruking komt. Het kan dan voorkomen dat men kiest voor een grotere groep van prime implicants met minder literals.

4.3 Voordeel van monotone functies

In de vorige paragraaf is beschreven hoe men een minimale expressie in de DF kan construeren. Een van de onderdelen was het vinden van een cover uit de set van prime implicants. Dit bleek voor de meeste functies een zogenaamd NP compleet probleem te zijn. Deze paragraaf behandelt het covering probleem voor monotone en unate functies. Het zal blijken dat voor deze functies niet gezocht hoeft te worden naar een cover.
Voor het geval van monotone functies is dit eenvoudig in te zien. Hierbij zijn twee theorema's uit [38] gebruikt.

Theorema 2 [38]

Elke prime implicant van een monotone functie bevat alleen de literals x_i van de variabele x_i.

Bewijs

Gegeven $t=t'\bar{x}_i \in I(f)$. De term t is dus een implicant die een ontkenning van een variabele bevat. Om het theorema te bewijzen is het voldoende om aan te tonen dat $t' \in I(f)$. Als $t' \in I(f)$ dan geldt $t \notin PI(f)$ omdat t' een echte subterm van t is. Het bovenstaande kan herhaald worden voor iedere ontkenning van een variabele in t. Iedere implicant die eventueel prime implicant is, bevat dan alleen de literal x_i van de variabele x_i en niet de ontkenning $\bar{x}_i$.

Als $t'(a)=1$ dan is $a_i=0$ of $a_i=1$. In het eerste geval is $t'\bar{x}_i(a)=1$ en dus ook $f(a)=1$ omdat t een implicant was. In het tweede geval neemt men een b zodanig dat $b_i=1$ en $a_j=b_j$ voor alle $j \neq i$. Er geldt nu $t'x_i(b)=1$ en dus $f(b)=1$. Volgens de gebruikelijke ordening (zie hoofdstuk 3) is $a \leq b$. Omdat f monotoon is geldt dan $f(a) \leq f(b)$ en dus $f(b)=1$ omdat $f(a)=1$. In beide gevallen impliceert $t'(a)=1$ dat $f(a)=1$. Bijgevolg is $t' \in I(f)$.

Q.E.D.

Dit theorema is in overeenstemming met lemma 1 uit paragraaf 3.1.2.

Theorema 3 [38]

Voor een monotone functie f bestaat de unieke minimale expressie uit alle prime implicants.

Bewijs

De expressie moet iedere prime implicant bevatten. Om aan te tonen dat prime implicant t essentieel is, is het voldoende een input $a \in f^{-1}(1)$ te construeren met $t(a)=1$ en $t'(a)=0$ voor alle $t' \neq t$ met $t' \in PI(f)$.

Volgens theorema 2 mag men zonder meer stellen dat $t(x)=x_1 \ldots x_k$. Neem een a met $a_i=1$ voor alle $i \leq k$ en $a_i=0$ voor alle andere i. Natuurlijk is dan $t(a)=1$.
Stel nu dat er een $t' \in PI(f)$ is met $t' \neq t$ en $t'(a)=1$. Volgens theorema 2 en de opbouw van de input a kan t' dan alleen maar bestaan uit literals x_i met $i \leq k$. Maar dan is t' een subterm van t hetgeen in strijd is met het gegeven $t \in PI(f)$. De conclusie hieruit is dat voor deze a dus $t'(a)=0$ voor alle $t' \in PI(f)$ met $t' \neq t$.

Q.E.D.

Iedere prime implicant van een monotone functie is dus nodig in een minimale expressie in de DF. Er is geen enkele prime implicant die weggelaten kan worden uit het minimal cover. Zo'n prime implicant die altijd in een minimal cover moet zitten heet een essentiële prime implicant. Volgens theorema 3 is iedere prime implicant van een monotone functie essentieel.
Voor unate functies geldt ook dat iedere prime implicant essentieel is. Deze uitspraak wordt in dit verslag niet bewezen. Voor een bewijs wordt verwezen naar [3]. Een dergelijk bewijs zal echter niet zo sterk afwijken van het bewijs hierboven. Net als een monotone functie is een unate functie immers monotoon in al zijn variabelen. Een unate functie kan echter naast monotoon stijgend, ook monotoon dalend zijn in een aantal variabelen. Dit in tegenstelling tot een monotone functie die monotoon stijgend is in alle variabelen (zie paragraaf 3.1.1).
Het probleem van het vinden van een cover (zie paragraaf 4.2) bestaat niet voor unate en monotone functies. Iedere prime implicant is immers essentieel. Er hoeft dus geen keuze gemaakt te worden. Voor monotone en unate functies is het vinden van een minimal cover daarom geen NP compleet probleem. Brayton et al. beschrijven in [3] hoe het minimalisatie-programma EXPRESSO zijn voordeel doet met het feit dat prime implicants van unate en monotone functies essentieel zijn.

4.4 Multiple-output functies

In het voorgaande is besproken hoe de set van prime implicants bepaald wordt voor een functie $f \in B_n^*$ (paragraaf 4.1) en hoe uit deze set een minimal cover gevonden wordt (paragraaf 4.2). Hoe zit het nu met minimale expressies in de DF voor functies $f \in B_{n,m}^*$ waarbij m>1?
Het is mogelijk deze zogenaamde multiple-output functies te splitsen in m functies $f_1, f_2, \ldots, f_m$ uit de set B_n^*. Daarna wordt voor ieder van deze m functies een set $PI(f_i)$ en een minimal cover bepaald. Op deze manier worden m minimale expressies verkregen.
Zijn de totale kosten van deze m expressie nu ook minimaal? Deze vraag moet in de meeste gevallen ontkennend beantwoord worden. Minimale expressies voor $f \in B_{n,m}^*$ bevatten in de meeste gevallen ook prime implicants van functies die het produkt zijn van een groep functies uit $\{f_1, \ldots, f_m\}$. Friedman geeft in [8] aan hoe minimale expressies voor een functie uit de set $B_{n,m}^*$ gevonden kunnen worden. Als eerste worden alle zogenaamde multiple-output prime implicants gegenereerd. Dit zijn prime implicants van produktfuncties van een groep functies uit $\{f_1, \ldots, f_m\}$. Daarna wordt uit de set van alle multiple-output prime implicants een minimal cover bepaald voor elke f_i $(i=1,\ldots,m)$. Voor uitgebreidere informatie wordt naar [8] verwezen. Tot slot de opmerking dat het principe van de methodes voor het vinden van een minimale expressie niet verandert voor multiple-output functies. De algoritmes voor het genereren van multiple-output prime implicants en het vinden van minimal covers zijn [8] echter aanzienlijk complexer (meer rekentijd en geheugenruimte) in het geval van multiple-output functies.

5 Logisch ontwerp voor decoded programmable logic arrays

In het vorige hoofdstuk is beschreven hoe tijdens het logisch ontwerp optimale Σ_2-circuits bepaald worden. Een Σ_2-circuit kan fysisch met een two-level PLA gerealiseerd worden. Deze zijn om historische redenen, gering aantal lagen en daarom een hoge snelheid (zie ook hoofdstuk 2), interessant. Σ_2-circuits zijn een subklasse van de verzameling van alle mogelijke circuits. Een andere subklasse wordt gevormd door de Σ_3-circuits. Dit hoofdstuk richt zich niet op alle Σ_3-circuits maar op een speciale soort Σ_3-circuit. Bij deze speciale Σ_3-circuits zijn voorwaarden opgelegd aan de verbindingen in het eerste niveau met OR gates. Deze circuits vormen een model van zogenaamde decoded PLA's met losse decoders of met D veld (zie paragraaf 3.2.2) en zijn er fysisch mee te realiseren. Decoded PLA's nemen in het algemeen niet meer ruimte in dan de "normale" two-level PLA's.
Een Σ_3-circuit met voorwaarden voor de verbindingen in het eerste niveau wordt in dit verslag een beperkt (Σ_3-) circuit genoemd. Met een beperkt Σ_3-circuit kan een multiple output functie $f \in B^*_{n,m}$ berekend worden. Net als in het vorige hoofdstuk beperkt men zich tot functies $f \in B^*_n$ om het logisch ontwerp toe te lichten.
Dit hoofdstuk beschrijft hoe optimale beperkte Σ_3-circuits bepaald kunnen worden. Hiervoor zijn gegeneraliseerde Booleaanse expressies (GBE's) nodig. In paragraaf 5.1 worden deze GBE's geïntroduceerd. Paragraaf 5.2 geeft aan hoe een minimale GBE in de DF (minimaal aantal termen en minimaal aantal literals per term) gevonden kan worden. Daarna worden in paragraaf 5.3 bovengrenzen afgeleid voor de afmetingen van beperkte Σ_3-circuits. Daarbij wordt gebruik gemaakt van "multiple-valued decomposition" van een GBE. Een van de problemen die optreden bij het logisch ontwerp voor decoded PLA's is het "input variable assignment problem" i.e. vinden van de "optimale" grootte van de

decoders en de "optimale" verdeling van de variabelen over de decoders. Dit probleem wordt in paragraaf 5.4 behandeld. Tot slot behandeld paragraaf 5.5 het probleem van de "output phase optimization". Hierbij wordt gekeken of het in het geval van een multiple output functie, niet voordeliger is de ontkenning van een of meerdere uitgangen te berekenen.

5.1 Gegeneraliseerde Booleaanse functies

Zoals gezegd, is het uitgangspunt een binaire functie $f \in B_n^*$ van n binaire variabelen : $f(x_1,\ldots,x_n) : \{0,1\}^n \rightarrow \{0,1\}$. In plaats hiervan wordt ook wel geschreven $f(x_1,\ldots,x_n) : \overbrace{B \times B \times \ldots \times B}^{n} \rightarrow B$

waarbij B={0,1}. In dit verslag is de verzameling B altijd gelijk aan {0,1}. Deze functie kan gerepresenteerd worden met een Booleaanse expressie van binaire variabelen x_i (i=1,2,...,n). Hieronder wordt, in navolging van [27] het begrip gegeneraliseerde Booleaanse functie (Engels : generalized Boolean function) geïntroduceerd.
Men maakt met behulp van definitie 12 uit paragraaf 3.1.2 een partitie $(X_1,\ldots,X_r)$ van een variabele $X \in \{0,1\}^n$. De bovengenoemde functie f is nu te schrijven als :

$$f(X_1,\ldots,X_r) : B^{n_1} \times B^{n_2} \times \ldots \times B^{n_r} \rightarrow B.$$

Hierin is $n_i=d(X_i)$ voor i=1,2,...,r, het aantal binaire variabelen in X_i (zie definitie 12 in paragraaf 3.1.2). Let op het verschil in schrijfwijze tussen de binaire (twee-waardige) variabele x_i en de meer-waardige variabele X_i . Een functie in bovenstaande vorm wordt een gegeneraliseerde Booleaanse functie genoemd. Het begrip gegeneraliseerde Booleaanse functie wordt

afgekort tot GBF. Iedere GBF kan men representeren met een gegeneraliseerde Booleaanse expressie (afgekort tot GBE) van 2^{n_i}-waardige variabelen X_i. Volgens bovenstaande formule zijn er r van deze variabelen nodig.

Een 2^{n_i}-waardige variabele neemt een uit 2^{n_i} waardes aan. Er zijn twee gangbare methodes om de waarde aan te geven. De eerste maakt gebruik van "normale" getallen. Voor een meerwaardige variabele X_i geldt dan $X_i \in \{0,1,\ldots,2^{n_i}-1\}$. Deze schrijfwijze wordt bijvoorbeeld door Rudell et al. [26] gebruikt voor de beschrijving van PLA optimalisatie met behulp van multiple-valued minimalisatie. De tweede methode gebruikt 2^{n_i} verschillende rijtjes van enen en nullen. Dit zijn de rijtjes die verkregen worden door de waardes van de n_i binaire variabelen waaruit X_i is opgebouwd, achter elkaar te zetten. Deze methode wordt bijvoorbeeld door Sasao in [27] gebruikt. In dit verslag wordt de tweede methode gebruikt. Ze sluit beter aan bij de definitie van een literal in definitie 6 (paragraaf 3.1.2) en draagt bij tot een efficiënte implementatie van algoritmes die met literals werken.

Hoe ziet een expressie (GBE) van een GBF (gegeneraliseerde Booleaanse functie) eruit? Dit kan men het beste toelichten aan de hand van een voorbeeld. In dit voorbeeld maakt men gebruik van de literals volgens definitie 6 (paragraaf 3.1.2).

Voorbeeld 11 [27] [31] ([6])
Gegeven een functie f van vier binaire variabelen variabelen. De functiewaardetabel ziet er als volgt uit :

Tabel 1 : Functiewaardetabel voor f

x_1	x_2	x_3	x_4	f	x_1	x_2	x_3	x_4	f
0	0	0	0	1	1	0	0	0	1
0	0	0	1	1	1	0	0	1	0
0	0	1	0	1	1	0	1	0	0
0	0	1	1	0	1	0	1	1	1
0	1	0	0	1	1	1	0	0	0
0	1	0	1	1	1	1	0	1	1
0	1	1	0	0	1	1	1	0	1
0	1	1	1	1	1	1	1	1	0

De som van mintermen expressie voor f is :

$$f(x_1,x_2,x_3,x_4) = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4+\bar{x}_1\bar{x}_2\bar{x}_3x_4+\bar{x}_1\bar{x}_2x_3\bar{x}_4+\bar{x}_1x_2\bar{x}_3\bar{x}_4+\bar{x}_1x_2\bar{x}_3x_4+$$
$$\bar{x}_1x_2x_3x_4+x_1\bar{x}_2\bar{x}_3\bar{x}_4+x_1\bar{x}_2x_3x_4+x_1x_2\bar{x}_3x_4+x_1x_2x_3\bar{x}_4.$$

Dit is de vorm die ook in het vorige hoofdstuk gebruikt is. Volgens voorbeeld 1 bij definitie 6 in paragraaf 3.1.2 was x een andere notatie voor X^1 en $\bar{x}$ voor X^0. Als de originele schrijfwijze van definitie 6 gebruikt wordt, ziet de expressie er als volgt uit :

$$f(X_1,X_2,X_3,X_4) = X_1^0X_2^0X_3^0X_4^0+X_1^0X_2^0X_3^0X_4^1+X_1^0X_2^0X_3^1X_4^0+X_1^0X_2^1X_3^0X_4^0+X_1^0X_2^1X_3^0X_4^1+$$
$$X_1^0X_2^1X_3^1X_4^1+X_1^1X_2^0X_3^0X_4^0+X_1^1X_2^0X_3^1X_4^1+X_1^1X_2^1X_3^0X_4^1+X_1^1X_2^1X_3^1X_4^0.$$

In termen van definitie 12 (paragraaf 3.1.2) is hier de triviale partitie (X_1,X_2,X_3,X_4) van $X=(x_1,x_2,x_3,x_4)$ gebruikt met $X_1=(x_1)$, $X_2=(x_2)$, $X_3=(x_3)$ en $X_4=(x_4)$. Bovenstaande expressie is een zogenaamde gegeneraliseerde Booleaanse expressie (GBE) i.e. een expressie die een GBF representeert. De "gewone", tot nu toe

gebruikte Booleaanse expressie is een speciaal geval van de GBE. Een andere partitie is (X_5,X_6) met $X_5=(x_1,x_2)$ en $X_6=(x_3,x_4)$. X_5 en X_6 zijn nu variabelen in $\{0,1\}^2$, in tegenstelling tot X_1 tot en met X_4 hierboven die variabelen in $\{0,1\}$ waren.
De functie f kan men dan ook met de volgende gegeneraliseerde Booleaanse expressie representeren :

$$f(X_5,X_6)=X_5^{(00)}X_6^{(00)}+X_5^{(00)}X_6^{(01)}+X_5^{(00)}X_6^{(10)}+X_5^{(01)}X_6^{(00)}+X_5^{(01)}X_6^{(01)}+$$
$$X_5^{(01)}X_6^{(11)}+X_5^{(10)}X_6^{(00)}+X_5^{(10)}X_6^{(11)}+X_5^{(11)}X_6^{(01)}+X_5^{(11)}X_6^{(10)}.$$

In deze expressie is (00) geschreven in plaats van (0,0) teneinde een compactere schrijfwijze te verkrijgen.
In de vorige expressies staat in de "exponent" van de literal steeds maar een constante waarde. Volgens definitie 6 is het ook mogelijk hier een verzameling neer te zetten. De volgende expressie is een minimale expressie voor de functie f waarbij weer de partitie (X_5,X_6) is gebruikt. In deze expressie komen in de exponent wel verzamelingen van constanten voor.

$$f(X_5,X_6)=X_5^{\{00,01\}}X_6^{\{00,01\}}+X_5^{\{00,11\}}X_6^{\{01,10\}}+X_5^{\{01,10\}}X_6^{\{00,11\}}.$$

Ook hier is weer een verkorte schrijfwijze gebruikt. {00,01} bijvoorbeeld, staat voor {(00),(01)}, hetgeen weer een afkorting was voor {(0,0),(0,1)}. Een minimale expressie voor de partitie (X_1,X_2,X_3,X_4) is :

$$f(X_1,X_2,X_3,X_4)=X_1^0X_2^{\{0,1\}}X_3^0X_4^{\{0,1\}}+X_1^0X_2^0X_3^{\{0,1\}}X_4^0+X_1^{\{0,1\}}X_2^0X_3^0X_4^0+$$
$$X_1^0X_2^1X_3^{\{0,1\}}X_4^1+X_1^{\{0,1\}}X_2^1X_3^0X_4^0+X_1^1X_2^0X_3^1X_4^1+X_1^1X_2^1X_3^1X_4^0.$$

Merk op dat de literal $X_2^{\{0,1\}}$ altijd de waarde 1 levert en dus uit de expressie verwijderd kan worden.
(einde voorbeeld)

In het bovenstaande uitgebreide voorbeeld heeft men kennis gemaakt met de gegeneraliseerde Booleaanse expressie (GBE). Het uiterlijk van de expressie hangt af van de gekozen partitie. In het voorbeeld zijn twee partities gebruikt. Voor elke partitie is een minimale GBE gegeven. Uit het voorbeeld blijkt duidelijk dat het aantal termen in de minimale expressie afhangt van de gekozen partitie. Hoe men een minimale expressie bepaalt en hoe men een "goede" partitie kiest, wordt in een van de volgende paragrafen behandeld.

De algemene vorm van een gegeneraliseerde Booleaanse expressie (GBE) in de DF voor een gegeneraliseerde Booleaanse functie (GBF) is [27] :

$$f(X_1,\ldots,X_r) = \bigvee_{(S_1,\ldots,S_r)} X_1^{S_1} X_2^{S_2} \ldots X_r^{S_r} \ .$$

Hierin is $(X_1,\ldots,X_r)$ een partitie van X. Iedere willekeurige functie f kan [27] in deze vorm geschreven worden. Als P een minimale expressie in de DF is voor een functie f, dan geldt voor t(P) i.e. het aantal termen in P :

$$t(P) \leq 2^{n-\max\{n_i\}} \quad \text{met } n_i = d(X_i) \ .$$

In het voorgaande zijn GBF's en GBE's ingevoerd. Wat is het nut hiervan? Het blijkt mogelijk te zijn [27] een GBE direct om te zetten in een beperkt Σ_3-circuit. Er is een één op één afbeelding mogelijk tussen een GBE in de DF en een beperkt Σ_3-circuit, net als tussen normale expressies (of GBE's met een triviale partitie) en Σ_2-circuits.
Hieronder is aangegeven hoe de één op één afbeelding tot stand komt. In een gewone Booleaanse expressie zijn de literals gelijk aan de ingangsvariabelen $x_1,\ldots,x_n$ en/of hun ontkenningen. Bij een GBE is dit niet zo behalve in het geval van de triviale

partitie. De literals $X_i^{S_i}$ moeten op een of andere manier berekend worden uit de ingangen $x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n$. De uitdrukking voor een literal X^S uit definitie 6 (paragraaf 3.1.2) kan met behulp van de rekenregels voor literals geschreven worden als :

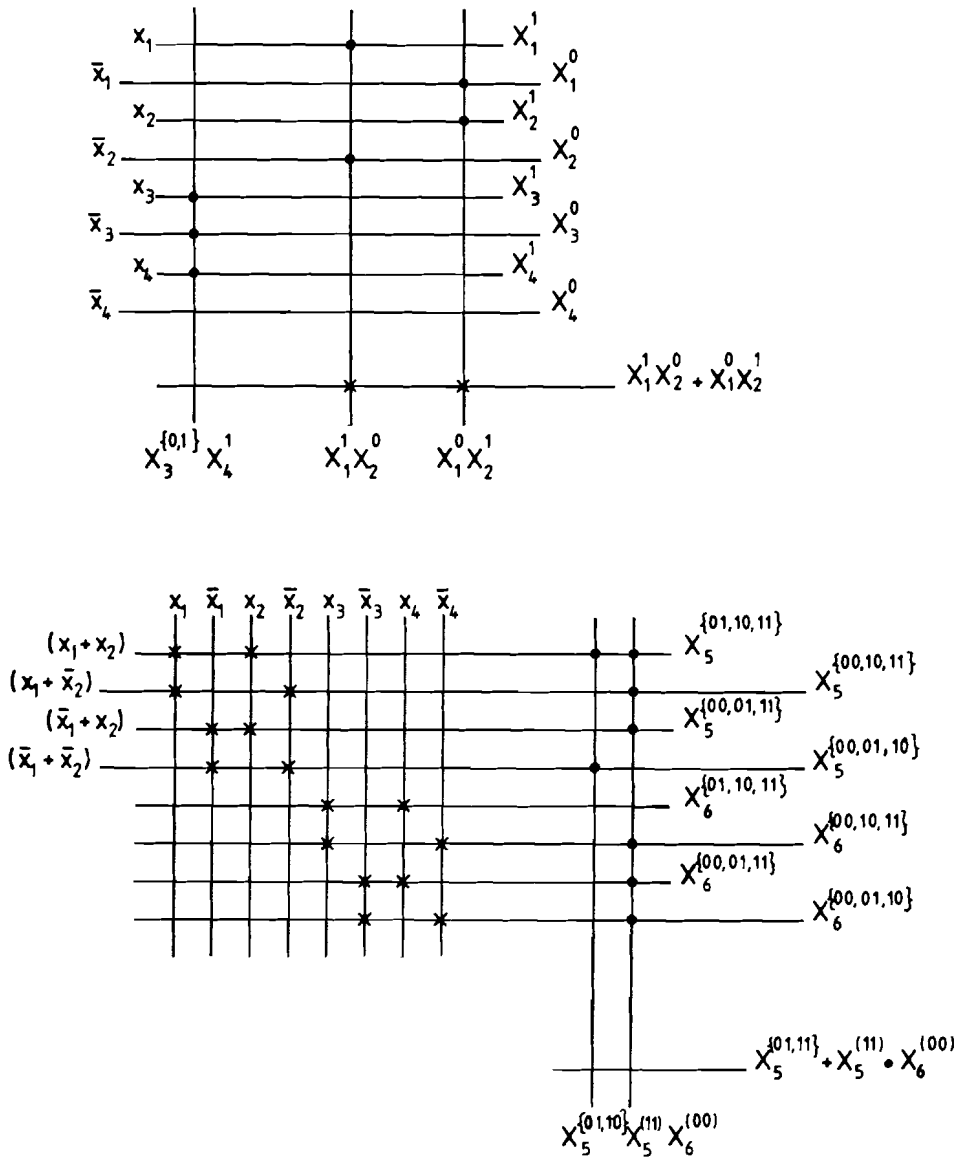
$$X^S = \overline{\overline{X^S}} = \overline{\overline{\bigvee_{a \in S} X^a}} = \overline{\bigvee_{a \in (B^n - S)} X^a} = \bigwedge_{a \in (B^n - S)} \overline{X^a}$$

Hierin is $\overline{X^a}$ een maxterm van n binaire variabelen. In bovenstaande formule kan zonder meer de index i worden toegevoegd. De maxtermen kunnen in het eerste OR (niveau) veld van een beperkt Σ_3-circuit berekend worden. Een literal $X_i^{S_i}$ is dus samengesteld uit een aantal (ten hoogste 2^{n_i}) maxtermen die afhankelijk zijn van precies n_i binaire variabelen.
Een andere dan een van deze n_i variabelen is niet nodig bij de berekening van de maxtermen ten behoeve van deze literal. Daarom zal een (groot) aantal van alle mogelijke verbindingen in het eerste OR veld nooit gebruikt worden. Het aantal gebruikte verbindingen is dus beperkt. Dit is de reden waarom van een beperkt Σ_3-circuit wordt gesproken.

De maxtermen voor de literal $X_i^{S_i}$ worden met een gate (in een kolom) van het AND veld van het beperkte Σ_3-circuit samengevoegd. Op dezelfde gate kunnen de maxtermen voor alle andere literals aangesloten worden. Per term in de formule voor de algemene vorm van een GBE (zie boven) is nu precies een AND gate nodig. Al deze AND gates zijn verbonden met een OR gate in het tweede OR veld. Aan de uitgang van deze gate staat de gewenste functie f. Per functie is een OR gate in het tweede OR veld nodig.

In figuur 10 zijn, ter verduidelijking, voorbeelden van een Σ_2-

circuit en een beperkt Σ_3-circuit gegeven. De circuits berekenen een functie f van vier variabelen. In deze figuur zijn de gates op een andere manier getekend dan in voorgaande figuren. De lijnen zijn de gates, en een bolletje of kruisje op een kruising geeft aan dat de ene lijn een ingang is van de andere gate (lijn). Een kruisje representeert een verbinding met een OR



Figuur 10 : Σ_2-circuit (boven) en Σ_3-circuit (onder)

gate een bolletje een verbinding met een AND gate. Voor de duidelijkheid is de partitie (X_5,X_6) genoemd. Natuurlijk had ze ook (X_1,X_2) kunnen heten. De partitie verdeelt de variabelen in twee groepen van ieder twee variabelen.
In het eerste OR veld worden alle mogelijke maxtermen gevormd die bij de gegeven partitie mogelijk zijn. In figuur 10 zijn dit

$X_5^{\{01,10,11\}}$, $X_5^{\{00,10,11\}}$, $X_5^{\{00,01,11\}}$ en $X_5^{\{00,01,10\}}$ voor X_5 en

$X_6^{\{01,10,11\}}$, $X_6^{\{00,10,11\}}$, $X_6^{\{00,01,11\}}$ en $X_6^{\{00,01,10\}}$ voor X_6. In

het AND veld is het mogelijk uit deze maxtermen elk gewenst produkt van literals te vormen. In de "exponent" is elke gewenst set te verkrijgen. Dit gaat op dezelfde manier als in het AND veld van het Σ_2-circuit. Een zaak die opvalt in vergelijk met Σ_2-circuits is de volgende. Is in een produkt (term) uit het AND veld bijvoorbeeld $X_5^{(00)}$ gewenst dan zijn er drie ingangen op de

AND gate nodig ($X_5^{\{00,10,11\}}$, $X_5^{\{00,01,11\}}$ en $X_5^{\{00,01,10\}}$). Als in

een term van een expressie de literal $X_5^{\{00,11\}}$ voorkomt dan zijn

er twee ingangen op de AND gate nodig namelijk $X_5^{\{00,01,11\}}$ en

$X_5^{\{00,10,11\}}$. Dit komt omdat in het OR veld maxtermen gevormd worden. In een produkt maken deze een uitgang selectief laag. Dit in tegenstelling tot mintermen die een uitgang selectief hoog maken.

In deze paragraaf zijn GBE's geïntroduceerd. Een GBE in de DF met literals volgens definitie 6 en een partitie volgens definitie 12 uit paragraaf 3.1.2, kan direct omgezet worden in een beperkt Σ_3-circuit. Zo'n GBE heeft dus dezelfde eigenschap als expressies in de DF met de triviale partitie die direct in Σ_2-

circuits om te zetten zijn. Om efficiënte beperkte Σ_3-circuits te ontwerpen, worden de GBE's getransformeerd. In de volgende paragraaf wordt hierop nader ingegaan.

5.2 Vinden van een minimale gegeneraliseerde Booleaanse expressie

In dit hoofdstuk is het begrip gegeneraliseerde Booleaanse expressie (GBE) ingevoerd. Zo'n expressie is een uitbreiding van de expressies in het vorige hoofdstuk. Net als in het vorige hoofdstuk is men geïnteresseerd in het vinden van een minimale GBE in de DF. Zo'n minimale GBE heeft een zo klein mogelijk aantal termen en zo min mogelijk literals per term (zie hoofdstuk 2 en 4). Een minimale GBE levert een beperkt Σ_3-circuit met een minimale complexiteit. Dit is dezelfde redenering als die in hoofdstuk 4 gebruikt is voor minimale expressies en Σ_2-circuits met een minimale complexiteit.
Hoe wordt een minimale GBE van een gegeven GBE bepaald? In het voorafgaande is al aangegeven dat een GBE gezien kan worden als een functie van meerwaardige (multiple-valued) variabelen. Minimalisatie van de expressie kan dan ook geschieden met behulp van de theorie van "multiple-valued logic". Het minimalisatie-programma MINI gebruikt deze theorie om heuristisch [11], [3] een minimale GBE te vinden. Ook absolute minimalisatie is [11] mogelijk met behulp van de multiple-valued logic theorie [31], [27] en [34]. Hierbij moet aangetekend worden dat de partitie gegeven moet zijn. De uitdrukkingen die bij multiple-valued logic gebruikt worden zijn [11] in het algemeen complex. Bovendien wijken ze af van de uitdrukkingen die bij methodes voor minimalisatie van normale Booleaanse expressies (zie hoofdstuk 4) gebruikt worden. Daarom wordt minimalisatie met behulp van de theorie van multiple-valued logic in dit verslag niet behandeld.
Men kan een GBE ook minimaliseren door de theorie uit hoofdstuk 4 uit te breiden. De theorie uit hoofdstuk 4 wordt dan een speciaal

geval van deze theorie. Hieronder is in het kort aangegeven hoe deze theorie eruit ziet. Voor een volledige beschrijving wordt verwezen naar de literatuur, in het bijzonder naar [11]. In dit verslag wordt de beschrijving van [11] gebruikt.

Uitgangspunt is een niet triviale partitie $(X_1,\ldots,X_r)$ volgens definitie 12 uit paragraaf 3.1.2 van n binaire variabelen. Voor de trivale partitie worden de resultaten van hoofdstuk 4 verkregen.
Een hyperterm is een produkt van literals. Het voorvoegsel "hyper" is gebruikt om aan te geven dat het niet om de triviale partitie gaat. Een "entailant" van een functie f is een hyperterm h met de eigenschap :

Voor alle $a \in B^n$ met $h(a)=1$ geldt $f(a)=1$.

Men zegt ook wel dat de entailant de functie impliceert.
Een "prime entailant" is een entailant die geen enkele andere entailant impliceert. Prime entailant en entailant zijn de uitbreidingen van respectievelijk prime implicant en implicant uit hoofdstuk 4. Voor de triviale partitie gaat een (prime) entailant over in een (prime) implicant. De conjuctie (som) van alle prime implicants is een representatie voor de functie f. Analoog is een zogenaamde "complete hypersum" i.e. de conjunctie van alle prime entailants, een representatie voor f. Voor een minimale hypersum zijn meestal niet alle prime entailants nodig. Een minimale hypersum is een hypersum van een zo klein mogelijk aantal prime entailants zodanig dat de hypersum een representatie is van de functie f.

De Karnaugh- en Quine-McCluskey methode zijn, zie hoofdstuk 4, methodes waarmee men alle prime implicants kan vinden. De Karnaugh methode is niet geschikt [11] om prime entailants te vinden. Het blijkt dat de mintermen van de prime entailants niet altijd een rechthoek vormen in de grafische Karnaugh diagrammen. De prime entailants kunnen wel gevonden worden met een Quine-McCluskey achtige methode. Hierin maakt men gebruik van de

begrippen "hypoconsensus" en "hyperconsensus".
Een hyperterm h is een hypoconsensus van twee hypertermen h1 en h2 als aan de volgende drie eigenschappen voldaaan is :
- h impliceert h1+h2 .
- $h.\overline{h1} \neq 0$.
- $h.\overline{h2} \neq 0$.

Een hypoconsensus h van h1 en h2 is een hyperconsensus als h geen van de andere hypoconsensi van h1 en h2 impliceert.
Een som van hypertermen expressie g voor een functie f is [11] een complete hypersum van f dan en slechts dan als :

1) Geen enkele hyperterm van g een andere impliceert.
2) Voor ieder paar hypertermen er geen hyperconsensi bestaan of ieder van deze hyperconsensi impliceert een andere hyperterm in g.

Hiermee kan de volgende procedure opgesteld worden voor het genereren van een complete hypersum expressie voor een functie f. Startpunt is een willekeurige som van hypertermen expressie in de DF voor f.
Procedure [11]

1) Verwijder iedere hyperterm in de huidige expressie die een andere hyperterm impliceert.
2) Kijk of alle paren van hypertermen voldoen aan bovenstaande conditie 2).
   Zo ja, stop.
   Zo nee, neem dan een paar dat niet aan de conditie voldoet.
3) Genereer alle hyperconsensi voor dit paar. (Hoe dit kan is in [11] beschreven.)
4) Voeg de gevonden hyperconsensi toe aan de expressie en ga verder met stap 1).

Er is [11] nog een andere procedure om een complete hypersum te bepalen. Men ordent hiervoor eerst de hypertermen per partitie-element X_i. Het is dan mogelijk een lattice op te stellen. In de procedure worden dan de partitie-elementen een voor een aflopen.

Ieder partitie-element hoeft slechts een keer bekeken te worden. Als alle partitie-elementen behandeld zijn, stopt de procedure. Voor ieder partitie-element worden alle bestaande hyperconsensi met betrekking tot het gekozen partitie-element, bepaald voor ieder paar van hypertermen in de expressie. Deze hyperconsensi worden toegevoegd aan de expressie. Daarna worden hypertermen die een andere impliceren, verwijderd. Vervolgens wordt verder gegaan met het volgende partitie-element.
Deze laatste procedure om een complete hypersum te bepalen, komt [11] overeen met de methode van Tison [36] om alle prime implicants te bepalen. Deze laatste procedure is in het algemeen efficiënter dan de eerste procedure.

Als de complete hypersum berekend is, kan een minimale hypersum bepaald worden. Alle prime entailants worden daartoe bijvoorbeeld in een tabel gezet. Het is hier mogelijk kosten toe te kennen aan iedere prime entailant. Met behulp van deze tabel bepaalt men vervolgens een minimal cover. Dit gaat geheel analoog aan de in paragraaf 4.2 beschreven methode om een minimal cover uit een prime implicants tabel te vinden. Ook de andere in paragraaf 4.2 genoemde methodes kunnen worden gebruikt voor het oplossen van het covering problem.

Bij de bovenstaande minimalisatie methode, zoeken van prime entailants en vinden van een minimal cover, zijn de kosten-factoren "aantal gates" en "aantal ingangen per gate" gebruikt. Dit leidt tot een zo klein mogelijk aantal termen (prime entailants) en zo min mogelijk literals per prime entailant.
Zowel in hoofdstuk 4 als hier wordt steeds de hele prime implicants cq. prime entailants set bepaald. Het aantal prime entailants is een maat voor de complexiteit van het algoritme dat een minimale expressie bepaald. Sasao en Terada hebben [31] metingen en berekeningen uitgevoerd naar het (gemiddelde) aantal prime entailants van functies van n, p-waardige variabelen. Het voert hier te ver om de berekeningen en metingen te beschrijven.

Wel zullen de belangrijkste resultaten gepresenteerd worden.
Er zijn boven- en ondergrenzen af te leiden voor het aantal prime entailants van functies van n, p-waardige variabelen. Alhoewel er functies bestaan die het maximale aantal prime entailants hebben, ligt het aantal prime entailants van de meeste functies ver onder dit maximum.
Veel interessanter zijn de gegevens over het gemiddelde aantal prime entailants. Laat $G_p(n,u)$ het gemiddelde aantal prime entailants zijn van een functie van n variabelen waarbij iedere variabele p-waardig is. Het getal u is de fractie van het aantal enen in de functiewaardetabel van de functie. Als functie van u stijgt $G_p(n,u)$ eerst monotoon totdat u een bepaalde waarde bereikt. Vanaf die waarde van u daalt $G_p(n,u)$ monotoon. Er is dus een maximum.
Ook blijkt dat $G_4(n/2,u) \geq G_2(n,u)$ als n en u voldoende groot zijn. Dit betekent dat als men partities $(X_1,\ldots,X_r)$ met $d(X_i)=2$ gebruikt, het aantal prime entailants niet kleiner is dan in het geval van de triviale partitie. Het laatste wordt ook bevestigd door experimenten ([31]).
De experimenten beperken zich tot het vergelijken van functies met vijf, vier-waardige variabelen en functies met tien, twee-waardige variabelen. Het aantal prime entailants in een minimale expressie neemt af naarmate u stijgt. Het gemiddelde aantal prime entailants in minimale expressies voor functies van vier-waardige variabelen is iets kleiner dan dat van functies van twee-waardige variabelen (voor dezelfde waarde van u). Wat opvalt is dat het aantal essentiële prime entailants voor functies van vier-waardige variabelen veel kleiner is dan voor functies van twee-waardige variabelen (voor $0,2 \leq u \leq 0,5$).
Bij vier-waardige variabelen is het aantal prime entailants groter maar het aantal essentiële prime entailants is kleiner dan bij twee-waardige variabelen.
Sasao en Terada concluderen in [31], mede op grond van het bovenstaande, dat de klassieke minimalisatie methodes (bijvoorbeeld de methode van Quine-McCluskey) niet geschikt zijn

voor het oplossen van grote minimalisatie problemen. Grote problemen zijn functies van tien of meer variabelen. Er moet daarom gezocht worden naar een heuristisch algoritme dat relatief snel een bijna optimale oplossing levert.
Sasao presenteert in [29] een algoritme dat alle **essentiële** prime entailants detecteert zonder eerst alle prime **entailants** te vinden. Het algoritme kan echter geen essentiële prime entailants detecteren die ontstaan na verwijdering van zogenaamde kolom- of rij-dominantie. (Zie [8] voor een uitleg van kolom- en rijdominantie.)
Algoritmes die een expressie minimaliseren zonder eerst de hele prime entailants tabel te vinden zijn veel sneller dan andere die wel de hele tabel vormen. In het geval de triviale partitie gebruikt wordt, is het nog niet bekend [38] of men een absoluut minimale expressie kan vinden zonder alle prime implicants te bepalen.

5.3 Afmetingen van beperkte Σ_3-circuits

In het voorgaande is besproken hoe een gegeneraliseerde Booleaanse expressis (GBE) geminimaliseerd kan worden. De vraag rijst echter of het van te voren te zeggen is hoeveel termen een expressie voor een bepaalde functie zal bevatten. Het aantal termen is een directe maat voor het aantal gates i.e. de complexiteit van het AND veld in het beperkte Σ_3-circuit. In deze paragraaf worden boven- en ondergrenzen afgeleid voor het aantal termen in een GBE en de complexiteit van beperkte Σ_3-circuits voor verschillende soorten functies. Hierbij wordt gebruik gemaakt van "multiple-valued decomposition" [27] en [31]. De resultaten van de decompositie kunnen ook gebruikt worden voor het oplossen van het, in paragraaf 5.4 besproken, probleem van de input variable assignment. Vanwege het belang van de decompositie is deze in paragraaf 5.3.1 vrij uitvoerig beschreven. Daarna

wordt in paragraaf 5.3.2 een grens bepaald voor het nodige en voldoende aantal termen in een expressie. Met behulp van deze bovengrens voor het aantal termen in een expressie, wordt in paragraaf 5.3.3 de complexiteit van beperkte Σ_3-circuits bepaald. Tot slot staan in paragraaf 5.3.4 enige opmerkingen over de invloed van de grootte van de partitie-elementen op de boven- en ondergrenzen voor de complexiteit.

5.3.1 Multiple-valued decomposition

Deze paragraaf belicht de multiple-valued decomposition [27], [31] van gegeneraliseerde Booleaanse functies.
Uitgangspunten zijn een partitie $(X_1,\ldots,X_r)$ van de variabele $X=(x_1,\ldots,x_n)$ en een functie $f(X) : B^{n_1} \times B^{n_2} \times \ldots \times B^{n_r} \rightarrow B$.

Gegeven een $a,b,c \in B^{n_i}$. De notatie $f(X|X_i=a)$ staat voor : $f(X_1,X_2,\ldots,X_{i-1},a,X_{i+1},\ldots,X_r)$.

De relatie $\tilde{i}$ is gedefinieerd als $a \tilde{i} b \iff f(X|X_i=a)=f(X|X_i=b)$. Zo'n relatie is een equivalentie relatie. Als $a \tilde{i} b$ dan geldt ook $b \tilde{i} a$. Als $a \tilde{i} b$ en $a \tilde{i} c$ dan is ook $b \tilde{i} c$. Voor elk paar a,b kan bekeken worden of a in relatie staat met b. Daarna wordt een partitie van alle elementen in B^{n_i} gevormd. Deze partitie heet

$$\pi_i=(L_0^i,L_1^i,\ldots,L_{k_i-1}^i).$$

De partitie π_i is zo gemaakt dat $a,b \in B^{n_i}$ beide in L_j^i $(j=0,\ldots,k_i-1)$ zitten dan en slechts dan als $a \tilde{i} b$.

De L_j^i zijn de partitie-elementen. De i in de "exponent" geeft aan dat men te maken heeft met de variabele X_i. Op dit moment

zijn er dus twee soorten partities. De eerste partitioneert de binaire variabelen. De tweede partitioneert alle mogelijke ingangscombinaties van binaire variabelen in elk partitie-element van de eerste partitie.

Het getal k_i is gelijk aan het aantal groepen (partitie-elementen) in de partitie π_i. Het aantal groepen is tenminste 1 en maximaal 2^{n_i}. In het eerste geval staan alle elementen uit B^{n_i} met elkaar in relatie. In het laatste geval is er geen enkel tweetal elementen uit B^{n_i} dat met elkaar in relatie staat. Voor k_i geldt nu : $1 \leq k_i \leq 2^{n_i}$.
Er wordt nu een verzameling M_i gedefinieerd als zijnde $M_i=\{0,1,\ldots,k_i-1\}$. Deze verzameling wordt straks gebruikt om een partitie functie te definiëren. De elementen uit M_i worden daarbij gebruikt om de groepen van partitie π_i te nummeren.
Alvorens de partitie functie te definiëren, wordt in voorbeeld 12 het begrip partitie verduidelijkt.

Voorbeeld 12 [27]
Gegeven een functie f

$$f(X)=(\bar{x}_1+\bar{x}_2)(x_3 \oplus x_4)(\bar{x}_5+\bar{x}_6)+(x_1+x_2)(x_3 \oplus \bar{x}_4)x_5+(x_1 \oplus x_2)(x_5 \oplus x_6).$$

Uitgangspunt is de partitie (X_1,X_2,X_3) met $X_1=(x_1,x_2)$, $X_2=(x_3,x_4)$ en $X_3=(x_5,x_6)$. Deze partitie is willekeurig gekozen.
Voor deze functie geldt :

$f(X|X_1=(01))=f(X|X_1=(10))$
$f(X|X_2=(00))=f(X|X_2=(11))$
$f(X|X_2=(01))=f(X|X_2=(10))$.

De partities voor de equivalentie relaties $\tilde{i}$ (i=1,2,3) zijn dan

π_1=([00],[01,10],[11])

π_2=([00,11],[01,10])

π_3=([00],[01],[10],[11]).

De tweede partitie geeft de informatie dat voor X_2=(00) en X_2=(11) de functie dezelfde functiewaarde heeft voor alle waardes van X_1 en X_3. Voor X_2=(01) en X_2=(10) geldt hetzelfde. De eerste en derde partitie geven soortgelijke informatie.

Ter verduidelijking zijn de vier mogelijkheden voor X_1 uitgeschreven :

$f(X|X_1=(00))=(x_3 \oplus x_4)(\bar{x}_5+\bar{x}_6)$

$f(X|X_1=(01))=(x_3 \oplus x_4)(\bar{x}_5+\bar{x}_6)+(x_3 \oplus \bar{x}_4)x_5+(x_5 \oplus x_6)$

$f(X|X_1=(10))=(x_3 \oplus x_4)(\bar{x}_5+\bar{x}_6)+(x_3 \oplus \bar{x}_4)x_5+(x_5 \oplus x_6)$

$f(X|X_1=(11))=(x_3 \oplus \bar{x}_4)x_5$.

Hierin is duidelijk te zien dat alleen de expressies $f(X|X_1=(01))$ en $f(X|X_1=(10))$ aan elkaar gelijk zijn. In de eerste partitie moeten (01) en (10) dan ook bij elkaar geplaatst worden.

(einde voorbeeld)

Een partitie π_i bestaat uit k_i partitie-elementen. Deze zijn genummerd van 0 tot en met k_i-1. De partitie functie Ψ_i geeft voor elke $a \in B^{n_i}$ het nummer van het partitie-element waarin a zit. Voor de functie Ψ_i : $B^{n_i} \longrightarrow M_i$ geldt dat $\Psi_i(a)=j$ dan en slechts dan als $a \in L^i_j$. De functie Ψ_i wordt in het volgende lemma gebruikt om een functie g te definiëren.

Lemma 3 [27]

Er bestaat een functie g : $M_1 \times M_2 \times \ldots \times M_r \rightarrow B$ zodanig dat

$f(X_1,X_2,\ldots,X_r)=g(\Psi_1(X_1),\ldots,\Psi_r(X_r))$ waarbij $1 \leq |M_i| = k_i \leq 2^{n_i}$.

De functie g is een meerwaardige ingangs-, tweewaardige uitgangsfunctie. De functie g lijkt veel op een GBF. Het verschil is echter dat het aantal elementen in de verzamelingen M_i in het algemeen geen macht van 2 is. De functie g is ook weer te geven met een expressie [27]. De algemene expressie voor g is

$$g(Y_1,\ldots,Y_r) = \bigvee_{(T_1,\ldots,T_r)} Y_1^{T_1} Y_2^{T_2} \ldots Y_r^{T_r} .$$

De verzamelingen T_i zijn een deelverzameling van de M_i. De $Y_i^{T_i}$ zijn de literals in deze expressie. Ze zijn gedefinieerd op een manier analoog aan definitie 6. Het domein B^n van de afbeelding aldaar moet echter vervangen worden door M.

Voorbeeld 13 [27]

De partitie functies van het vorige voorbeeld 12 staan in tabel 2.

Tabel 2 : Partitie functies voor de functie f uit voorbeeld 12

X_i	$\Psi_1(X_1)$	$\Psi_2(X_2)$	$\Psi_3(X_3)$
00	0	0	0
01	1	1	1
10	1	1	2
11	2	0	3

Een andere partitie van $X=(x_1,\ldots,x_6)$ zal in het algemeen leiden tot andere partitie functies. De functiewaardetabel voor de functie g volgens lemma 3 staat in de volgende tabel.

Tabel 3 : Functiewaardetabel voor de functie g

Y_1	Y_2	Y_3	g	Y_1	Y_2	Y_3	g	Y_1	Y_2	Y_3	g
0	0	0	0	1	0	0	0	2	0	0	0
0	0	1	0	1	0	1	1	2	0	1	0
0	0	2	0	1	0	2	1	2	0	2	1
0	0	3	0	1	0	3	1	2	0	3	1
0	1	0	1	1	1	0	1	2	1	0	0
0	1	1	1	1	1	1	1	2	1	1	0
0	1	2	1	1	1	2	1	2	1	2	0
0	1	3	0	1	1	3	0	2	1	3	0

Een expressie voor g is :

$$g(Y_1,Y_2,Y_3)=Y_1^0Y_2^1Y_3^0+Y_1^0Y_2^1Y_3^1+Y_1^0Y_2^1Y_3^2+$$
$$Y_1^1Y_2^0Y_3^1+Y_1^1Y_2^0Y_3^2+Y_1^1Y_2^0Y_3^3+Y_1^1Y_2^1Y_3^0+$$
$$Y_1^1Y_2^1Y_3^1+Y_1^1Y_2^1Y_3^2+Y_1^2Y_2^0Y_3^2+Y_1^2Y_2^0Y_3^3.$$

Net als bij voorgaande expressies kan ook deze expressie vereenvoudigd worden tot :

$$g(Y_1,Y_2,Y_3)=Y_1^{\{0,1\}}Y_2^1Y_3^{\{0,1,2\}}+Y_1^{\{1,2\}}Y_2^0Y_3^{\{2,3\}}+Y_1^1Y_3^{\{1,2\}}.$$

(einde voorbeeld)

5.3.2 Aantal termen in een expressie

In het voorafgaande zijn twee algemene expressies gebruikt :

$$f(X_1,\ldots,X_r) = \bigvee_{(S_1,\ldots,S_r)} X_1^{S_1}X_2^{S_2}\ldots X_r^{S_r} \text{ en}$$

$$g(Y_1,\ldots,Y_r) = \bigvee_{(T_1,\ldots,T_r)} Y_1^{T_1}Y_2^{T_2}\ldots Y_r^{T_r} .$$

Voor een paar functies f en g dat aan lemma 3 (paragraaf 5.3.1) voldoet, geldt [27] dat een literal $Y_i^{T_i}$ correspondeert met een literal $X_i^{S_i}$ als $S_i=\Psi_i^{-1}(T_i)$. Een term $Y_1^{T_1}\ldots Y_r^{T_r}$ komt dan overeen met een term $X_1^{S_1}\ldots X_r^{S_r}$.

Voorbeeld 14 [27]
Neem de functies f en g uit het vorige voorbeeld. De literal Y_1^0 komt overeen met $X_1^{(00)}$. De literal Y_1^1 komt overeen met de literal $X_1^{\{01,10\}}$ omdat $\Psi_1^{-1}(1)=\{01,10\}$ volgens de tabel in het vorige voorbeeld. Een term $Y_1^{\{1,2\}}Y_2^0Y_3^{\{2,3\}}$ komt dan overeen met een term $X_1^{\{01,10,11\}}X_2^{\{00,11\}}X_3^{\{10,11\}}$.

(einde voorbeeld)

Is er nu een uitspraak te doen over het aantal termen in een expressie? Het antwoord is ja. In het volgende theorema vindt men een bovengrens voor het aantal termen in een minimale expressie.

Theorema 4 [27]

Gegeven een partitie $(X_1,\ldots,X_r)$ van X en twee functies f en g met :

$f(X_1,\ldots,X_r)=g(\Psi_1(X_1),\ldots,\Psi_r(X_r))$.

Hierin is $\Psi_i : B^{n_i} \longrightarrow M_i$, $M_i=\{0,1,\ldots,k_i-1\}$, $n_i=d(X_i)$ en

$1 \leq |M_i| = k_i \leq 2^{n_i}$.

Laat P en Q minimale expressies in de DF zijn voor respectievelijk f en g dan geldt :

$$t(P)=t(Q) \leq \left(\prod_{i=1}^{r} k_i\right)/(\max\{k_i\}) \ .$$

Hierbij is t(A) het aantal termen in de expressie A.

Theorema 4 geeft een bovengrens voor het aantal termen in een minimale expressie in de DF. Er bestaat [27] minstens een functie waarvan het aantal termen in een minimale expressie in de DF gelijk is aan de bovengrens. Sasao [27] geeft aan hoe zo'n expressie gevormd kan worden.

Met de hierboven beschreven theorie is het mogelijk het aantal termen voor verschillende (soorten) functies te bepalen. In dit verslag worden de volgende drie (groepen van) functies beschouwd :

- Willekeurige functies.
- Partieel symmetrische functies.
- Pariteitsfunctie.

Voor de eenvoud is bij de bespreking uitgegaan van een en dezelfde partitie $(X_1,\ldots,X_r)$. In deze partitie is het aantal binaire variabelen in ieder partitie-element gelijk. Dit aantal

heet q. Er geldt dan $d(X_i)=q$ en $n=q.r$.

Willekeurige functies

Voor een willekeurige functie f geldt [27] dat het nodige en voldoende aantal termen gelijk is aan :

$$(2^q)^r/(2^q)=(2^q)^{(r-1)}=2^{(n-q)} .$$

Voor alle k_i uit theorema 4 geldt immers $k_i=2^q$. Bij een willekeurige functie is er geen aanwijzing te vinden dat een of meer k_i's kleiner zijn dan de bijbehorende n_i's. Voor de volgende functies zijn die aanwijzingen er wel.

In het geval van de triviale partitie (q=1) is het aantal termen $2^{(n-1)}$. Dit is het aantal termen in een minimale expressie voor de pariteitsfunctie. Als q>1 is er een andere functie die dit maximale aantal termen nodig heeft. Sasao geeft [27] aan hoe zo'n functie geconstrueerd kan worden.

Partieel symmetrische functies

Voor een partieel symmetrische functie geldt [27] dat het nodige en voldoende aantal termen gelijk is aan :

$$(q+1)^r/(q+1)=(q+1)^{(r-1)} .$$

Wat is een partieel symmetrische functie? Een functie is partieel symmetrisch met betrekking tot het partitie-element X_i als geldt dat f(X) invariant is voor iedere permutatie van variabelen in X_i. Een functie is partieel symmetrisch als de functie partieel symmetrisch is met betrekking tot alle X_i (i=1,2,...,r).

In elke partitie X_i zitten, in dit geval, q binaire variabelen. Van deze variabelen kunnen er 0,1,... of q gelijk zijn aan een. De ene permutatie met 0 enen levert $f(X|X_i=(0,...,0))$. Evenzo is er maar een permutatie met q enen. Er zijn q permutaties met 1 een. Voor al deze permutaties is $f(X|X_i)$ gelijk. De partitie-functie Ψ_i is in dit geval $\Psi_i(X_i)=|X_i|$, waarbij $|X_i|$ gelijk is aan het aantal enen in X_i. Dan is $k_i=q+1$ voor i=1,2,...,r. Met theorema 4 kan dan het aantal termen bepaald worden.

Pariteitsfunctie

Het nodige en voldoende aantal termen voor een pariteitsfunctie is $2^{(r-1)}$. De partitie functies voor een pariteitsfunctie zijn :

$$\Psi_i(X_i)=\begin{cases}1 \text{ als } |X_i|=1 \pmod 2\\ 0 \text{ als } |X_i|=0 \pmod 2.\end{cases}$$

Een pariteitsfunctie is een speciaal geval van een partieel symmetrische functie. De maximale aantallen termen voor een willekeurige-, partieel symmetrische- en pariteitsfunctie worden, in deze volgorde, steeds kleiner (voor q>1).

5.3.3 Complexiteit van beperkte Σ_3-circuits

In het voorafgaande is aangetoond dat er een één op één afbeelding mogelijk is tussen GBE's en beperkte Σ_3-circuits. Vervolgens zijn er grenzen afgeleid voor het aantal termen in een GBE. In deze paragraaf wordt met behulp van deze grens, een bovengrens afgeleid voor de complexiteit van beperkte Σ_3-circuits.

De complexiteitsgrens wordt eerst bepaald voor een partitie met $d(X_i)=q$ voor $i=1,2,\ldots,r$ en $n=q.r$. Dit is een partitie die "zorgt" voor r evengrote decoders bij een fysische implementatie. Deze partitie is ook in de vorige paragraaf gebruikt.
In het eerste OR veld worden voor ieder van de r partitie-elementen alle 2^q maxtermen van de q binaire ingangsvariabelen gevormd. Ingangen van het eerste OR veld zijn de n binaire variabelen en de ontkenningen. In het OR veld zijn gates met een onbeperkte fan-in beschikbaar (zie paragraaf 3.2.4).
voor iedere maxterm is één OR gate nodig. Per partitie-element zijn dan 2^q gates nodig. Voor r partitie-elementen zijn dan $r.2^q$ OR gates nodig in het eerste OR veld. Verder is in het tweede OR veld een gate nodig voor iedere uitgangsfunctie. In het hier

voorkomende geval van functies $f \in B_n$ betekent dit dat er een OR gate in het tweede OR veld nodig is. Deze OR gate vormt de conjunctie van de uitgangen van het AND veld.
In het AND veld heeft men een AND gate nodig voor iedere term in een GBE voor de te realiseren functie. In de vorige paragraaf is afgeleid hoe groot dit aantal moet zijn.

Samengevat kan men stellen [27] dat het nodige en voldoende aantal gates bij beperkte Σ_3-circuits met r partitie-elementen van q variabelen elk, gelijk is aan :

- $r.2^q+2^{(n-q)}$ +1 voor willekeurige functies.

- $r.2^q+(q+1)^{(r-1)}$ +1 voor partieel symmetrische functies.

- $r.2^q+2^{(r-1)}$ +1 voor een pariteitsfunctie.

Deze getallen zijn dus een bovengrens voor de complexiteit $C_\Omega(f)$ van het beperkte Σ_3-circuit. De basis Ω is hier $\{\wedge, \vee, \neg\}$. In bovenstaande formules zijn de inverters (NOT gates) niet meegeteld.
Er is maar een pariteitsfunctie van n variabelen. Met bovenstaande formule voor de pariteitsfunctie is de complexiteit van een beperkt Σ_3-circuit voor de pariteitsfunctie precies te bepalen. Er zijn vele verschillende willekeurige en partieel symmetrische functies. De twee overige formules hierboven geven een bovengrens voor een groep van functies met een bepaalde eigenschap (bijvoorbeeld partiële symmetrie). De bovengrens wordt slechts gehaald door een (of enkele) functie(s) uit de groep.

Hoe groot zijn de bovengrenzen voor het algemene geval? In dit geval is de partitie $(X_1,\ldots,X_r)$ met $n_i=d(X_i)$, $i=1,\ldots,r$. De getallen n_i mogen willekeurige getallen groter dan nul zijn die voldoen aan : $n_1+\ldots+n_r=n$.
Van ieder partitie-element X_i worden in het eerste OR veld alle

maxtermen van de n_i variabelen gevormd. Voor iedere maxterm is een OR gate nodig. Dit levert een totaal aantal OR gates van :

$$2^{n_1}+\ldots+2^{n_r}=\sum_{i=1}^{r} 2^{n_i} .$$

Verder is voor functies met een uitgang een OR gate in het tweede OR veld nodig. In het AND veld is voor iedere term in de GBE een gate nodig. Met behulp van theorema 4 (paragraaf 5.3.2) zijn de nodige en maximale aantallen termen (gates) te vinden. Dit geschiedt op een manier die analoog verloopt met de bespreking in paragraaf 5.3.2 voor r evengrote partitie-elementen van q variabelen elk.
Het nodige en voldoende aantal gates [27] bij beperkte Σ_3-circuits met r partitie-elementen is gelijk aan :

- $\sum_{i=1}^{r} 2^{n_i}+2^{(n-\max\{n_i\})}+1$ voor willekeurige functies.

- $\sum_{i=1}^{r} 2^{n_i}+(\prod_{i=1}^{r}(n_i+1))/(\max\{n_i+1\})+1$ voor partieel symmetrische functies.

- $\sum_{i=1}^{r} 2^{n_i}+2^{(r-1)}+1$ voor de pariteitsfunctie.

Tot nog toe zijn de afmetingen van beperkte Σ_3-circuits gegeven in het aantal gates dat nodig is. In verschillende artikelen worden de afmetingen voor fysische realisatie in een andere "eenheid" uitgedrukt.
Fleisher en Maissel [7] drukken de afmetingen van een two-level en decoded PLA uit in het aantal bits van het AND veld. Dit aantal bits is gelijk aan het produkt van het totale aantal ingangen en het totale aantal uitgangen van het AND veld. De term bits is als volgt te verklaren. Men koppelt aan een AND veld een "personality" matrix van nullen en enen. Ieder matrix element is een bit. Een een op plaats (i,j) in de matrix betekent dat ingang

i invloed heeft op de uitgang van gate j. Een nul betekent dat de ingang geen invloed heeft. De afmetingen van de OR velden worden niet in rekening gebracht.
Sasao [27], [31] gebruikt nog een andere maat. Sasao maakt zelfs onderscheid tussen decoded PLA's met losse decoders en met een D veld.
Bij decoded PLA's met losse decoders worden de gates van de decoders niet meegeteld. De afmeting van een decoded PLA met losse decoders is (H+m)W. Hierin is m het aantal uitgangen van de PLA. De hoogte (heigth) H is het aantal ingangen van het AND veld i.e. het totale aantal maxtermen. De breedte (width) W is gelijk aan het aantal AND gates i.e. het aantal termen in een GBE in de DF.
Bij een decoded PLA met D veld is de afmeting gedefinieerd als (2n+W)H+Wm met n het aantal variabelen. Ten opzichte van decoded PLA's met losse decoders is de term 2nH toegevoegd. Dit is het produkt van het aantal ingangen en uitgangen van het eerste OR veld.
Samengevat kan gesteld worden dat zowel Sasao als Fleisher en Maissel de afmetingen van PLA's uitdrukken in een produkt van ingangen en uitgangen van de verschillende velden. Een produkt is een maat voor het oppervlak van een AND- of OR veld. Wat het "echte" oppervlak bij fysische realisatie (in een PLA) zal zijn hangt echter ook af van het feit of er wel of geen array folding en/of array partitioning (zie paragraaf 2.2.3) wordt toegepast. Het produkt kan daarom slechts dienen als indicatie voor het fysische oppervlak. De getallen die de complexiteit aangeven zijn veel handzamer (kleiner) dan de getallen van de produkten. Bovendien blijkt het in praktijk zo te zijn dat als de complexiteit groot is het produkt dit ook is. Op grond van deze redenen wordt de oppervlaktemaat in dit verslag zoveel mogelijk vermeden. De afmetingen worden dan uitgedrukt in de complexiteit i.e. het aantal gates van de AND- en OR velden van het Σ_k-circuit i.e. een model van PLA's. Daar de complexiteit meestal gegeven is als som van de complexiteiten van de verschillende

velden, kunnen de afmetingen van Fleisher en Maissel of Sasao uit de complexiteit afgeleid worden.

5.3.4 Boven- en ondergrenzen

De tot nog toe afgeleide grenzen voor de complexiteit van beperkte Σ_3-circuits zijn bovengrenzen. Een bovengrens geeft als functie van de getallen r en $n_1,\ldots,n_r$ (de grootte van de partitie-elementen van de partitie) aan hoeveel gates maximaal nodig zijn om een functie te berekenen. In paragraaf 5.3.3 zijn bovengrenzen gegeven voor groepen van functies met een bepaalde eigenschap. Er is steeds een functie aan te geven [27] die het aantal gates van de bovengrens nodig heeft. De meeste functies hebben minder gates nodig. Er is echter een minimaal aantal gates nodig om een functie te realiseren. De ondergrens geeft dit minimale aantal aan als functie van r en $n_1,\ldots,n_r$.
Met behulp van boven- en ondergrenzen is het mogelijk een uitspraak te doen over de grootte (complexiteit) van een beperkt Σ_3-circuit als functie van de gekozen partitie (r en $n_1,\ldots,n_r$). Hiermee is het mogelijk de getallen r en $n_1,\ldots,n_r$ te bepalen die het Σ_3-circuit met de kleinste complexiteit leveren. Ook is het mogelijk beperkte Σ_3-circuits met verschillende partities te vergelijken. In deze paragraaf worden ten eerste de onder- en bovengrenzen besproken. Vervolgens wordt de complexiteit van two-level en decoded PLA's met two-bit decoders vergeleken. Tot slot wordt met behulp van de bovengrenzen aangegeven welke partities $(r,n_1,\ldots,n_r)$ leiden tot de kleinste complexiteit.

Men zou het liefst de beschikking hebben over een boven- en ondergrens voor iedere functie. In het ideale geval dat deze grenzen samenvallen, is het aantal gates dat nodig is om een functie in een beperkt Σ_3 circuit te realiseren zelfs exact bekend. De pariteitsfunctie (zie paragraaf 5.3.3) is een functie

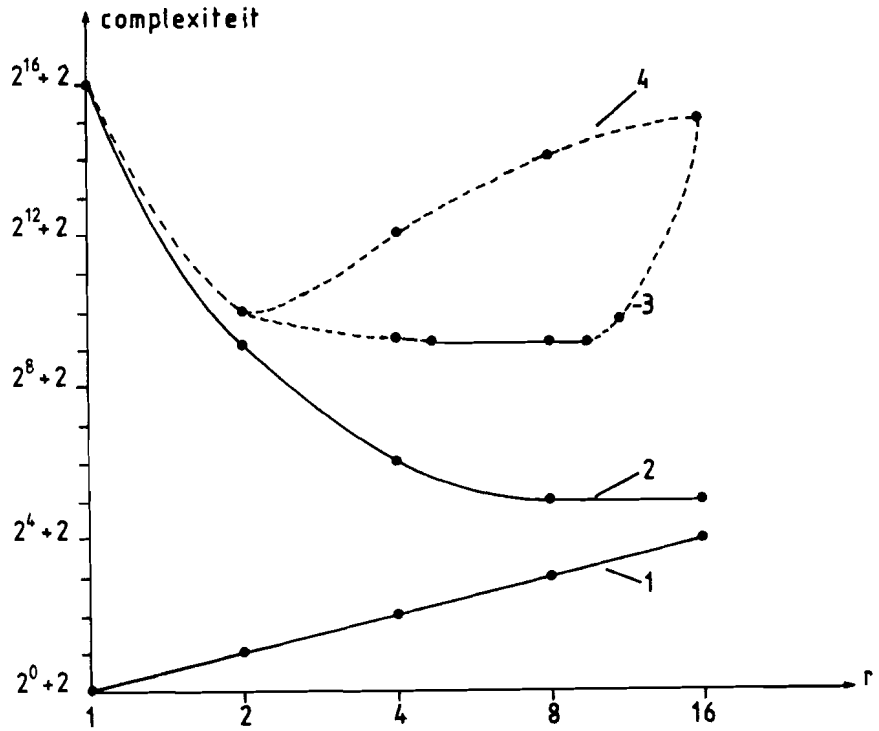
waarvan het aantal gates in een beperkt Σ_3-circuit als functie van r en $n_1,\ldots,n_r$ exact bekend is. Voor de meeste functies gaat dit echter niet op. Boven- en ondergrenzen voor een functie zijn moeilijk te bepalen.
Voor een groep van functies met een bepaalde eigenschap is het vaak wel mogelijk een bovengrens af te leiden. In paragraaf 5.3.3 zijn bovengrenzen afgeleid voor groepen van willekeurige- en partieel symmetrische functies.
Een ondergrens voor de complexiteit voor een functie is [38] moeilijk te geven. Om een ondergrens te bepalen moeten alle circuits die de functie berekenen, bekeken worden. De complexiteit van het "kleinste" circuit levert dan de ondergrens. In feite komt dit neer op het bepalen van een minimale expressie. Het bepalen van een ondergrens kost om deze reden in de praktijk te veel tijd. Daarom wordt een vrij globale ondergrens gebruikt.

Hieronder zijn twee van zulke grenzen voor de complexiteit van beperkte Σ_3-ciruits gegeven. Ze gelden voor iedere functie.
Bij de eerste wordt in het AND veld en tweede OR veld een gate gebruikt. Het circuit berekent dan een functie $f \in B_n$ waarvan de expressie in de DF een term heeft. In het eerste OR veld worden r OR gates gebruikt. Voor ieder van de r partitie-elementen $X_1,\ldots,X_r$ is er een OR gate. Als voor een partitie-element geen OR gate nodig is dan hebben de binaire variabelen in dat element geen invloed op de functiewaarde. In dat geval is de functie onafhankelijk van de variabelen en kunnen ze net zo goed verwijderd worden. Op deze manier kan men functies realiseren die het produkt zijn van r maxtermen waarbij uit ieder partitie-element precies een maxterm gebruikt wordt. Het totale aantal gates (de ondergrens) is dan r+1+1=r+2.
Als slechts een maxterm per partitie-element is toegestaan, is niet iedere literal te berekenen. Bij de tweede ondergrens wordt, in navolging van Fleisher en Maissel [7], het totale aantal mogelijke maxtermen uit het eerste OR veld meegeteld. Dit aantal

is volgens paragraaf 5.3.3 gelijk aan $\sum_{i=1}^{r} 2^{n_i}$ voor een partitie $(X_1,\ldots,X_r)$ met $d(X_i)=n_i$. In figuur 11 zijn beide ondergrenzen getekend voor (een) functie(s) van zestien variabelen. Voor de eenvoud is het aantal partitie-elementen in figuur 11 een macht van 2. Met behulp van bovenstaande formule is eenvoudig in te zien dat voor de ondergrens de partitie-elementen evengroot moeten zijn.

Wat opvalt in figuur 11 is dat de waardes van de tweede ondergrens voor r=8 en r=16 gelijk zijn. Het blijkt dat $\sum_{i=1}^{r} 2^{n_i}$ in het algemeen dezelfde waarde oplevert voor de partities $(X_1,\ldots,X_n)$ met $d(X_i)=1$ $(i=1,\ldots,n)$ en $(X_1,\ldots,X_{n/2})$ met $d(X_i)=2$ $(i=1,\ldots,n/2$ voor even n). De eerste partitie is de triviale partitie.

Via de modellen (Σ_k-circuits) is het nu mogelijk two-level en decoded PLA's met two-bit decoders met elkaar te vergelijken. Decoded PLA's met two-bit decoders zijn nooit groter dan two-level PLA's. Deze uitspraak is echter niet zonder meer waar. De reden hiervoor is dezelfde als die, die geleid heeft tot de invoering van Σ_k-circuits, namelijk de invloed van het fysisch ontwerp.
Beter is het om te zeggen dat het aantal ingangen en uitgangen van het AND veld van een decoded PLA met two-bit decoders (Σ_3-circuit met een partitie $d(X_i)=2$ voor $i=1,\ldots,n/2$ (n even)) nooit groter is dan respectievelijk het aantal ingangen en uitgangen van het AND veld van een two-level PLA. Voor het aantal ingangen van het AND veld volgt dit uit de twee hierboven genoemde ondergrenzen (zie ook figuur 11). Voor het aantal uitgangen van het AND veld dient dit wellicht even toegelicht te worden. De



Figuur 11 : Onder- en bovengrenzen voor de complexiteit van beperkte Σ_3-circuits voor willekeurige functies van zestien variabelen

1 ondergrens 1

2 ondergrens 2

3 bovengrens ongelijke grootte partitie-elementen

4 ondergrens gelijke grootte partitie-elementen

term $2^{n-\max\{n_i\}}$ in de uitdrukking voor de complexiteit van beperkte Σ_3-circuits voor willekeurige functies in paragraaf 5.3.3 geeft dit aantal. Voor de triviale partitie $(X_1,\ldots,X_n)$ geldt $n_i=1$, voor een niet triviale partitie geldt dat er minstens een $n_i>1$ is. Het aantal uitgangen (AND gates of termen) is dus nooit groter in het geval de triviale partitie gebruikt wordt. Ook gevoelsmatig is het duidelijk te maken. Een term in een GBE

voor de triviale partitie is altijd te schrijven als een term in een GBE met een niet triviale partitie.

De marges tussen boven- en ondergrenzen voor de complexiteit als functie van r en $\underline{n}=(n_1,\ldots,n_r)$ zijn vaak aanzienlijk (zie bijvoorbeeld figuur 11). Het is moeilijk of ondoenlijk om scherpe of samenvallende boven- en ondergrenzen te geven voor (groepen van) functies. Ook zullen er maar enkele functies uit een groep zijn die het aantal gates volgens de grenzen nodig hebben. De complexiteit van beperkte Σ_3-circuits voor de meeste functies (uit een groep) ligt tussen boven- en ondergrens in.
Hoe is het mogelijk de invloed van verschillende partities op de complexiteit van beperkte Σ_3-circuits te meten? (NB een Σ_2-circuit is een beperkt Σ_3-circuit met de triviale partitie.) Hiervoor wordt het gemiddelde van de complexiteit van een aantal random gegenereerde functies genomen. In onderstaande tabel 4 (overgenomen van Sasao [27]) zijn het gemiddelde aantal gates in het AND veld van beperkte Σ_3-circuits uitgezet voor de triviale partitie en de partitie met $d(X_i)=2$ ($i=1,\ldots,n/2$ en n is even) voor een optimale en niet optimale verdeling (assignment) van de variabelen over de partitie-elementen (zie paragraaf 5.4). De functies zijn afhankelijk van acht binaire variabelen. Het getal u is het percentage enen in de functiewaardetabel van de functies. Ieder getal in de tabel is het gemiddelde van de resultaten van tien verschillende functies. Bij de niet optimale assignment is het gemiddelde van alle 105 mogelijke assignments (zie paragraaf 5.4.1) genomen. Uit de tabel blijkt duidelijk dat (de AND velden van) beperkte Σ_3-circuits met $d(X_i)=2$ aanzienlijk kleiner zijn. Afgezien van de effecten van array folding en array partitioning betekent dit dat decoded PLA's met two-bit decoders gemiddeld kleiner zullen zijn dan two-level PLA's.

Tabel 4 : Statistische resultaten van de vergelijking van verschillende partities

partitie \ u (in %)	5	10	15	20	30	40
$d(X_i)=2$ optimale partitie	8.4	14.7	19.9	24.2	24.8	30.2
$d(X_i)=2$ niet optimale partitie	10.1	17.3	22.7	27.8	31.8	33.6
$d(X_i)=1$	10.8	19.2	26.5	33.1	39.5	44.4

De bovengrens voor een willekeurige functie is volgens paragraaf 5.3.3 gelijk aan $\sum_{i=1}^{r} 2^{n_i} + 2^{(n-\max\{n_i\})} + 1$. Stel dat r i.e. het aantal partitie-elementen gegeven is. De vector $\underline{n}$ wordt gedefinieerd als $\underline{n}=(n_1,\ldots,n_r)$. De getallen $n_1,\ldots,n_r$ moeten voldoen aan $n_1+\ldots+n_r=n$. De bovengrens wordt dan op de volgende manier gevonden. Bepaal bij gegeven r het minimum van bovenstaande expressie gerekend over alle mogelijke vectoren $\underline{n}$. De $\underline{n}$ waarvoor dit minimum optreedt is te benaderen met behulp van de gradiënt van bovenstaande formule.
De vector $\underline{n}$ waarbij het minimum optreedt ziet er als volgt uit. Neem aan dat $n_1 \geq n_2,\ldots,n_r$. Voor de "minimale" $\underline{n}$ geldt :
- Indien mogelijk $n_1=n_2+\ldots+n_r$ en anders $n_1 \approx n_2+\ldots+n_r$.
- $n_1,\ldots,n_r$ moeten zo klein mogelijk, liefst aan elkaar gelijk en, zo mogelijk, groter dan 1 zijn.

Voor n=16 en enkele waardes van r zien de vectoren er als volgt uit :

r=1 : $\underline{n}$=(16) (65538)
r=2 : $\underline{n}$=(8,8) (769)
r=4 : $\underline{n}$=(8,3,3,2) (533)
r=5 : $\underline{n}$=(8,2,2,2,2) (529)
r=8 : $\underline{n}$=(8,2,1,1,1,1,1,1) (529)
r=16 : $\underline{n}$=(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1) (32801)
Achter de vectoren is de complexiteit aangegeven. De complexiteit voor vectoren $\underline{n}$ met $n_1=...=n_r=n/r$ (evengrote partitie-elementen) is :
r=4 : $\underline{n}$=(4,4,4,4) (4161)
r=8 : $\underline{n}$=(2,2,2,2,2,2,2,2) (16417)
Voor n=1,2 en 16 zijn de resultaten hetzelfde als hierboven.

Gerekend over alle r levert de vector $\underline{n}$=(8,2,2,2,2) voor r=5 de kleinste complexiteit. Voor deze $\underline{n}$ is de complexiteit gelijk aan 529. Deze vector levert hetzelfde getal als bijvoorbeeld $\underline{n}$=(8,2,1,1,1,1,1,1). Een overstap van twee partitie-elementen met een variabele naar een partitie-element van twee variabelen levert, zoals hierboven al is toegelicht, nooit een verslechtering (grotere complexiteit) op. De bovengrenzen van bovenstaande vectoren met partitie-elementen van gelijke en ongelijke grootte zijn in figuur 11 getekend.

De hier gebruikte bovengrenzen gelden voor willekeurige functies. Voor (groepen van) functies met speciale eigenschappen kan men ook bovengrenzen afleiden. Deze zullen nooit groter zijn dan de bovengrens voor willekeurige functies. Voor partieel symmetrische functies bijvoorbeeld is de bovengrens met behulp van de formules van paragraaf 5.3.3, op dezelfde manier te vinden als hierboven gebruikt is voor willekeurige functies. Ook voor de pariteitsfunctie is de bovengrens op deze manier te bepalen. Zoals al eerder gezegd is, valt deze samen met de ondergrens. Sasao heeft in [27] de vectoren $\underline{n}$ bepaald voor verschillende waardes van n voor willekeurige-, partieel symmetrische- en pariteitsfuncties, die de laagste waarde voor de bovengrens

geven, gerekend over alle r. Voor willekeurige functies met n=16 wordt daar ook de vector $\underline{n}=(8,2,2,2,2)$ gevonden. Opvallend hierbij is dat de beste vectoren voor partieel symmetrische functies bestaan uit relatief weinig elementen. Dit komt overeen met partities met relatief weinig partitie-elementen en meerdere variabelen per element.

De complexiteit van een beperkt Σ_3-circuit voor een gegeven functie ligt altijd tussen de boven- en ondergrens. Voor een (willekeurige) functie van n variabelen is het mogelijk (zie boven) r en $\underline{n}$ te bepalen waarvoor de bovengrens minimaal is. Deze vector $\underline{n}=(n_1,\ldots,n_r)$ kan gebruikt worden bij het ontwerpen van een beperkt Σ_3-circuit voor een bepaalde functie. Voor deze partitie wordt eventueel een optimale verdeling van de variabelen over de partitie-elementen (zie paragraaf 5.4) bepaald en de GBE voor deze functie geminimaliseerd. Verwacht wordt dat de partitie met de kleinste bovengrens uiteindelijk ook het beperkte Σ_3-circuit zal leveren met de kleinste complexiteit. Dit hoeft echter helemaal niet zo te zijn.
Een dergelijke keuze, het gebruiken van de mogelijkheid die de kleinste bovengrens levert, komt vaker voor. In de volgende paragraaf zal duidelijk worden dat ook bij de verdeling van de variabelen over de partitie-elementen, afgegaan wordt op bovengrenzen voor de circuitcomplexiteit.

5.4 Input variable assignment

In de voorafgaande paragrafen is steeds uitgegaan van een partitie $(X_1,\ldots,X_r)$ van een variabele $X=(x_1,\ldots,x_n)$. Er is niet stil gestaan bij de keuze van de grootte van de partitie-elementen en de verdeling van de n binaire variabelen over de r partitie-elementen. In paragraaf 5.1 is wel opgemerkt dat het aantal termen in een minimale GBE voor een functie, afhankelijk

is van de partitie. In deze paragraaf wordt nader ingegaan op het kiezen van een partitie. Het probleem van het vinden van een "optimale" partitie voor een beperkt Σ_3-circuit staat bekend onder de naam "input variable assignment problem". De optimale partitie is die partitie waarmee een absoluut minimale (weinig termen en weinig literals per term) expressie in de DF, gerekend over alle mogelijke partities, voor een bepaalde functie wordt gevonden. Het input variable assignment problem, kortweg assignment probleem genoemd, valt uiteen in twee delen :

- Bepalen van getallen $n_i=d(X_i)>0$, $i=1,2,...,r$, die voldoen aan $n_1+n_2+...+n_r=n$.
- Verdelen van de variabelen over de partitie-elementen van een partitie met de hierboven afgeleide grootte.

In de literatuur [31], [29] gaat men er vaak vanuit dat de getallen $n_1,...,n_r$ gegeven zijn. Dit komt in de praktijk voor als het beperkte Σ_3-circuit een model is van een decoded PLA met one-, two- of three-bit decoders. Deze PLA's zijn "populair" omdat het aantal maxtermen (aantal ingangen in het AND veld) niet zo groot is. Voor one- en two-bit decoders is het aantal zelfs gelijk. Voor het bepalen van de getallen n_i zijn geen algoritmes gevonden. Misschien is het mogelijk functies in te delen in groepen met een bepaalde eigenschap. Voor functies met een bepaalde eigenschap is dan wellicht een bovengrens voor de complexiteit af te leiden. Vervolgens worden de n_i $(i=1,...,r)$ bepaald waarvoor de bovengrens minimaal is (zie paragraaf 5.3.4). In het nu volgende is er vanuit gegaan dat de getallen r en $n_1,...,n_r$ gegeven zijn. Het assignment probleem wordt in dit geval gereduceerd tot het vinden van een verdeling van de variabelen over de partitie-elementen.
Om een idee te krijgen van de omvang van het probleem, wordt in paragraaf 5.4.1 eerst het aantal mogelijke verdelingen van variabelen over een partitie bepaald. Daarna worden in paragraaf 5.4.2 enkele heuristische algoritmes voor het assignment probleem besproken.

5.4.1 Aantal mogelijke assignments

Bij een algemene partitie $(X_1,...,X_r)$ van n variabelen zijn er verschillende vrijheden voor de getallen $n_i=d(X_i)$. Voor deze getallen geldt : $n_i>0$, i=1,...,r en $n_1+...+n_r=n$.
Aan de partitie wordt een vector $\underline{n}=(n_1,...,n_r)$ verbonden. De lengte van de vector $\underline{n}$ is gelijk aan r. Voor r i.e. het aantal partitie-elementen, geldt $1\leq r\leq n$. Als r=n dan is de partitie de triviale partitie. Als r=1 dan worden alle maxtermen van n variabelen in het eerste OR veld berekend. In dit geval is het beperkte Σ_3-circuit een model voor een zogenaamde ROM waarin een willekeurige functie als produkt van maxtermen geschreven is. Het tweede OR veld is nu overbodig.
Voor een gegeven $\underline{n}$ (en r) zijn de variabelen op een aantal manieren te verdelen over de verschillende partitie-elementen. Hoeveel manieren zijn er?

Voor het eenvoudige geval n=8 en $n_i=2$ voor i=1,...,r is dit aantal hieronder afgeleid. Als $n_i=2$ dan betekent dit dat r=n/2=4. Er zijn vier groepjes van twee variabelen elk. De variabelen worden a,b,c,d,e,f,g en h genoemd. Deze worden over acht plaatsen (plaats 1 (p1) tot en met plaats 8 (p8)) in de vier partitie-elementen verdeeld.
In het eerste partitie-element komen twee variabelen. Dit kan op $\binom{8}{2}$ manieren. In het tweede partitie-element komen de volgende twee variabelen. Dit kan op $\binom{6}{2}$ manieren, twee variabelen zijn immers al gebruikt in het eerste partitie-element. Voor het derde en vierde partitie-element zijn er dan respectievelijk $\binom{4}{2}$ en $\binom{2}{2}$ manieren. Het aantal manieren is dan $\binom{8}{2}.\binom{6}{2}.\binom{4}{2}.\binom{2}{2}$.
Dit aantal moet nog gedeeld worden door het aantal permutaties van de partitie-elementen i.e. r!=(n/2)!=4!. De verdeling p1,p2,p3,p4,p5,p6,p7,p8=a,b,c,d,e,f,g,h levert bijvoorbeeld hetzelfde resultaat als p1,p2,p3,p4,p5,p6,p7,p8=c,d,a,b,e,f,g,h.
Het totale aantal mogelijke verdelingen van de acht variabelen

over de vier partitie-elementen met ieder twee variabelen, is dan $(\binom{8}{2}.\binom{6}{2}.\binom{4}{2}.\binom{2}{2})/4!=105$.

Voor het algemene geval met "willekeurige" n_i vindt men de volgende formule. In de teller staat :

$$\binom{n}{n_1}.\binom{n-n_1}{n_2}.....\binom{n-n_1-...-n_{r-1}}{n_r}.$$

Ook hier moet gedeeld worden door het aantal permutaties van de partitie-elementen. Het is alleen mogelijk partitie-elementen van gelijke grootte te permuteren. Laat c_i het aantal getallen zijn uit $n_1,...,n_r$ dat gelijk is aan i $(i=1,...,n)$. Het aantal mogelijke verdelingen van de variabelen over de partitie-elementen is dan :

$$(\binom{n}{n_1}.\binom{n-n_1}{n_2}.....\binom{n-n_1-...-n_{r-1}}{n_r})/(c_1!.....c_n!)=$$

$$=n!/(n_1!.....n_r!.c_1!.....c_n!).$$

Voor tien variabelen (n=10) en vijf partitie-elementen van ieder twee variabelen ($n_i=2$ voor $i=1,2,...,5$ dus $c_2=5$ en $c_i=0$ voor $i\neq 2$) levert bovenstaande formule al 945 manieren. Het totale aantal mogelijke oplossing voor het assignment probleem is voor praktische problemen met bijvoorbeeld tien of meer variabelen, erg groot. Het is in de praktijk ondoenlijk om voor iedere mogelijke assignment (verdeling van de variabelen) een minimale expressie te bepalen en hieruit de "kleinste" minimale expressie te nemen. Het uitproberen van alle mogelijkheden vergt een te grote hoeveelheid rekentijd. In de volgende paragraaf wordt besproken hoe binnen een redelijke rekentijd en met een beperkte geheugenruimte een (sub)optimale assignment te vinden is.

5.4.2 Algoritmes voor input assignment

In het voorafgaande is het input assignment probleem geformuleerd. Ook is duidelijk geworden dat het ondoenlijk is door middel van proberen een goede assignment te vinden. Stel nu dat de getallen $n_i=d(X_i)$ gegeven zijn i.e. de grootte van de partitie-elementen ligt vast. Is het nu mogelijk op voorhand aan te geven welke assignment (verdeling van variabelen over de partitie-elementen) het beste resultaat levert? Het antwoord is ja. In deze paragraaf worden drie principes besproken om de optimale verdeling van de variabelen te bepalen. Het zijn methodes van :

- Sasao en Terada.
- Sasao.
- Ektare en Al-Sheakhly.

Bij de eerste methode wordt gebruik gemaakt van de bovengrenzen uit paragraaf 5.3.2. Het bepalen van de bovengrenzen vergt echter nogal wat rekentijd. Bij de tweede methode wordt daarom een schatting voor de bovengrens gebruikt. Deze schatting is veel sneller te bepalen. De derde methode werkt met spectrale technieken om de verdeling van de variabelen over de partitie-elementen vast te stellen.

Sasao en Terada [31]

In paragraaf 5.3.1 is multiple-valued decomposition geïntroduceerd. Hiermee kan een geschikte assignment bepaald worden. Stel men heeft een partitie i.e. de n variabelen zijn op een bepaalde manier verdeeld over de r partitie-elementen. Voor elk partitie-element X_i wordt k_i berekend. De vector $\underline{k}=(k_1,\ldots,k_r)$ wordt aan deze partitie gekoppeld. Voor andere partities zijn deze vectoren ook te bepalen. De partitie die bij de vector $\underline{k}$ hoort die in theorema 4 van paragraaf 5.3.2 de kleinste bovengrens levert, wordt de beste kandidaat voor een optimale assignment [31]. Deze werkwijze is in het volgende

voorbeeld verduidelijkt.

Voorbeeld 15 [31]

Uitgangspunt is de functie f van voorbeeld 11 in paragraaf 5.1. Dit is een functie van vier variabelen. Stel dat alleen partities $d(X_i)=2$ bekeken worden. Er zijn drie van deze partities : $((x_1,x_2),(x_3,x_4))$, $((x_1,x_3),(x_2,x_4))$ en $((x_1,x_4),(x_2,x_3))$.

Voor de partitie (X_1,X_2) met $X_1=(x_1,x_2)$ en $X_2=(x_3,x_4)$ geldt :

$f(X|X_1=(00))=X_2^{\{00,01,10\}}$ $\quad$ $f(X|X_2=(00))=X_1^{\{00,01,10\}}$

$f(X|X_1=(01))=X_2^{\{00,01,11\}}$ $\quad$ $f(X|X_2=(01))=X_1^{\{00,01,11\}}$

$f(X|X_1=(10))=X_2^{\{00,11\}}$ $\quad$ $f(X|X_2=(10))=X_1^{\{00,11\}}$

$f(X|X_1=(11))=X_2^{\{01,10\}}$ $\quad$ $f(X|X_2=(11))=X_1^{\{01,10\}}$

Dit levert $k_1=4$ en $k_2=4$. De partities π_1 en π_2 (zie paragraaf 5.3.1) zijn respectievelijk ([00],[01],[10],[11]) en ([00],[01],[10],[11]).

Voor de partitie (X_1,X_2) met $X_1=(x_1,x_3)$ en $X_2=(x_2,x_4)$ geldt :

$f(X|X_1=(00))=X_2^{\{00,01,10,11\}}$ $\quad$ $f(X|X_2=(00))=X_1^{\{00,01,10\}}$

$f(X|X_1=(01))=X_2^{\{00,11\}}$ $\quad$ $f(X|X_2=(01))=X_1^{\{00,11\}}$

$f(X|X_1=(10))=X_2^{\{00,11\}}$ $\quad$ $f(X|X_2=(10))=X_1^{\{00,11\}}$

$f(X|X_1=(11))=X_2^{\{01,10\}}$ $\quad$ $f(X|X_2=(11))=X_1^{\{00,01,10\}}$

In dit geval geldt $k_1=3$ en $k_2=2$. De partities π_1 en π_2 zijn nu respectievelijk ([00],[01,10],[11]) en ([00,11],[01,10]).

Voor de partitie (X_1,X_2) met $X_1=(x_1,x_4)$ en $X_2=(x_2,x_3)$ geldt :

$f(X|X_1=(00))=X_2^{\{00,01,10\}}$ $\quad$ $f(X|X_2=(00))=X_1^{\{00,01,10\}}$

$f(X|X_1=(01))=X_2^{\{00,10,11\}}$ $\quad$ $f(X|X_2=(01))=X_1^{\{00,11\}}$

$f(X|X_1=(10))=X_2^{\{00,11\}}$ $\quad$ $f(X|X_2=(10))=X_1^{\{00,01,11\}}$

$f(X|X_1=(11))=X_2^{\{01,10\}}$ $\quad$ $f(X|X_2=(11))=X_1^{\{01,10\}}$

Ook nu is $k_1=4$ en $k_2=4$.

Volgens theorema 4 uit paragraaf 5.3.2 zijn de bovengrenzen voor het aantal termen in een minimale expressie met deze drie partities respectievelijk gelijk aan 4,2 en 4. De beste kandidaat voor de optimale partitie is dus de tweede i.e. $X_1=(x_1,x_3)$ en $X_2=(x_2,x_4)$. Is deze partitie nu ook werkelijk een optimale partitie? Daartoe worden de volgende drie minimale expressies beschouwd.

Voor $X_1=(x_1,x_2)$ en $X_2=(x_3,x_4)$:

$$X_1^{\{00,01\}}X_2^{\{00,01\}}+X_1^{\{00,11\}}X_2^{\{01,10\}}+X_1^{\{01,10\}}X_2^{\{00,11\}}.$$

Voor $X_1=(x_1,x_3)$ en $X_2=(x_2,x_4)$:

$$X_1^{\{00,01,10\}}X_2^{\{00,11\}}+X_1^{\{00,11\}}X_2^{\{01,10\}}.$$

Voor $X_1=(x_1,x_4)$ en $X_2=(x_2,x_3)$:

$$X_1^{\{00,11\}}X_2^{\{01,10\}}+X_1^{\{01,10\}}X_2^{\{00,11\}}+X_1^{\{00,01\}}X_2^{\{00,10\}}.$$

De tweede expressie hierboven heeft de minste termen. De partitie (X_1,X_2) met $X_1=(x_1,x_3)$ en $X_2=(x_2,x_4)$ is dus de optimale.
(einde voorbeeld)

Hierboven is een regel afgeleid voor de keuze van een bepaalde verdeling van variabelen over de partitie. De regel maakt het mogelijk twee of meer verdelingen met elkaar te vergelijken. De regel gebruikt de bovengrens van het aantal termen in een minimale expressie als criterium om een assignment te kiezen. De assignment die de kleinste bovengrens levert, wordt gekozen. Hierbij moet men bedenken dat het best mogelijk kan zijn dat er een betere assignment bestaat. Het gaat immers om bovengrenzen voor het aantal termen in een expressie.

Sasao [29]

Sasao en Terada [31] bieden alleen de mogelijkheid meerdere assignments onderling te vergelijken. Er is niet aangegeven hoe een goede assignment gevonden kan worden. Sasao gaat, in [29], nader in op het input assignment probleem. Sasao zegt dat er reeds zestien verschillende, heuristische algoritmes ontwikkeld zijn voor input assignment (zie bijvoorbeeld [13]). Zelfs de eenvoudigste levert al circuits die 10% kleiner zijn dan bij een random verdeling van de variabelen over de partities. Sasao presenteert in [29] zelf het volgende, heuristische algoritme voor assignment. Dit algoritme levert (sub)optimale assignments voor partities $(X_1,\ldots,X_r)$ met $d(X_i)=2$. Het kan gebruikt worden voor functies $f \in B_{n,m}$.

Er wordt gebruik gemaakt van een complete graaf, de zogenaamde assignment graaf. De graaf heeft n knooppunten, voor iedere variabele $x_1,\ldots,x_n$ een. Tussen ieder paar knooppunten is een tak aangebracht. Aan iedere tak wordt een getal gekoppeld. Dit getal, q(i,j), tussen knooppunt i en j is gelijk aan het aantal verschillende termen in een expressie voor f waarin de variabelen x_i en x_j niet voorkomen. Voor een functie uit B_n (m=1) zal dit geen probleem opleveren. Voor een functie uit $B_{n,m}$ met m>1 verdient dit enige toelichting. Zo'n functie kan met m expressies beschreven worden. Per expressie wordt het aantal verschillende termen bepaald als x_i en x_j verwijderd zijn. Het getal q(i,j) is

gelijk aan de som van de m op deze manier gevonden getallen.
De procedure voor het verkrijgen van een (sub)optimale assignment met behulp van een assignment graaf ziet er als volgt uit :
Procedure

1) Bepaal een (bijna) minimale expressie in de DF voor een functie f.
2) Bepaal de assignment graaf.
3) Selecteer een groep van takken zodanig dat :
   - De som van de gewichten van de takken uit de groep minimaal is.
   - Ieder knooppunt door precies een tak uit de groep gebruikt wordt.
4) Vorm de partitie $(X_1,\ldots,X_r)$ met $d(X_i)=2$. De variabelen x_i en x_j zitten in X_a als de tak i,j in de geselecteerde groep zit.

Het volgende voorbeeld licht de procedure toe.

Voorbeeld 16

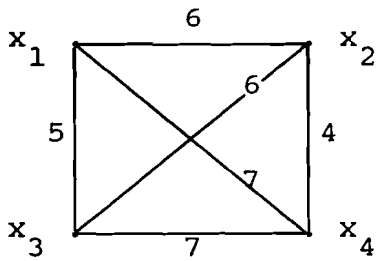
Uitgangspunt is weer de functie f uit voorbeeld 11 in paragraaf 5.1 en voorbeeld 15 in paragraaf 5.4.2. Een minimale expressie voor f is :

$$f(x_1,x_2,x_3,x_4)=\bar{x}_1\bar{x}_3+\bar{x}_1\bar{x}_2\bar{x}_4+\bar{x}_2\bar{x}_3\bar{x}_4+\bar{x}_1x_2x_4+x_2\bar{x}_3\bar{x}_4+$$
$$x_1\bar{x}_2x_3x_4+x_1x_2x_3\bar{x}_4.$$

Er zijn $\binom{4}{2}=6$ verschillende q(i,j). De q(i,j) zijn achtereenvolgens gelijk aan :

q(1,2)=6 , q(1,4)=7 , q(2,4)=4 ,
q(1,3)=5 , q(2,3)=6 en q(3,4)=7 .

De graaf voor f ziet er dan als volgt uit :



Er zijn drie mogelijke partities (X_1,X_2) met $d(X_i)=2$ voor i=1,2.
De sommen van de gewichten voor deze partities zijn :

- Voor $((x_1,x_2),(x_3,x_4))$: $q(1,2)+q(3,4)=13$.
- Voor $((x_1,x_3),(x_2,x_4))$: $q(1,3)+q(2,4)=9$.
- Voor $((x_1,x_4),(x_2,x_3))$: $q(1,4)+q(2,3)=13$.

Hieruit wordt geconcludeerd dat de tweede partitie de beste assignment levert. Uit het vorige voorbeeld 15 blijkt dat dit ook daadwerkelijk de meest optimale assignment is.
(einde voorbeeld)

Ektare en Al-Sheakhly [6]

Hierboven is een heuristische procedure gegeven om op voorhand informatie (zogenaamde a priori informatie) te verzamelen over de invloed van verschillende partities. Deze informatie kan met relatief weinig werk verkregen worden. De heuristische procedure van Sasao [29] werkt alleen voor assignments aan partities $(X_1,...,X_r)$ met $d(X_i)=2$. Hieronder wordt een andere oplossing voor het assignment probleem aangedragen.
Ektare en Al-Sheakhly merken op [6] dat het belangrijk lijkt zulke a priori informatie te hebben om een efficiënt ontwerp mogelijk te maken. Ze tonen aan [6] dat deze informatie eenvoudig verkregen kan worden als de functie een vorm van symmetrie heeft. Het wordt zelfs betoogd dat de assignment er niet toe doet voor functies die geen vorm van symmetrie bevatten.

Als een functie een vorm van symmetrie bevat dan maakt het uit welke assignment men gebruikt. Het ligt dus voor de hand een

manier te zoeken waarmee symmetrieën te detecteren zijn. De detectie van functiesymmetrieën is [6] uitgebreid bestudeerd. Het blijkt dat het gebruik van spectrale technieken voor de detectie van symmetrieën erg nuttig is.
Het spectrum van een functie f bestaat uit een aantal zogenaamde spectrale coëfficiënten. Deze worden verkregen door een transformatie T : [T] F] = S] . Hierin is F een vector van de functiewaardetabel van f waarbij 0 door +1 en 1 door -1 zijn vervangen. De matrix T is een $2^n \times 2^n$ Rademacher-Walsh transformatie matrix. De vector S bevat de 2^n spectrale coëfficiënten. Zo'n coëfficiënt is aangegeven met de letter r gevolgd door enkele subscripts.

Voorbeeld 17
Neem een functie f van twee variabelen x_1 en x_2. Een expressie voor de functie is $f=\bar{x}_1\bar{x}_2+\bar{x}_1x_2+x_1\bar{x}_2$. De vector S is gelijk aan :

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \\ -2 \\ 2 \end{bmatrix}$$

$S=(r_0,r_1,r_2,r_{12})=(-2,-2,-2,2)$
(einde voorbeeld)

De spectrale coëfficiënten leveren de volgende informatie :
- r_0 geeft de verhouding van het aantal enen en nullen in de functiewaardetabel van f aan.
- r_i is een maat voor de correlatie tussen x_i en f.
- r_{ij} is een maat voor de correlatie tussen $x_i \oplus x_j$ en f.
- Etcetera.

Voorbeeld 18
De vector S uit het vorige voorbeeld levert de volgende informatie:

- Het aantal nullen min het aantal enen in de functiewaardetabel is $r_0=-2$.
- f is gecorreleerd met $\bar{x}_1$ (vanwege "-" teken) ($r_1=-2$).
- f is gecorreleerd met $\bar{x}_2$ (vanwege "-" teken) ($r_2=-2$).
- f is gecorreleerd met $x_1 \oplus x_2$ ($r_{12}=2$).

In de som van mintermen expressie voor f in het vorige voorbeeld zijn de correlaties duidelijk terug te vinden. Het gedeelte $\bar{x}_1 x_2 + x_1 \bar{x}_2$ is immers gelijk aan $x_1 \oplus x_2$. En in de term $\bar{x}_1 \bar{x}_2$ zitten $\bar{x}_1$ en $\bar{x}_2$.
(einde voorbeeld)

Met de vector S kunnen verschillende symmetrieën gedetecteerd worden. Drie symmetrieën die Ektare en Al-Sheakhly gebruiken zijn :

- Multiform symmetry (MS) als
  $f(x_1,\ldots,0,0,\ldots,x_n)=f(x_1,\ldots,1,1,\ldots,x_n)$ en
  $f(x_1,\ldots,0,1,\ldots,x_n)=f(x_1,\ldots,1,0,\ldots,x_n)$.

- Equivalence symmetry (ES) als
  $f(x_1,\ldots,0,0,\ldots,x_n)=f(x_1,\ldots,1,1,\ldots,x_n)$.

- Non-equivalence symmetry (NES) als
  $f(x_1,\ldots,0,1,\ldots,x_n)=f(x_1,\ldots,1,0,\ldots,x_n)$.

Deze symmetrieën hebben steeds betrekking op twee variabelen die hier x_i en x_j genoemd zullen worden. De symmetrieën [6] MS, ES en NES zijn te detecteren met behulp van de vector S.
Een functie heeft MS als voor alle k,l,... ongelijk aan i,j geldt :

$r_i = r_j = 0$
$r_{ik} = r_{jk} = 0$
$r_{il} = r_{jl} = 0$
$r_{ikl} = r_{jkl} = 0$
etcetera.

De voorwaarden voor ES zijn :

$r_i + r_j = 0$
$r_{ik} + r_{jk} = 0$
$r_{il} + r_{jl} = 0$
$r_{ikl} + r_{jkl} = 0$
etcetera.

Voor NES zijn de voorwaarden tot slot :

$r_i - r_j = 0$
$r_{ik} - r_{jk} = 0$
$r_{il} - r_{jl} = 0$
$r_{ikl} - r_{jkl} = 0$
etcetera.

Hoe worden MS, ES en NES gebruikt voor het bepalen van een assignment? Voor een willekeurige functie is de volgende procedure voor het samenstellen van een assignment op te stellen :

Procedure

Gegeven een functie f van n binaire variabelen. Achtereenvolgens worden de volgende stappen uitgevoerd. Bij initialisatie is f het enige residu en q=n.

1) Bepaal de spectra van alle residuen van q variabelen met behulp van de eerder beschreven transformatie.
2) Kijk of er MS, ES of NES aanwezig is.
3) Neem een van de gevonden symmetrieën. Hierbij heeft MS een hogere prioriteit dan ES of NES. Splits in alle residuen de twee variabelen met symmetrie af en verkrijg langs deze weg nieuwe residuen ter lengte q-2. (NB als er geen MS, ES en NES is, neem dan twee willekeurige variabelen.)
4) Ga verder met stap 1) als de residuen van meer dan een variabele afhankelijk zijn.

Het afsplitsen gaat als volgt. Stel dat f een functie is van acht variabelen $(x_1,\ldots,x_8)$ en ES heeft met betrekking tot x_1 en x_2. De functie f is dan te schrijven als :

$$f(x_1,\ldots,x_8) = (\bar{x}_1\bar{x}_2+x_1x_2)\ f_1(x_3,\ldots,x_8) + \bar{x}_1x_2\ f_2(x_3,\ldots,x_8) +$$

$$x_1\bar{x}_2\ f_3(x_3,\ldots,x_8).$$

De functies f_1, f_2 en f_3 heten de residuen van f.

De assignment bestaat nu uit partitie-elementen met elk twee binaire variabelen. Twee variabelen zitten in hetzelfde partitie-element als ze in bovenstaande procedure samen zijn afgesplitst. Het is goed mogelijk dat alhoewel een functie geen MS (ES of NES) bezit met betrekking tot de variabelen x_i en x_j, een of meer van de residuen wel een vorm van symmetrie heeft met betrekking tot deze variabelen.
Een partieel symmetrische functie uit paragraaf 5.3.2 bezit MS met betrekking tot twee variabelen in ieder partitie-element. Hierbij is aangenomen dat $n_i=2$ $(i=1,2,\ldots,n/2)$.

In het voorafgaande is steeds gekeken naar symmetrieën met betrekking tot twee variabelen. Het komt soms voor dat functies geen symmetrie met betrekking tot twee variabelen bezitten. In dat geval kan gekeken worden naar symmetrieën van meer dan twee variabelen. Muzio et al. [21] bespreken onder andere het detecteren van deze symmetrieën van meer dan twee variabelen. Er worden ook oplossingen gegeven voor het vinden van symmetrieën van niet volledig gespecificeerde functies. Zolang de symmetrieën zich niet over "te veel" variabelen uitstrekken, kan men ze binnen een redelijke rekentijd bepalen. Op deze manier is het ook mogelijk een "redelijk optimale" grootte van de partitie-elementen te bepalen. De grootte van een partitie-element is gelijk aan het aantal variabelen waarover een symmetrie zich uitstrekt. Bedenk dat kleine partitie-elementen (van twee of drie variabelen) de voorkeur genieten. Het aantal gates in het eerste OR veld, nodig om de maxtermen te genereren, blijft op deze manier beperkt.

5.5 Output phase optimization

Deze paragraaf bespreekt in het kort het probleem van de output phase optimization. Wat wordt met de output phase bedoeld? Een beperkt Σ_3-circuit berekent in het algemeen m uitgangsfuncties f_j, $j=1,\ldots,m$ met $f_j \in B_n$. Natuurlijk kan ook gezegd worden dat het beperkte Σ_3-circuit een functie $f \in B_{n,m}$ berekent. Vanwege de duidelijkere notatie is hier de eerste schrijfwijze gebruikt.
Het circuit berekent dus $(f_1,\ldots,f_m)$. In plaats van f_j kan het echter ook $\bar{f}_j$ berekenen. Later kan hieruit eenvoudig de functie f_j berekend worden met behulp van van een invertor (NOT gate). Als f_j dient als ingang van een ander beperkt Σ_3-circuit is de invertor zelfs overbodig. De keuze tussen f_j en $\bar{f}_j$ wordt de output phase, het teken (wel of geen ontkenning) van de uitgang, genoemd.
Berekenen van $\bar{f}_j$ in plaats van f_j kan leiden tot een beperkt Σ_3-circuit met een kleinere complexiteit. De keuze f_j of $\bar{f}_j$ is er voor alle $j=1,\ldots,m$. Dit levert een totaal van 2^m mogelijkheden voor de output phase van de m functies tesamen. Bij output phase optimization wordt geprobeerd hieruit die (of een) mogelijkheid te bepalen die het beperkte Σ_3-circuit met de kleinste complexiteit oplevert. Voor de oplossing van het output phase optimization probleem zijn verschillende algoritmes beschikbaar, zie bijvoorbeeld [28] of [29]. In dit verslag wordt het probleem niet verder uitgewerkt. In hoofdstuk 6 staat wel een voorbeeld waarin het nut van output phase optimization aan de orde komt.

6 Voorbeelden van realisaties van functies in Σ_2- en beperkte Σ_3-circuits

In de hoofdstukken 4 en 5 is het logisch ontwerp voor respectievelijk two-level en decoded PLA's besproken. In plaats van fysische PLA's is daarbij gebruik gemaakt van Σ_2- en beperkte Σ_3-circuits. Deze circuits vormen een model van de two-level en decoded PLA's. Ze kunnen tijdens het fysisch ontwerp, dat in paragraaf 2.2.3 is beschreven, omgezet worden in PLA's.
Dit hoofdstuk licht het logisch ontwerp uit de vorige twee hoofdstukken toe aan de hand van een tweetal voorbeelden. In het eerste voorbeeld (paragraaf 6.1) wordt een twee-bit optelfunctie gerealiseerd in een Σ_2- en een beperkt Σ_3-circuit. De optelfunctie is een functie uit de set $B_{4,3}$. Dit is een zogenaamde multiple-output functie. In paragraaf 6.2 staat het tweede voorbeeld. In dit voorbeeld wordt een Σ_2- en een beperkt Σ_3-circuit voor een exclusive or (EXOR) of pariteitsfunctie bepaald.
Het logisch ontwerp van een Σ_2-circuit is identiek aan het logisch ontwerp van een beperkt Σ_3-circuit met de triviale partitie. In hoofdstuk 5 zijn de logische ontwerpstap en alle problemen die daarbij optreden reeds besproken en met voorbeelden toegelicht. Dit hoofdstuk zet alle punten nog eens op een rijtje aan de hand van een tweetal voorbeelden. Het hoofdstuk bevat geen theorie. Voor een verklaring en uitleg van de gebruikte begrippen, definities en procedures wordt naar de vorige hoofdstukken verwezen.

6.1 Twee-bit optelfunctie

Deze paragraaf beschrijft de logische ontwerpstap waarin efficiënte Σ_2- en beperkte Σ_3-circuits ontworpen worden die een twee-bit optelfunctie realiseren. De twee-bit optelfunctie wordt ook in [29] door Sasao als voorbeeld gebruikt. De twee-bit optelfunctie maakt het optellen van twee, twee-bit getallen mogelijk. De getallen zijn x_1x_2 en x_3x_4. De binaire variabelen x_1 en x_3 zijn de meest significante. Voor de optelfunctie geldt : $2x_1+x_2+2x_3+x_4=4f_0+2f_1+f_2$.
Voor de optelfunctie $f=(f_0,f_1,f_2)$ geldt $f \in B_{4,3}$. De afzonderlijke functies f_0, f_1 en f_2 zijn functies uit de set $B_{4,1}$. De functie f_0 is de zogenaamde overflow. De functiewaardetabel van de optelfunctie ziet er als volgt uit.

Tabel 5 : Functiewaardetabel voor de optelfunctie

x_1	x_2	x_3	x_4	f_0	f_1	f_2	x_1	x_2	x_3	x_4	f_0	f_1	f_2
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	1	0	0	1	1	0	0	1	0	1	1
0	0	1	0	0	1	0	1	0	1	0	1	0	0
0	0	1	1	0	1	1	1	0	1	1	1	0	1
0	1	0	0	0	0	1	1	1	0	0	0	1	1
0	1	0	1	0	1	0	1	1	0	1	1	0	0
0	1	1	0	0	1	1	1	1	1	0	1	0	1
0	1	1	1	1	0	0	1	1	1	1	1	1	0

De som van mintermen (DNF) expressies voor f_0, f_1 en f_2 zijn :

$$f_0(x_1,x_2,x_3,x_4) = \bar{x}_1x_2x_3x_4+x_1\bar{x}_2x_3\bar{x}_4+x_1\bar{x}_2x_3x_4+x_1x_2\bar{x}_3x_4+$$
$$x_1x_2x_3\bar{x}_4+x_1x_2x_3x_4$$

$$f_1(x_1,x_2,x_3,x_4) = \bar{x}_1\bar{x}_2x_3\bar{x}_4+\bar{x}_1\bar{x}_2x_3x_4+\bar{x}_1x_2\bar{x}_3x_4+\bar{x}_1x_2x_3\bar{x}_4+$$
$$x_1\bar{x}_2\bar{x}_3\bar{x}_4+x_1\bar{x}_2\bar{x}_3x_4+x_1x_2\bar{x}_3\bar{x}_4+x_1x_2x_3x_4$$

$$f_2(x_1,x_2,x_3,x_4) = \bar{x}_1\bar{x}_2\bar{x}_3x_4+\bar{x}_1\bar{x}_2x_3x_4+\bar{x}_1x_2\bar{x}_3\bar{x}_4+\bar{x}_1x_2x_3\bar{x}_4+$$
$$x_1\bar{x}_2\bar{x}_3x_4+x_1\bar{x}_2x_3x_4+x_1x_2\bar{x}_3\bar{x}_4+x_1x_2x_3\bar{x}_4 \ .$$

Minimale expressies in de DF voor f_0, f_1 en f_2 zijn :

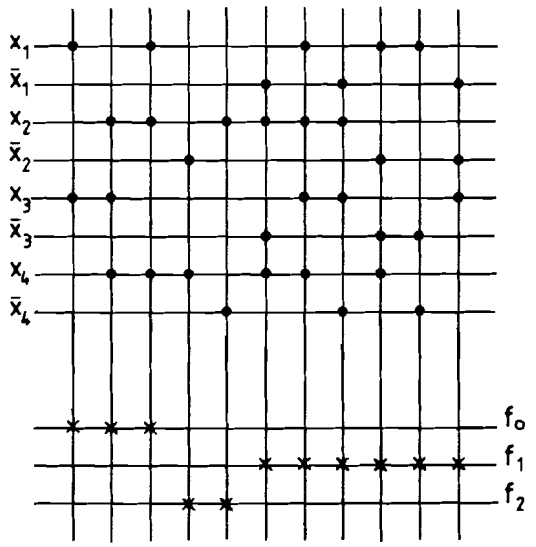
$$f_0(x_1,x_2,x_3,x_4) = x_1x_3+x_2x_3x_4+x_1x_2x_4$$

$$f_1(x_1,x_2,x_3,x_4) = \bar{x}_1x_2\bar{x}_3x_4+x_1x_2x_3x_4+\bar{x}_1x_2x_3\bar{x}_4+x_1\bar{x}_2\bar{x}_3x_4+$$
$$x_1\bar{x}_3\bar{x}_4+\bar{x}_1\bar{x}_2x_3$$

$$f_2(x_1,x_2,x_3,x_4) = \bar{x}_2x_4+x_2\bar{x}_4 \ .$$

Deze minimale expressies zijn bepaald met een algoritme voor de minimalisatie van multiple-output functies (zie paragraaf 4.4). Het aantal verschillende termen in de minimale expressies is 11. Een Σ_2-circuit dat deze functies realiseert is in figuur 12 getekend.
Het aantal gates i.e. de complexiteit, van het Σ_2-circuit is 11+3=14. Er zijn dus 14 AND en OR gates nodig om de optelfunctie te realiseren. Als een beperkt Σ_3-circuit gebruikt wordt als Σ_2-circuit dan wordt de complexiteit van het beperkte Σ_3-circuit gelijk aan 14+8=22. De 8 extra gates zijn nodig in het eerste OR veld om de ingangen van dit veld door te voeren naar

het AND veld. Het beperkte Σ_3-circuit gebruikt dan de triviale partitie (X_1,X_2,X_3,X_4) met $X_1=(x_1)$, $X_2=(x_2)$, $X_3=(x_3)$ en $X_4=(x_4)$ van $X=(x_1,x_2,x_3,x_4)$.

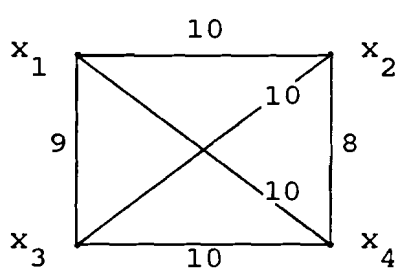


Figuur 12 : Minimaal Σ_2-circuit dat de twee-bit optelfunctie $f=(f_0,f_1,f_2)$ realiseert

Naast de triviale partitie kunnen ook andere partities gebruikt worden bij het beperkte Σ_3-circuit. Welke partitie en welke verdeling van de variabelen over de partitie-elementen moet men nemen? Dit is het in paragraaf 5.4. geformuleerde input variable assignment problem.
Als eerste moeten de getallen r en $n_1,\ldots,n_r$ bepaald worden. Voor r zijn er in het geval van de twee-bit optelfunctie, drie mogelijke waarden namelijk r=1, r=2 en r=3. De waarde r=4 levert de triviale partitie. Voor elke waarde van r kan men de bijbehorende vectoren $\underline{n}=(n_1,\ldots,n_r)$ bepalen. Voor dit eenvoudige voorbeeld van vier variabelen is het weinig werk om alle mogelijkheden op te schrijven. Men vindt voor :

- r=1 : $\underline{n}=(4)$.
- r=2 : $\underline{n}=(3,1)$ of $\underline{n}=(2,2)$.
- r=3 ; $\underline{n}=(2,1,1)$.

In paragraaf 5.3.4 is uitgelegd dat de bovengrens van de complexiteit afhankelijk is van $\underline{n}$. De vector $\underline{n}=(2,2)$ zal, volgens de redenering in paragraaf 5.3.4 voor willekeurige functies, de kleinste bovengrens opleveren. De vector $\underline{n}=(2,1,1)$ levert welliswaar dezelfde bovengrens maar het kan nooit kwaad twee partitie-elementen met ieder een variabele samen te voegen tot een partitie-element met twee variabelen (zie paragraaf 5.3.4).
In het vervolg wordt uitgegaan van een partitie (X_1,X_2) met $d(X_1)=d(X_2)=2$ voor de optelfunctie.
Hoe moeten de vier variabelen x_1, x_2, x_3 en x_4 over de partitie-elementen X_1 en X_2 verdeeld worden? Voor het oplossen van dit probleem wordt in dit voorbeeld gebruikt gemaakt van de in paragraaf 5.4.2 beschreven methode van Sasao [29]. Uitgaande van de reeds eerder gegeven minimale expressies voor f_0, f_1 en f_2 kan men de volgende assignment graaf opstellen.



De takken met de waardes 8 en 9 hebben de kleinste som. De partitie die men op grond hiervan kiest, is (X_1,X_2) met $X_1=(x_1,x_3)$ en $X_2=(x_2,x_4)$. De GBE's in de DNF voor de optelfunctie zien er als volgt uit :

$$f_0(X_1,X_2) = X_1^{(01)}X_2^{(11)}+X_1^{(11)}X_2^{(00)}+X_1^{(11)}X_2^{(01)}+X_1^{(10)}X_2^{(11)}+$$
$$X_1^{(11)}X_2^{(10)}+X_1^{(11)}X_2^{(11)}$$

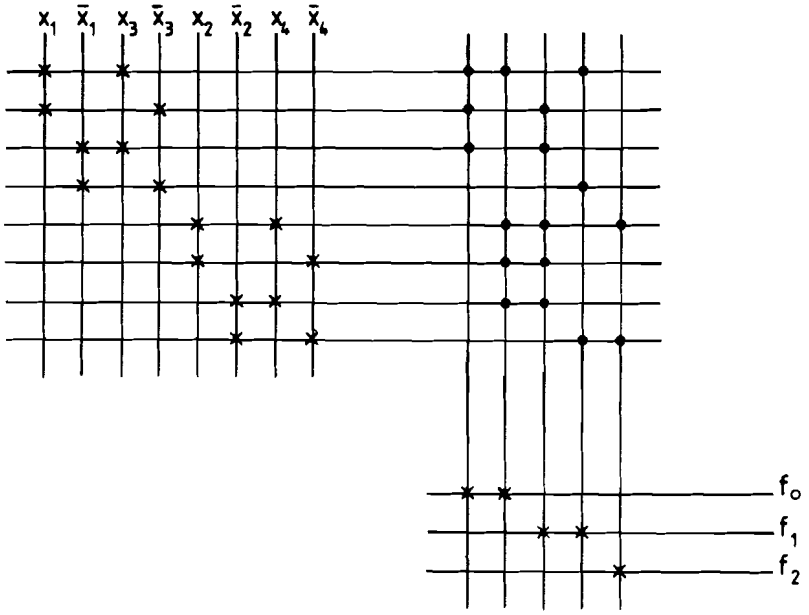
$$f_1(X_1,X_2) = X_1^{(01)}X_2^{(00)}+X_1^{(01)}X_2^{(01)}+X_1^{(00)}X_2^{(11)}+X_1^{(01)}X_2^{(10)}+$$
$$X_1^{(10)}X_2^{(00)}+X_1^{(10)}X_2^{(01)}+X_1^{(10)}X_2^{(10)}+X_1^{(11)}X_2^{(11)}$$
$$f_2(X_1,X_2) = X_1^{(00)}X_2^{(01)}+X_1^{(01)}X_2^{(01)}+X_1^{(00)}X_2^{(10)}+X_1^{(01)}X_2^{(10)}+$$
$$X_1^{(10)}X_2^{(01)}+X_1^{(11)}X_2^{(01)}+X_1^{(10)}X_2^{(10)}+X_1^{(11)}X_2^{(10)} .$$

Een stelsel van minimale GBE's voor deze partitie is :

$$f_0(X_1,X_2) = X_1^{(11)}+X_1^{\{01,10,11\}}X_2^{(11)}$$
$$f_1(X_1,X_2) = X_1^{\{00,11\}}X_2^{(11)}+X_1^{\{01,10\}}X_2^{\{00,01,10\}}$$
$$f_2(X_1,X_2) = X_2^{\{01,10\}} .$$

Het beperkte Σ_3-circuit dat bij deze expressies hoort is in figuur 13 getekend. In het eerste OR veld worden maxtermen gegenereerd. Als men bijvoorbeeld de literal $X_1^{(11)}$ wil realiseren dan heeft men hiervoor drie ingangen op de AND gate nodig. Een en ander is in paragraaf 5.1 reeds toegelicht. Het aantal gates in figuur 13 is 8+5+3=16. Ten opzichte van de triviale partitie heeft men een winst van 22-16=6 (AND) gates.

Ten overvloede misschien nog de volgende opmerking. Stel dat men een partitie met een vector $\underline{n}=(3,1)$ had gekozen. Het aantal OR gates in het eerste OR veld is dan $2^3+2^1=10$. Samen met de drie gates in het tweede OR veld levert dit 13 gates. In het AND veld zijn in dit geval minstens drie gates nodig. Dit is eenvoudig in te zien. Er moeten drie verschillende functies (expressies) gerealiseerd worden. Daar zijn minstens twee verschillende termen voor nodig. De expressie van de functie f_2 bestaat uit een term. Deze term kan in geen van de andere twee minimale expressies gebruikt worden. Er zijn nu nog minstens twee andere termen nodig om deze twee expressies verschillend te maken. Dit geeft een



Figuur 13 : Beperkt Σ_3-circuit met optimale input assignment voor de twee-bit optelfunctie

totaal van minstens drie AND gates. Het minimale aantal gates dat men bij bij $\underline{n}$=(3,1) nodig heeft, is dus 13+3=16. Gebruik van $\underline{n}$=(3,1) in plaats van $\underline{n}$=(2,2) levert dus geen kleiner circuit voor de optelfunctie. Beslissingen omtrent de input assignment die genomen zijn met behulp van bovengrezen, leiden dus tot een optimaal circuit.

Tot slot wordt nog de output phase optimization (zie paragraaf 5.5) behandeld. Het circuit in figuur 13 berekent (f_0,f_1,f_2). Het is echter ook mogelijk een of meer ontkenningen te berekenen. Zo kan men bijvoorbeeld een beperkt Σ_3-circuit ontwerpen dat $(f_0,f_1,\bar{f}_2)$ berekent. In het geval van drie uitgangsfuncties bij de optelfunctie zijn er al 2^3=8 mogelijkheden voor de output phase. Sasao presenteert in [29] een algoritme waarmee een (bijna) optimale output phase gevonden kan worden. In [29] vindt men ook de uitwerking van het probleem voor de twee-bit optelfunctie. In dit verslag is niet nader ingegaan op dit

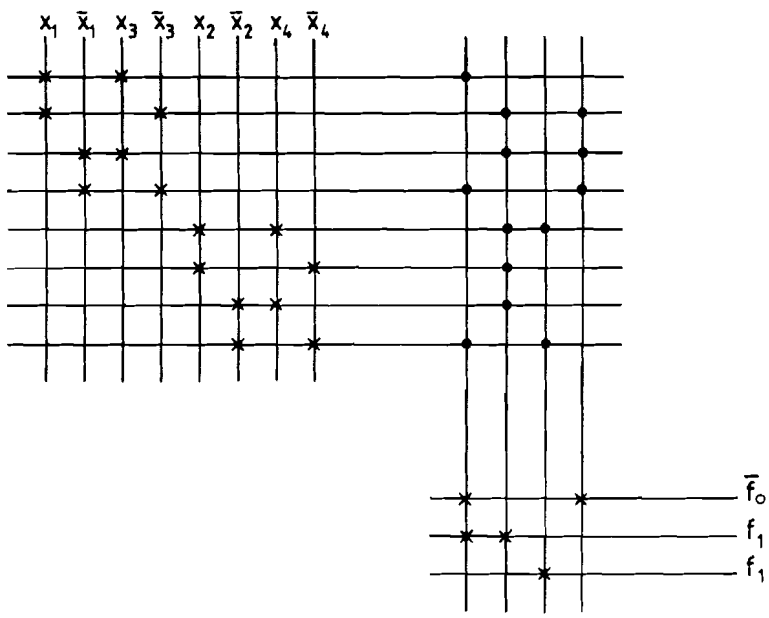
algoritme. Toch wordt hier ter illustratie het met het algoritme van Sasao bereikte resultaat gegeven. Het blijkt dat $(\bar{f}_0, f_1, f_2)$ (een van) de output phase(s) is die leiden tot een minimaal beperkt Σ_3-circuit. De bijbehorende minimale GBE's in de DF zijn :

$$\bar{f}_0(X_1,X_2) = X_1^{\{01,10\}} X_2^{\{00,01,10\}} + X_1^{(00)}$$

$$f_1(X_1,X_2) = X_1^{\{01,10\}} X_2^{\{00,01,10\}} + X_1^{\{00,11\}} X_2^{(11)}$$

$$f_2(X_1,X_2) = X_2^{\{01,10\}} .$$

Het bijbehorende beperkte Σ_3-circuit is in figuur 14 afgebeeld.



Figuur 14 : Beperkt Σ_3-circuit met optimale input assignment en output phase optimization voor een twee-bit optelfunctie.

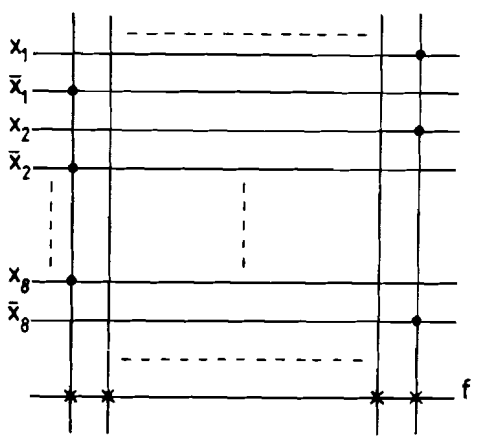
Het aantal gates in figuur 14 is 8+4+3=15. Ten opzichte van het beperkte Σ_3-circuit met de triviale partitie dat gebruikt wordt als Σ_2-circuit, levert dit een winst van 22-15=7 gates i.e. een besparing van zo'n 32%. Als men alleen het AND veld beschouwt, wordt een reductie van 11-4=7 AND gates gevonden.
Omdat het aantal ingangen van het AND veld gelijk blijft, is dit in dit geval ook een indicatie voor de oppervlaktewinst van een fysische decoded PLA ten opzichte van een two-level PLA. Deze winst bedraagt afgezien van array folding en array partitioning zo'n 64%.

In het bovenstaande voorbeeld is duidelijk geworden dat het gebruik van niet triviale partities, input variable assignment en output phase optimization leidt tot beperkte Σ_3-circuits met een aanzienlijk kleinere complexiteit dan beperkte Σ_3-circuits met de triviale partitie i.e. een Σ_2-circuit.

6.2 Pariteitsfunctie

Het tweede voorbeeld uit dit hoofdstuk behandelt een pariteitsfunctie. De pariteitsfunctie wordt ook wel de exclusive or functie genoemd. De functie is een element van de set B_n. In tegenstelling tot de vorige paragraaf heeft deze functie dus maar een "uitgang". De pariteitsfunctie levert de functiewaarde 1 dan en slechts dan als het aantal variabelen die de waarde 1 hebben, oneven is. Een expressie voor een pariteitsfunctie is $f(x_1,\ldots,x_n)=x_1 \oplus x_2 \oplus \ldots \oplus x_n$. De reden waarom de expressie niet in de DNF geschreven is, is het feit dat het aantal mintermen van een pariteitsfunctie nogal groot is. Het aantal mintermen in een normale Booleaanse expressie voor een pariteitsfunctie van n variabelen is gelijk aan $2^{(n-1)}$. Voor een functie van acht variabelen is het aantal reeds $2^7=128$.

In deze paragraaf worden een Σ_2- en een beperkt Σ_3-circuit gegeven voor een pariteitsfunctie f van acht binaire variabelen i.e. $f(x_1,\ldots,x_8)=x_1\oplus\ldots\oplus x_8$. De Booleaanse expressie voor f in de DNF bevat 128 mintermen. Ook een minimale expressie (triviale partitie) voor f bevat 128 termen. De minimale expressie in de DF is identiek aan de som van mintermen expressie. Er is geen minimalisatie mogelijk. Vanwege het grote aantal termen zijn deze expressies hier niet vermeld. In figuur 15 is (een gedeelte van) een minimaal Σ_2-circuit getekend dat de functie f realiseert. Het aantal gates in het circuit is $2^7+1=129$. Dit aantal kan ook verkregen worden met de formule uit paragraaf 5.3.3 voor het aantal gates in een beperkt Σ_3-circuit met de waarden r=8 en q=1. Van het met de formule gevonden aantal gates moet men dan echter het aantal gates van het eerste OR veld (16) aftrekken. Als straks het Σ_2-circuit met een beperkt Σ_3-circuit vergeleken wordt, gaat men toch uit van het met deze formule gevonden aantal gates (16+128+1=145) voor de triviale partitie.



Figuur 15 : Minimaal Σ_2-circuit voor de pariteitsfunctie

Net als in de vorige paragraaf wordt bekeken hoe de pariteitsfunctie met een beperkt Σ_3-circuit berekend kan worden.

De formule $\sum_{i=1}^{r} 2^{n_i+2^{(r-1)}}$ uit paragraaf 5.3.3 geeft het aantal gates dat nodig is voor het realiseren van een pariteitsfunctie met een partitie $(X_1,\ldots,X_r)$ met $d(X_i)=n_i$.
Ook hier moet het input variable assignment probleem worden opgelost. Zoals in paragraaf 5.4 is toegelicht, valt dit probleeem uiteen in twee delen namelijk :

- Het vinden van de optimale grootte van de partitie-elementen i.e. bepalen van $\underline{n}=(n_1,\ldots,n_r)$.
- Het vinden van een optimale verdeling van de variabelen over de partitie-elementen.

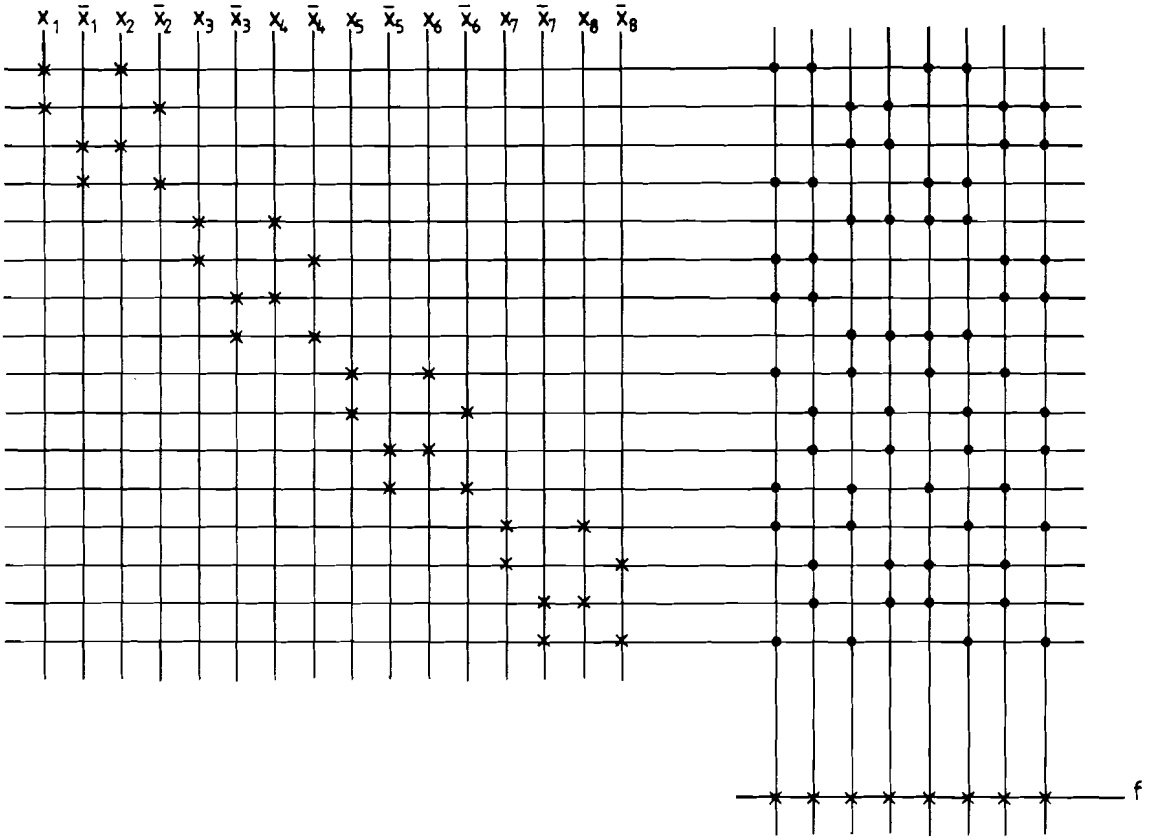
Het laatste probleem zal als eerste uitgewerkt worden. In paragraaf 5.4.2 staat dat Ektare en Al-Sheakhly beweren dat de verdeling er niet toe doet voor functies zonder symmetrie. De pariteitsfunctie bevat wel degelijk een vorm van symmetrie volgens de methode van Ektare en Al-Sheakhly. Het bijzondere van de pariteitsfunctie is echter dat de symmetrie met betrekking tot alle paren van variabelen gelijk is. De pariteitsfunctie verandert niet als (groepen van) variabelen gepermuteerd worden. Bij de pariteits functie is er geen enkele groep van variabelen aan te wijzen die "meer" symmetrie bevat dan een andere groep. Op grond hievan is het geoorloofd aan te nemen dat de verdeling van de variabelen over de partitie-elementen geen invloed heeft op de grootte van een beperkt Σ_3-circuit voor een pariteitsfunctie.

Het assignment probleem bestaat in dit geval alleen uit het vinden van een "goede" vector $\underline{n}$. Uit de formule voor het aantal gates voor de pariteitsfunctie (zie paragraaf 5.3.3 of hierboven) blijkt dat dit aantal afhangt van r en de n_i. Het aantal gates zal minimaal zijn als (r-1) en de n_i in de dezelfde orde van grootte liggen. Voor n=8 levert $\underline{n}=(2,2,2,2)$ het kleinste aantal gates (zie ook [27]) namelijk 16+8+1=25. Een minimale expressie voor f met de partitie (X_1,X_2,X_3,X_4) met $X_1=(x_1,x_2)$, $X_2=(x_3,x_4)$, $X_3=(x_5,x_6)$ en $X_4=(x_7,x_8)$ ziet er als volgt uit :

$$f(X_1,X_2,X_3,X_4) = X_1^{\{01,10\}}X_2^{\{00,11\}}X_3^{\{01,10\}}X_4^{\{01,10\}} +$$
$$X_1^{\{01,10\}}X_2^{\{00,11\}}X_3^{\{00,11\}}X_4^{\{00,11\}} +$$
$$X_1^{\{00,11\}}X_2^{\{01,10\}}X_3^{\{01,10\}}X_4^{\{01,10\}} +$$
$$X_1^{\{00,11\}}X_2^{\{01,10\}}X_3^{\{00,11\}}X_4^{\{00,11\}} +$$
$$X_1^{\{01,10\}}X_2^{\{01,10\}}X_3^{\{01,10\}}X_4^{\{00,11\}} +$$
$$X_1^{\{01,10\}}X_2^{\{01,10\}}X_3^{\{00,11\}}X_4^{\{01,10\}} +$$
$$X_1^{\{00,11\}}X_2^{\{00,11\}}X_3^{\{01,10\}}X_4^{\{00,11\}} +$$
$$X_1^{\{00,11\}}X_2^{\{00,11\}}X_3^{\{00,11\}}X_4^{\{01,10\}} .$$

Ook hier is geen GBE in de DNF gegeven omdat deze veel (128) termen heeft. In figuur 16 is een beperkt Σ_3-circuit getekend dat de pariteitsfunctie berekent. Er bestaat geen beperkt Σ_3-circuit met een kleinere complexiteit. Het circuit uit figuur 16 bestaat uit 25 gates. Een beperkt Σ_3-circuit met de triviale partitie bestaat uit 145 gates (zie terug). De besparing is dus 120 gates. Het circuit in figuur 16 is dan zo'n 83% minder complex dan een circuit met de triviale partitie.

Voor de vector $\underline{n}$ is hier (2,2,2,2) genomen. Deze keuze is gemaakt omdat men op voorhand wist dat de functie een pariteitsfunctie was en de complexiteit van een beperkt Σ_3-circuit voor de pariteitsfunctie bekend is (zie paragraaf 5.3.3). Als op voorhand niet bekend is voor welke functie een circuit geconstrueerd moet worden of de complexiteit niet exact bekend is dan is de beste keuze $\underline{n}$=(4,2,2). Deze keuze wordt gemaakt op grond van de algemene bovengrens (paragraaf 5.3.4). De complexiteit van een beperkt Σ_3-circuit met deze partitie is 29 gates. Dit is slechts 4 meer dan bij het optimale circuit.



Figuur 16 : Minimaal beperkt Σ_3-circuit voor de pariteitsfunctie

Voor een pariteitsfunctie van zestien variabelen is dit verschil echter aanzienlijk groter. De optimale $\underline{n}$ voor deze functie is (4,3,3,3,3). Voor een willekeurige functie had men echter $\underline{n}$=(8,2,2,2,2) gekozen. De complexiteiten van een beperkt Σ_3-circuit dat de pariteitsfunctie van zestien variabelen berekent met deze vectoren zijn respectievelijk 65 en 289. Deze getallen worden verkregen door de n_i van de vectoren in te vullen in de formule voor de pariteitsfunctie in paragraaf 5.3.3.

In dit hoofdstuk zijn twee functies met zowel een Σ_2-circuit (beperkt Σ_3-circuit met de triviale partitie) als met een beperkt Σ_3-circuit berekend. In beide voorbeelden is de

complexiteit van het laatst genoemde circuit kleiner. Dit blijkt zelfs zo te zijn als geen moeite gedaan wordt om een optimale input assignment en/of output phase te bepalen.

7 Conclusies en aanbevelingen

Tot slot volgen in het laatste hoofdstuk van dit verslag de conclusies en aanbevelingen. In dit hoofdstuk komen drie punten aan de orde. Als eerste wordt over twee laags en decoded PLA's, Σ_2- en beperkte Σ_3-circuits gesproken. Daarna wordt nader ingegaan op Σ_k-circuits voor k>3. Tot slot wordt aandacht besteed aan verschillende aspecten met betrekking tot complete sets van (basis)functies.

Programmable logic arrays (PLA's) zijn een manier om combinatorische logica te implementeren. Ze bezitten een regelmatige structuur. Dit maakt het gebruik van automatische ontwerphulpmiddelen bij het ontwerp van VLSI i.c.'s mogelijk.
Bij het ontwerpen van PLA's heeft men een abstracte beschrijvingsvorm nodig. Deze is gevonden in de vorm van Σ_k-circuits. De Σ_2-circuits blijken een model te zijn voor de normale, bekende twee laags (two-level) PLA's. Een functie die door een PLA gerealiseerd wordt, kan met een Booleaanse expressie beschreven worden. Als deze expressie in de DF (disjunctive form) staat is een één op één afbeelding mogelijk tussen de expressies en de Σ_2-circuits. Er zijn algoritmes om een expressie in de DF met een minimaal aantal termen en literals per term te bepalen. Deze minimale expressies leiden tot Σ_2-circuits met een minimale complexiteit.

Als k i.e. de diepte van het Σ_k-circuit groter wordt, neemt de complexiteit in het algemeen af. In plaats van circuits voor willekeurige k te beschouwen, is in dit verslag eerst naar circuits met k=3 gekeken. Voor een speciale klasse van deze circuits de zogenaamde beperkte Σ_3-circuits, is een één op één afbeelding op gegeneraliseerde Booleaanse expressies (GBE's) in de DF mogelijk. GBE's zijn een veralgemenisering van de normale

Booleaanse expressies. De algoritmes die gebruikt worden om normale Booleaanse expressies te minimaliseren, kunnen worden uitgebreid om GBE's te minimaliseren. Beperkte Σ_3-circuits zijn in de meeste gevallen al aanzienlijk minder complex dan Σ_2-circuits. Onderzoek naar logische ontwerpmethodes voor beperkte Σ_3-circuits is daarom aan te bevelen.

Bij de minimalisatie van GBE's treedt het covering problem op. Voor monotone en unate functies bestaat dit probleem niet in het geval van normale expressies (GBE's met de triviale partitie). Alle prime implicants van deze functies zijn immers essentieel. De vraag rijst of iets dergelijks ook geldt voor prime entailants in het geval van minimalisatie van GBE's. Enige aandacht voor het antwoord op deze vraag is geboden omdat de voor de minimalisatie benodigde rekentijd erdoor verkleind kan worden (zie bijvoorbeeld Rudell et al. [26]).

Een ander probleem dat optreedt bij GBE's is het input variable assignment problem. Daarbij moet de optimale grootte van en de verdeling van de variabelen over de partitie-elementen worden vastgesteld. Het is onmogelijk voor iedere functie snel en exact aan te geven wat de complexiteit is als functie van de partitie-element grootte en de verdeling van de variabelen. Voor het verkrijgen van een oplossing van het assignment problem worden daarom heuristische algoritmes gebruikt [31] die werken met bovengrenzen voor de circuit-complexiteit. Oplossingen met de kleinste bovengrens hebben de voorkeur. Omdat met bovengrenzen gewerkt wordt, is niet gegarandeerd dat zulke oplossingen de meest optimale zijn. Het is interessant te onderzoeken hoe optimaal deze oplossingen zijn.
Het berekenen van de bovengrenzen neemt nogal wat rekentijd in beslag. Daarom is een ander heuristisch algoritme gebruikt. Dit algoritme werkt [29] met schattingen voor deze bovengrenzen en vergt minder rekentijd. Een nadeel is dat het algoritme alleen werkt voor partitie-elementen met twee variabelen.

Het verdient aanbeveling een heuristiek voor de input variable assignment te ontwikkelen omdat een (bijna) optimale assignment leidt tot een kleinere complexiteit van de beperkte Σ_3-circuits. Zo'n heuristiek zou bijvoorbeeld een uitbreiding kunnen zijn van het algoritme uit [29] dat met schattingen werkt. De uitbreiding moet er voor zorgen dat de bovengrens ook snel geschat kan worden indien er partitie-elementen van drie of meer variabelen zijn. Daarbij kan eventueel gebruik gemaakt worden van symmetrieën in de te realiseren functie. De symmetrieën kunnen bijvoorbeeld gedetecteerd worden met spectrale technieken.

Behalve input variable assignment wordt output phase optimization gebruikt om de complexiteit van beperkte Σ_3-circuits te verkleinen. Bij de in dit verslag gegeven voorbeelden wordt eerst een assignment bepaald en daarna de output phase geöptimaliseerd. Misschien is het echter wel zo dat voor een andere assignment wordt gekozen als de output phase van te voren bekend is. De vraag hierbij is dus eigenlijk of input variable assignment en output phase optimization onafhankelijk zijn van elkaar. Als dit niet zo is, is het wellicht voordelig om algoritmes voor input variable assignment en output phase optimization te integreren.

Naast de beperkte Σ_3-circuits zijn er Σ_3-circuits zonder beperkingen in het eerste OR veld. De restrictie dat iedere ingangsvariabele maar in een partitie-element zit, komt te vervallen. Een variabele die in meer dan een partitie-element voorkomt, wordt door Fleisher en Maissel [7] een "split variable" genoemd. Fleisher en Maissel merken op dat een split variable vaak niet leidt tot een kleinere complexiteit maar het wel voor kan komen. Het zou onderzocht moeten worden wat de invloed van split variables is op de complexiteit van Σ_3-circuits. Indien zou blijken dat ze leiden tot circuits met een kleinere complexiteit dan moet bekeken worden of en hoe bestaande algoritmes voor GBE's aan te passen zijn voor het geval van split variables.

Voor Σ_2- en beperkte Σ_3-circuits zijn er expressies met een bepaalde vorm bekend die één op één zijn af te beelden op deze circuits. Daarnaast zijn er minimalisatie algoritmes voor deze expressies. Een minimale expressie leidt tot een circuit met minimale complexiteit. Voor echte Σ_3- of Σ_k-circuits (k>3) is op dit moment noch bekend hoe de expressies met een één op één afbeelding eruit zien noch hoe hiermee minimale circuits ontworpen kunnen worden.
Men kan zich afvragen of het zinvol is ontwerpmethodieken voor Σ_k-circuits met k>3 te bepalen. Er zijn de schrijver van dit verslag onvoldoende gegevens bekend om een aanbeveling te doen over het wel of niet ontwikkelen van zulke methodieken. Daarom wordt hier volstaan met het geven van enkele gedachten die bij de beantwoording van deze vraag een rol spelen.

Is het zinvol Σ_k-circuits met drie of meer niveaus te gebruiken? In dit verslag is in paragraaf 2.2.2 al aangegeven dat Fleisher en Maissel [7] het volgende opgemerkt hebben. Circuits met een diepte van drie tot zes hebben de kleinste complexiteit. Afname van de diepte (aantal lagen) leidt tot een significante toename van de complexiteit terwijl toename van de diepte nauwelijks een kleinere complexiteit tot gevolg heeft.
Het samendrukken van een circuit (kleinere diepte) leidt dus soms tot een explosieve groei. Hoe is dit te verklaren? Wegener introduceert in [38] het begrip size-depth(poly,const). De "size" is de complexiteit en de "depth" is de diepte van een circuit. Een functie $f \in B_n$ zit in de klasse van functies size-depth(poly,const) als f berekend kan worden door een circuit met een onbeperkte fan-in met een complexiteit die kleiner is dan cn^q en een diepte d. De complexiteit is dus een polynomiale functie van het aantal variabelen van de te realiseren functie f.
Het blijkt dat vele fundamentele functies niet in size-depth(poly,const) zitten. Daarom zou men [38] circuits moeten gebruiken die dieper zijn (meer niveaus) naarmate het aantal ingangen (variabelen) n toeneemt.

Hieronder wordt het begrip size-depth(poly,const) nader toegelicht. Een voorbeeld van een functie die wel in size-depth(poly,const) zit is de twee-bit optelfunctie. Algemeen zit [38] de optelfunctie van twee p-bit getallen in size-depth(poly,const). Het aantal ingangen van deze optelfuncties is n=2p. Omdat de twee-bit optelfunctie in size-depth(poly,const) zit, is de complexiteit altijd begrensd door een polynoom van de vorm cn^q. Voor de twee-bit optelfunctie is n=4 dus het polynoom is $c4^q$. Welke circuit diepte men ook kiest, de complexiteit van het Σ_k-circuit is altijd begrensd door een polynoom. Het aantal gebruikte gates i.e. de complexiteit zal in het algemeen groter worden naarmate de diepte kleiner wordt (lagere waarde voor k). Dit is bijvoorbeeld zo voor de twee-bit optelfunctie in hoofdstuk 6. De complexiteit van een beperkt Σ_3-circuit met de triviale partitie (Σ_2-circuit) is groter dan de complexiteit van een circuit met een niet triviale partitie.

De pariteitsfunctie is een voorbeeld van een functie die niet in size-depth(poly,const) zit (zie [38]). De complexiteit van een Σ_k-circuit dat de pariteitsfunctie realiseert is meestal niet begrensd door een polynoom in n (het aantal variabelen). Wegener geeft in [38] aan hoe groot de minimale diepte k van het Σ_k-circuit moet zijn opdat de complexiteit nog wel begrensd is door een polynoom. Deze minimale diepte stijgt als functie van n. Dit is in overeenstemming met bovenstaande opmerking dat men circuits moet gebruiken die dieper zijn naarmate het aantal ingangen groeit. Stel dat het aantal ingangsvariabelen (n) van de pariteitsfunctie gegeven is. Tevens is er een waarde voor k gegeven waarvoor de complexiteit van het Σ_k-circuit dat de pariteitsfunctie berekent, begrensd is door een polynoom in n. Als k kleiner wordt, neemt de complexiteit toe. In eerste instantie is de complexiteit nog begrensd door een polynoom. Vanaf een bepaalde waarde k_0 voor k stijgt de complexiteit echter exponentieel. Omgekeerd geldt dat de complexiteit eerst sterk afneemt bij toename van k. Vanaf $k=k_0$ zal de afname minder sterk zijn.

In hoofdstuk 6 blijkt dat de afname van de complexiteit voor de pariteitsfunctie, bij overgang van de triviale partitie naar een niet triviale partitie, relatief gezien veel groter is dan voor de twee-bit optelfunctie (83% tegen respectievelijk 32%). Een en ander is in overeenstemming met de hier gegeven theorie. De minimale diepte (k_0) voor een pariteitsfunctie van acht variabelen lijkt groter te zijn dan drie. Bij twee en drie niveaus is de complexiteit van deze functie nog niet begrensd door een polynoom. Ook de opmerking van Fleisher en Maissel lijkt hiermee verklaard te zijn.
Een gedeeltelijk antwoord op de eerder gestelde vraag of meer niveaus zinvol zijn, luidt dan als volgt. Circuits met meer niveaus zijn vooral zinvol voor functies die niet in size-depth(poly,const) zitten en van een relatief groot aantal variabelen afhankelijk zijn. Hoe vaak zulke functies voorkomen en hoe groot het "kritische" aantal variabelen is, zal voor vele praktische gevallen onderzocht moeten worden.

Het is al eerder gezegd dat niet bekend is hoe (als ze al bestaan) expressies en methodes voor minimalisatie er uit zien voor Σ_k-circuits (k>3). Een ontwerpmethode voor deze circuits zou kunnen bestaan uit het splitsen van een functie in meerdere andere functies. Splitsen kan bijvoorbeeld geschieden met algebraïsche werktuigen zoals "weak division" en "strong division". De afgesplitste deelfuncties zouden dan in een gedeelte van de beschikbare niveaus gerealiseerd worden. Daarna worden de deelfuncties in de resterende niveaus samengevoegd tot de oorspronkelijke functie. Afgesplitste functies zouden bij voorkeur in size-depth(poly,const) moeten zitten of anders in een niet te gering aantal niveaus gerealiseerd moeten worden. Het moet bekeken worden of deze afgesplitste deelfuncties, unate functies of misschien unate functies met een bepaalde eigenschap mogen zijn. Een reden hiervoor is dat men al enigszins bekend is met het afsplitsen van unate functies. Zie daarvoor bijvoorbeeld Brayton et al. in [3].

Tot slot de volgende opmerking. Het zal niet zo zijn dat een functie die niet in size-depth(poly,const) zit, op te splitsen is in functies die er wel in zitten. Het is [38] met "niet in size-depth(poly,const) zitten" net als met het NP compleet zijn van problemen. Het zijn beide concepten voor "reduceerbaarheid". Als een probleem A reduceerbaar is tot probleem B dan zal het oplossen van probleem B niet veel moeilijker (complexer) zijn dan het oplossen van probleem A. Als een functie niet in size-depth(poly,const) zit dan is de complexiteit van een circuit dat de functie berekend groot. Opsplitsen van de functie in enkele andere zal deze complexiteit niet drastisch verkleinen of vergroten. "Verkeerd" opsplitsen zal tot een toename van de comlexiteit leiden.

Het laatste punt dat in dit hoofdstuk wordt behandeld, gaat over de set van basisfuncties. Bij Σ_k-circuits (en dus bij PLA's) heeft men alleen de beschikking over een set van drie functies {AND,OR,NOT} (zie paragraaf 3.2.3). De NOT gate is ook bij Σ_k-circuits nodig om de ontkenningen van variabelen op niveau 0 te bepalen.
Heeft het nut de bovenstaande complete set uit te breiden? Zo'n uitbreiding kan bijvoorbeeld bestaan uit het toevoegen van de EXOR (pariteitsfunctie). Volgens de beschouwing in paragraaf 3.2.3 heeft een circuit met de set {AND,OR,NOT} een complexiteit die een tot vijf maal groter en een diepte die een tot drie maal groter is dan in het geval de set {EXOR,AND,OR,NOT} gebruikt wordt. De beschouwing in paragraaf 3.2.3 geldt voor willekeurige circuits en gates met twee ingangen. Als Σ_k-circuits gebruikt worden, zullen de complexiteit en diepte eerder toenemen dan afnemen vanwege de voorgeschreven volgorde van AND- en OR gates en het grotere aantal ingangen van de EXOR gates. Als bovendien de diepte vastligt (bijvoorbeeld k=2 of k=3) dan kan de complexiteit nog eens extra toenemen als de beschreven functie niet in size-depth(poly,const) zit. Het is dus waarschijnlijk voordelig de EXOR in de set op te nemen. Realiseren van de

pariteitsfunctie is nu in ieder geval geen probleem meer.
Het is nu niet zo dat de complexiteit van Ω-circuits met Ω={EXOR,AND,OR,EXOR} voor alle functies die niet in size-depth(poly,const) zitten, begrensd is door een polynoom. Toevoegen van één functie (EXOR) die niet in size-depth(poly,const) zit is dus geen garantie om altijd een circuit te krijgen met een complexiteit die door een polynoom begrensd is.
Hoe is het een en ander fysisch te realiseren? Cho stelt in [4] een drie laags structuur voor waarin naast AND- en OR gates ook EXOR gates voorkomen. Het geheel lijkt op een drie laags (three-level) PLA. De eerste OR laag is echter vervangen door een laag met EXOR gates. Realisatie van een EXOR gate vergt meer transistoren dan realisatie van een OR gate. Het zal bekeken c.q. proefondervinderlijk vastgesteld moeten worden of de extra transistoren gecompenseerd worden door een afname van transistoren in het AND veld. Cho beschrijft in [4] hoe men minimale EXOR-AND-OR PLA's kan bepalen. In de beschrijving blijkt dat het ontwerp voor EXOR-AND-OR PLA's overeenkomt met het ontwerp van AND-OR PLA's. Er wordt echter een "ingangstransformatie" toegepast. Hierop wordt hier niet verder ingegaan. Het is echter interessant te kijken of er overenkomsten zijn tussen deze methode en de methode voor het logisch ontwerp van beperkte Σ_3-circuits.

Naast de EXOR kunnen ook andere functies in de set van basisfuncties worden opgenomen. Gates die deze functies realiseren, vergen in het algemeen meer transistoren dan AND- en OR gates. Het toevoegen van een functie aan de set van basisfuncties heeft dus alleen zin als de extra transistoren gecompenseerd worden door een kleinere complexiteit. Een en ander moet telkens opnieuw worden afgewogen.
Tot slot de opmerking dat ook multiple-valued logic gebruikt kan worden om PLA's te implementeren. De set van basisfuncties gaat er dan heel anders uitzien (zie paragraaf 3.2.1). Het zou

onderzocht moeten worden of deze functies met de huidige technologieën te maken zijn. Verwacht wordt dat met multiple-valued logic PLA's in de toekomst efficiënte circuits gebouwd kunnen worden.

Samengevat kan men zeggen dat de complexiteit van Σ_k-circuits afneemt naarmate k groter wordt. Het probleem is het vinden van een geschikte ontwerpmethode. Voor Σ_2- en beperkte Σ_3-circuits bestaan deze methodes. Hoe methodes voor Σ_k-circuit met $k \geq 3$ eruit zien is nog niet (voldoende) bekend. Het moet onderzocht worden welke waarde van k leidt tot de meest optimale circuits. Voor de bepaling van een optimale k zijn criteria nodig zoals kosten en prestaties. Deze criteria moeten op basis van gelijkheid behandeld worden (zie hoofdstuk 2). Op deze manier zullen ook circuits met k>2 een kans krijgen.

Literatuur

[1] Biswas N.N.
Foldable compatibility matrix for the folding of programmable logic arrays. Int. J. Electron. (GB), Vol. 61, No. 1, July, 1986, p. 97-103

[2] Biswas N.N., Jacob J.
A testable PLA design with minimal hardware and test set. International Test Conference 1985 Proceedings: The Future of Test. Philadelphia, PA, USA, 19-21 Nov. 1985. Washington, DC, USA: IEEE Comput. Soc. Press, 1985, p. 583-588

[3] Brayton R.K., Hachtel G.D., McMullen C.T.,
Sangiovanni-Vincentelli A.L.
Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers, Boston, 1984 (The Kluwer international series in engineering and computer science consulting editor : Allen J.)

[4] Cho D.S., Hwang H.Y.
On the three-level PLA for logic minimization. Conference Proceedings IEEE SOUTHEASTCON 85. Rayleigh, NC, USA, 31 March-3 April 1985. New York, USA: IEEE, 1985, p. 366-370

[5] Coli V.J.
Introduction to programmable array logic. Byte (USA), Vol. 12, No. 1, Jan., 1987, p. 207-219

[6] Ektare A.B., Al-Sheakhly M.K.H.
Function symmetries and decoded-PLA realization. Int. J. Electron. (GB), Vol. 60, No. 6, June, 1986, p. 691-707

[7] Fleisher H., Maissel L.I.

An introduction to array logic. IBM J. Res. Develop., Vol. 19, March, 1975, p. 98-109

[8] Friedman A.D.
Fundamentals of logic design and switching theory. Rockville, MD, USA: Computer Science Press, 1986

[9] Hennessy J.
Partitioning programmable logic array-summary. IEEE International Conference on Computer Aided Design (ICCAD-83) Digest of technical papers. Santa Clara, CA, USA, 12-15 Sept. 1983. New York, USA: IEEE, 1983, p. 180-181

[10] Hideo Fujiwara
Logic testing and design for testability. The MIT Press, Cambridge, Massachusetts, 1985, p. 108-115

[11] Hong S.J., Muroga S.
Binary logic theory for design of minimal decoded-PLA's. Univ. Illinois at Urbana-Champaign, USA, Dec. 1984. Report UIUCDCS-R-84-1192, 54 pp.

[12] Hurst S.
Multiple-valued logic : its status and its future. IEEE Trans. Comput., Vol. C-33, No. 12, Dec., 1984, p. 1160-1179

[13] Ishikawa K., Sasao T., Terada H.
An assignment method for programmable logic arrays with decoders. Trans. Inst. Electron. & Commun. Eng. Jpn. Sect. E (Japan), Vol. E 65, No. 5, May, 1982, p. 300

[14] Kuo Y.S., Hu T.C.
Effective algorithms for optimal PLA folding. 1985 International Symposium on VLSI Technology, Systems and Applications. Taipei, Taiwan, 8-10 May 1985. Hsinchu, Taiwan: ERSO, 1985, p. 199-203

[15] Lewin D.
Design of logic systems. Wokingham, England : Van Nostrand Reinhold Co. Ltd., 1985.

[16] Ligthart M., Aarts E., Beenker F.
Design-for-testability of PLA's using statistical cooling. 23 rd ACM/IEEE Design Automation Conference. Las Vegas, NV, USA, 29 June-2 July 1986. Washington, DC, USA: IEEE Comput. Soc. Press, 1986, p. 339-345

[17] Liu W., Atkins D.
Bounds on the saved area ratio due to PLA folding. ACM/IEEE 20 th Design Automation Conference Proceedings. Miami Beach, FL, USA, 27-29 june 1983. New York, USA: IEEE, 1983, p. 538-544

[18] Luby M., Vazirani V., Sangiovanni-Vincentelli A.
Some theoretical results on the optimal PLA folding problem. IEEE International Conference on Circuits and Computers (ICCC 82). New York, USA, 28 Sept-1 Okt. 1982. New York, USA: IEEE, 1982, p. 165-170

[19] Makarenko D., Tartar J.
An efficient algorithm for the optimal folding of PLA's. Proceedings of IEEE International Conference on Computer Design: VLSI in Computers. Port Chester, NY, USA, 7-10 Oct. 1985. Washington, DC, USA: IEEE Comput. Soc. Press, 1985, p. 57-60

[20] Mouftah H.T., Smith K.C.
CMOS integrated circuits for multivalued logic. Int. J. Electron. (GB), Vol. 58, No. 1, Jan., 1985, p. 43-50

[21] Muzio J.C. et al.
Multivariable symmetries and their detection. Proceedings Institution of Electrical Engineers, Vol. 130, Sept. 1983, PT. E, No. 5, p. 141-147

[22] Novikov Ya.
One approach to multiple fault detection in PLA's. Autom. Control & Comput. Sci. (USA), Vol. 17, No. 3, 1983, p. 30-36

[23] Osann R.
Testing programmable logic: an overview. MIDCON 84: Electronic Show and Convention. Dallas, TX, USA, 11-13 Sept. 1984. Los Angeles, CA, USA: Electron. Conventions, 1984, p. 13/1/1-7

[24] Papachristou C.A.
PLA compaction by partition and fusion. IEEE International Conference on Computer Aided Design. Santa Clara, CA, USA, 18-21 Nov. 1985. Washington, DC, USA: IEEE Comput. Soc. Press, 1985, p. 166-168

[25] Rajski J., Tyszer J.
Combinatorial approach to multiple contact faults coverage in programmable logic array. IEEE Trans. Comput. (USA), Vol. C-34, No.6, June, 1985, p. 549-553

[26] Rudell R., Sangiovanni-Vincentelli A.
Multiple-valued minimization for PLA optimization. Proceedings of the 17 th International Symposium on Multiple-valued logic. Boston, MA, USA, May 1987. Washington, DC, USA: IEEE Comput. Soc. Press, 1987, p. 198-208

[27] Sasao T.
Multiple-valued decomposition of generalized Boolean functions and the complexity of programmable logic arrays. IEEE Trans. Comput. (USA), Vol. C-30, No. 9, Sept., 1981, p. 635-643

[28] Sasao T.
An application of multiple-valued logic to a design of masterslice gate array LSI. Proceedings of the Twelfth International Symposium on Multiple-valued Logic. Paris, France,

25-27 May 1982. New York, USA: IEEE, 1982, p. 45-54

[29] Sasao T.
Input variable assignment and output phase optimization of PLA's. IEEE Trans. Comput., Vol. C-33, No. 10, Oct., 1984

[30] Sasao T.
On the optimal design of multiple-valued PLA's. Proceedings of the 16 th International Symposium on Multiple Valued Logic. Blacksburg, VA, USA, 27-29 May 1986. Washington, DC, USA: IEEE Comput. Soc. Press, 1986, p. 214-223

[31] Sasao T., Terada H.
Multiple-valued logic and the design of programmable logic arrays with decoders. Proceedings 1979 International Symposium on Multiple Valued Logic, p. 27-37

[32] Sievers W.
How testability is designed into programmable logic devices. MIDCON 84: Electronic Show and Convention. Dallas, TX, USA, 11-13 Sept. 1984. Los Angeles, CA, USA: Electron. Conventions, 1984, p. 10/2/1-10

[33] Somenzi F., Gai S.
Fault detection in programmable logic arrays. Proc. IEEE (USA), Vol. 74, No. 5, May, 1986, p. 655-668

[34] Su S.Y.H., Cheung P.T.
Computer Minimization of Multiple-valued Switching Functions. IEEE Trans. Comput., Vol. C-21, No. 9, Sept., 1972, p. 995-1003

[35] Tirumalai P., Butler J.
On the realization of multi-valued logic functions using CCD PLA's. Proceedings of the 14 th International Symposium on Multiple-valued Logic. Winnipeg, Man., Canada, 29-31 May 1984.

Silver Springs, MD, USA: IEEE Comput. Soc. Press, May, 1984, p. 33-42

[36] Tison P.L.
An algebra for logic systems-switching circuits application. IEEE Transactions on Computers, March 1971, Vol. C-.., p. 339-351

[37] van Vlaenderen K.J.
Een programma voor het opsplitsen van schakelfuncties. Stageverslag, EB036, Eindhoven University of Technology.

[38] Wegener I.
The Complexity of Boolean Functions. John Wiley & Sons, Chichester, 1987

[39] Wong D.F. et al.
PLA folding by simulated annealing. IEEE J. Solid State Circuits, Vol. SC-22, No. 2, April, 1987, p. 208-215

[40] Xiang X., Muroga S.
Interactive reduction of folded PLA's. Proceedings IEEE International Conference on Computer Design: LSI in Computers (ICCD'86). Port Chester, NY, USA, Oct. 1986. Washington: IEEE Comput. Soc. Press, 1986, p. 592-595

[41] Yu Q., Wing O.
Interval-graph-based PLA folding. 1985 International Symposium on Circuits and Systems. Kyoto, Japan, 5-7 June 1985. New York, USA: IEEE, Vol. 3, 1985, p. 1463-1466