Digitale Techniek

Een inleiding in het ontwerpen van digitale systemen

Jesse op den Brouw

Deel 1

Versie: 0.99pl13
Datum: 31 juli 2018

DE HAAGSE HOGESCHOOL

Suggesties en/of opmerkingen over dit boek kunnen worden gestuurd naar:
J.E.J.opdenBrouw@hhs.nl.

Voorwoord

"Alweer een boek over digitale techniek?" hoor ik veel lezers nu al zeggen. Ja, alweer een boek. "Waarom dan wéér een boek?" is dan uiteraard de volgende vraag. Omdat ik de bestaande boeken niet toereikend vind. Er zijn drie beweegredenen waarom ik dit boek geschreven heb: de onderwerpen, de Nederlandse taal en de prijs.

Een van de eigenschappen van docenten is dat ze het materiaal van anderen nooit perfect vinden voor hun eigen lessen. Slides worden toch nét weer op een andere volgorde gezet dan het boek suggereert en sommige hoofdstukken uit het voorgeschreven boek worden overgeslagen. Daarnaast missen veel boeken nog wel eens stof die wel behandeld zou moeten worden zoals timing.

Er zijn nauwelijks Nederlandse boeken te vinden op het gebied van digitale techniek. Een eenvoudige zoekopdracht op een bekende boeken-site levert een tiental boeken op waarvan de meest recente in 2008 is verschenen. Er zijn zeker goede Engelse boeken te vinden over digitale techniek. Het nadeel is dat deze boeken over het algemeen vrij duur zijn, wat een drempel vormt voor studenten om het boek aan te schaffen. Ook de taal is hieraan debet.

Dit boek is geschreven voor studenten van de hbo-opleiding Elektrotechniek aan De Haagse Hogeschool. De populatie wordt gevormd door een mengelmoes van niveaus: havo-ers, mbo-ers, overstappers van andere hbo-opleidingen en tu-opleidingen. Dat levert nog wel eens problemen op met het niveau van de lessen: sommige studenten vinden de onderwerpen te makkelijk en anderen juist te moeilijk. Dit boek probeert een balans te vinden tussen de twee uitersten. Er zijn extra paragrafen opgenomen met verbredende en verdiepende informatie, zoals vereenvoudigen met de Quine-McCluskey-methode. Voor sommigen is bepaalde stof toch nog moeilijk te begrijpen. Er is getracht om alle principes en voorbeelden goed en uitvoerig te beschrijven. Sommige studenten hebben echter al aan een half woord genoeg.

Wat betreft de Nederlandse taal is dit boek in twee schrijfstijlen geschreven. Enerzijds wordt er informeel geschreven, andere stukken zijn wat formeler opgemaakt. Dat geeft de lezer een prettige afwisseling. Waar nodig zijn Engelse termen gebruikt omdat de industrie die nou eenmaal gebruikt.

Dit boek is opgemaakt in LaTeX [1]. LaTeX leent zich uitstekend voor het opmaken van lopende tekst, tabellen, figuren, programmacode, vergelijkingen en referenties. De gebruikte LaTeX-distributie is TexLive uit 2018 [2]. Als editor is TexMaker [3] gebruikt. Tekst

is gezet in Charter [4], één van de standaard fonts in LaTeX. De keuze hiervoor is dat het een prettig te lezen lettertype is, een *slanted* letterserie heeft en een bijbehorende wiskundige tekenset heeft. Code is opgemaakt in Nimbus Mono [5] met behulp van de *listings*-package [6]. Karnaughdiagrammen zijn opgemaakt met de *askmaps*-package, door de auteur ontwikkeld en beschikbaar gesteld op CTAN [7]. Voor het tekenen van toestandsdiagrammen is *TikZ/PGF* gebruikt [8].

Bijna alle figuren zijn door de auteur zelf ontwikkeld. Waar nodig is gebruik gemaakt van royalty free figuren.

Natuurlijk zullen docenten ook nu opmerken dat dit boek niet aan al hun verwachtingen voldoet. Dat zal altijd wel zo blijven. Daarom wordt de broncode van dit boek op termijn beschikbaar gemaakt. Eenieder die dit wil, kan de inhoud vrijelijk aanpassen, mits voor niet-commercieel gebruik en de wijzigingen gedeeld worden. Hiermee wordt dit boek een levend document dat nooit af is.

Leeswijzer

Hoofdstuk 1 geeft een algemeen overzicht van de digitale techniek. Het begint met een uiteenzetting waarom digitale techniek zo'n belangrijke plaats in de elektrotechniek heeft gekregen. Daarna volgt een lijst met belangrijke gebeurtenissen in de digitale techniek. We kijken kort naar het digitaliseren van analoge signalen om vervolgens te laten zien dat het ontwerpen van digitale systemen op een ordelijke manier moet gebeuren door middel van een ontwerptraject. We bespreken digitale bouwstenen aan de hand van elektrische eigenschappen, logische werking en tijdgedrag. De logische werking wordt ondersteund door het gebruik van waarheidstabellen, logische functie en schemasymbolen. Om snel enkele regels van de schakelalgebra te begrijpen sluiten we af met een eerste blik op logische schakelingen.

In hoofdstuk 2 worden de decimale, binaire en hexadecimale talstelsels uitgelegd. Bijna alle digitale systemen gebruiken het binaire talstelsel om informatie te verwerken. Het octale talstelsel wordt kort toegelicht. Er wordt een introductie gegeven op fracties (het deel van een getal na de komma). Vervolgens wordt de conversie tussen de talstelsels besproken. We gaan hierbij steeds uit van getallen groter dan of gelijk aan 0. We besteden ook aandacht aan het bepalen van het aantal bits voor een decimaal geheel getal en laten enige voorkomende bitbreedtes zien. We bespreken de BCD-code die soms gebruikt wordt bij numerieke gegevens. Als laatste besteden we aandacht aan een aantal coderingen, onder andere voor karakters.

Hoofdstuk 3 legt het fundament voor de schakelalgebra, de wiskunde achter de digitale schakelingen. Rekenregels worden uitgelegd en de som-van-producten-vorm en product-van-sommen-vorm van logische functies worden toegelicht. We geven ook enkele voorbeelden van bewerkingen met schakelfuncties. Als laatste lichten we nog het begrip don't care toe.

In hoofdstuk 4 worden combinatorische schakelingen behandeld. Er wordt getoond hoe een schakeling geanalyseerd en gesynthetiseerd (opgebouwd) wordt vanuit een schakelfunctie. We laten een aantal manieren zien om functies te vereenvoudigen waaronder schakelalgebra en Karnaughdiagrammen. Vervolgens wordt de realisatie met poorten,

multiplexers en ROM's besproken. We werken in een aantal voorbeelden uit hoe een schakeling wordt gerealiseerd vanuit een geschreven specificatie. Er wordt stilgestaan bij de elektrische eigenschappen van ingangen en uitgangen zoals Schmitt-trigger, tri-state en open drain. Ten slotte lichten we het verschijnsel timing hazards toe.

Hoofdstuk 5 behandelt de vier elementaire binaire rekenoperaties: optellen, aftrekken, vermenigvuldigen en delen. Het eerste deel van dit hoofdstuk toont de bewerkingen voor natuurlijke getallen, gehele getallen groter dan of gelijk aan 0. We laten zien hoe de optelling, aftrekking en vermenigvuldiging kunnen worden gerealiseerd met schakelingen. Het tweede deel introduceert het gebruik van gehele getallen met teken. De bekende wijze met plus- en minteken is niet geschikt voor het realiseren van eenvoudige schakelingen. We bespreken de two's complement representatie die wel leidt tot realisatie van eenvoudige schakelingen. Een aantal belangrijke aspecten passeert de revue zoals bereik van getallen en overflow. Ten slotte bespreken we de opteller voor BCD-gecodeerde getallen.

Hoofdstuk 6 gaat geheel over geheugenelementen. We behandelen diverse varianten van latches om vervolgens tot de conclusie te komen dat deze schakelingen nogal wat problemen opleveren. We laten zien dat met flipflops deze problemen grotendeels kunnen worden vermeden. Een belangrijk aspect is de timing van latches en flipflops. Kennis van dit onderwerp is belangrijk om inzicht te krijgen in het tijdgedrag van geheugenelementen en voor het ontwerpen van robuuste schakelingen. We bespreken de asynchrone set en reset waarmee flipflops tijdens het opstarten van de schakeling in een bekende stand kunnen worden gedwongen. Tot slot behandelen we een drietal voorbeelden van flipflops: een flipflop met enable (stuursignaal om de flipflop nieuwe data te laten overnemen), het register en het schuifregister.

Studiewijzer

In onderstaande tabel wordt een overzicht gegeven van de stof die een student bij een eerste introductie onderwezen zou moeten krijgen. Uiteraard is iedereen vrij om zelf de onderwerpen te kiezen.

Inleiding	
Hoofdstuk 1	1.1–1.6, 1.7 (alleen 1.7.1), 1.8–1.12
Hoofdstuk 2	2.1–2.11, 2.13, 2.15
Hoofdstuk 3	3.1–3.6, 3.9, 3.10
Hoofdstuk 4	4.1–4.3 (alleen 4.3.1 en 4.3.2), 4.4–4.6, 4.8–4.10
Hoofdstuk 5	5.1–5.3, 5.6–5.19, 5.21, 5.23
Hoofdstuk 6	6.1–6.3, 6.5, 6.6, 6.8–6.12, 6.15–6.18

Verantwoording inhoud

De opzet van het boek is redelijk klassiek. In het eerste deel wordt nog geen gebruik gemaakt van VHDL om de student niet af te leiden van het doel: leren wat digitale techniek inhoudt. Elk hoofdstuk wordt begeleid door een aantal vraagstukken. Veel van deze vraagstukken geven extra informatie omtrent een bepaald onderwerp. Het maken van de vraagstukken wordt daarom ook aanbevolen. In dit boek wordt de TTL-technologie

slechts kort beschreven. Reden hiervoor is dat de meeste digitale schakelingen niet meer met deze technologie gemaakt worden. CMOS wordt veruit het meeste gebruikt. Daarom wordt aan deze technologie meer aandacht besteed.

Het eerste hoofdstuk geeft in vogelvlucht de digitale techniek weer. Er wordt nog niet gesproken over binaire getallen. Dit is bewust gedaan om naast de theoretische kennis ook snel aan de slag te gaan met praktische vaardigheden. Er is een paragraaf gewijd aan elektrische schakelingen voor ingangen en uitgangen omdat de studenten tijdens praktische werkzaamheden hier mee te maken krijgen.

Moderne computersystemen rekenen (voor de gebruiker althans) vrijwel allemaal in het binaire talstelsel. Een gedegen kennis hiervan is essentieel. Voor grotere getallen wordt het hexadecimale stelsel gebruikt, zeker als de student later met microcontrollers en computerarchitectuur aan de slag gaat. Getallen kunnen ook codes voorstellen. We laten de BCD-code zien die in veel systemen nog gebruikt wordt als decimale uitvoer noodzakelijk is. Verder passeert de ASCII-code de revue. Veel microcontrollers zijn uitgevoerd met een seriële interface waarover deze code wordt getransporteerd en veel computerbestanden zijn de ASCII-code opgeslagen.

Schakelalgebra is veelal een taaie kost. Toch is het noodzakelijk hier de grondslagen van te beheersen om een digitaal systeem te doorgronden. Combinatorische schakelingen vormen het begin van de “echte” digitale techniek. Hoewel talen als VHDL veel maskeren (denk aan minimalisatie) is het nodig om te spelen met logische bouwstenen.

Over de zin en onzin van (handmatig) minimaliseren kan gediscussieerd worden. De meeste ontwerpsoftware is al met een *minimizer* uitgerust. Minimaliseren wordt in dit boek besproken om de student er bewust van te maken dat het mogelijk is om een schakeling op meerdere manieren te realiseren en dat een bepaalde ontwerpkeuze kan leiden tot meer of minder logica. Het gebruik van talen als VHDL verbergt dat in grote mate. Het minimaliseren geeft de latere beroepsbeoefenaar *awareness* dat zulke talen hardware genereren en geen software.

Vroeg of laat moeten studenten zich ook bezig houden met de elektrische kant van een schakeling. Het is van belang om te weten hoe ingangen moeten worden aangestuurd en hoe uitgangen de rest van de schakeling beïnvloeden. Er wordt alleen gekeken naar CMOS-technologie omdat de meeste grote chips in deze technologie worden ontworpen. De TTL-technologie is grotendeels verdwenen. Soms zijn ingangen en uitgangen *TTL-compatible* zodat ze zouden kunnen worden aangesloten op TTL-(compatible) chips. De student moet zich dit deel dan zelf eigen maken. Referenties naar literatuur worden gegeven.

De two’s complement representatie is dominant in de digitale techniek en software. Er is dan ook een behoorlijk stuk ingeruimd om dit onderwerp te bespreken. De student heeft hier profijt van omdat veel andere vakgebieden, zoals microcontrollers en programmeren, gebruik maken van de two’s complement representatie.

In het hoofdstuk over geheugenschakelingen zijn de bekende schakelingen als SR-latch, gated SR- en D-latch en D-flipflop prominent aanwezig. De JK-flipflop is als verdiepend aangemerkt omdat alle moderne chips uitsluitend met D-flipflops worden uitgerust. De

SR-flipflop wordt niet behandeld. Een enable-faciliteit mag alleen synchroon worden ingebouwd en er wordt toegelicht waarom clock gating beter vermeden kan worden.

Website

Op de website http://ds.opdenbrouw.nl zijn slides, practicumopdrachten en aanvullende informatie te vinden. De laatste versie van dit boek wordt hierop gepubliceerd. Er zijn ook voorbeeldprojecten voor de Quartus-software van Altera te vinden. De projecten kunnen vaak zonder aanpassingen op het DE0-bordje van Terasic uitgeprobeerd worden.

Dankbetuigingen

Dit boek had niet tot stand kunnen komen zonder hulp van een aantal mensen. Ik wil collega Harry Broeders van Hogeschool Rotterdam bedanken voor zijn geweldige bijdrage. Niet alleen op technisch gebied, maar ook taalkundig en op de indeling van dit boek. Collega Ben Kuiper heeft een prima bijdrage geleverd op technisch en taalkundig gebied. Veel type- en stijlfouten zijn door hem ontdekt en gecorrigeerd. Mehmet Can wordt bedankt voor zijn bijdrage op het gebied van MOS-transistoren in hoofdstuk 4. Verder wordt Marcel Jongkind bedankt voor het opsporen van enkele fouten.

De ASCII-tabel op pagina 58 is ontwikkeld door Victor Eijkhout. Deze tabel kan gevonden worden op http://www.ctan.org/tex-archive/info/ascii-chart.

De uitspraak “fenomenologische benadering” op pagina 117 is bedacht door Annel van Houts en Jan van Yperen.

Jesse op den Brouw, 2018.

Inhoudsopgave

1

Introductie

Er wordt wel gezegd dat we in het digitale tijdperk leven. Dit tijdperk is begonnen ergens tussen 1950 en 1970 met de opkomst van digitale computers en digitale systemen in het algemeen. Een belangrijke bedrage aan het digitale tijdperk is de massaproductie van digitale systemen, zoals computers, cd-spelers en smartphones. De productie van een *chip* (een plakje silicium met daarop miljoenen transistoren) wordt steeds goedkoper en het aantal transistoren op een chip neemt nog steeds toe. Een microprocessor is voor een paar euro te koop.

Ook analoge systemen worden gedigitaliseerd. Een voorbeeld hiervan is de audioversterker. Door toevoeging van digitale elektronica kan de luisteraar direct muziek streamen via het internet. De klankkleur (hoge en lage tonen) kan snel worden ingesteld door *presets*. Een kleine computer in de versterker berekent dan automatisch de juiste sterkte.

Digitale systemen zijn “intelligenter” dan analoge systemen. Neem als voorbeeld de thermostaat van de centrale verwarming. Een jaar of 30 geleden was dit een simpele aan/uit-regeling. Door toevoeging van drukknoppen, een display en een microprocessor met software kan de gebruiker het complete dagritme van de verwarming instellen. Het is zelfs mogelijk om de thermostaat draadloos te bedienen met een smartphone of tablet. Daarnaast kunnen er allerlei gegevens worden bijgehouden als temperatuurverloop over de dag, luchtvochtigheid en luchtdruk. Voor de laatste twee is de thermostaat eigenlijk niet bedoeld, maar het is mooi meegenomen.

Het internet heeft ontegenzeggelijk bijgedragen aan de digitalisering van systemen. Denk hierbij alleen maar aan websites met een webshop. Bedrijven hebben dan geen winkel meer nodig en kunnen voorraden tot een minimum beperken. De laatste trend is het *Internet of Things* (IoT), een netwerk van kleine en grote apparaten die informatie met elkaar uitwisselen.

In dit hoofdstuk bespreken we de grondslagen van de digitale techniek. We laten een aantal gedigitaliseerde toepassingen de revue passeren. We bespreken hoe het ontwerptraject van een digitaal systeem in elkaar steekt en waar een ontwerper op moet letten.

Ten slotte geven we een kort overzicht van de elektrische eigenschappen, tijdgedrag en logische werking van enkele basale digitale bouwstenen.

1.1 Analoog tegenover digitaal

Veel toepassingen die ooit analoog waren, zijn gedigitaliseerd. Hieronder is een aantal van deze toepassingen opgesomd.

- *Fotografie.* In zijn boek over digitale techniek uit 2000 schrijft Wakerly dat de meeste foto's nog met film worden geschoten [9]. De vraag is of hij toen kon weten of digitale fotografie zo'n vlucht zou nemen. De ontwikkeling en gebruik van goedkope, grote geheugenkaarten samen met de digitale verwerking van de foto's heeft ervoor gezorgd dat er op dit moment nog nauwelijks "analoge" foto's gemaakt en ontwikkeld worden.
- *Audio-opnamen.* Een belangrijke mijlpaal in de overgang van het analoge naar het digitale tijdperk is opgenomen muziek. Begin jaren '80 heeft het digitale formaat van de optische *compact discs* de analoge formaten zoals grammofoonplaten en cassettebanden verdrongen [10]. En zelfs dat is weer achterhaald. Tegenwoordig is het mogelijk om muziek te *streamen*: het afspelen van muziek via een directe internetverbinding [11]. Een bekende speler op de markt is Spotify. Toch is "analoog" niet helemaal verdrongen, integendeel zelfs. Op het moment van schrijven is er een revival van de ouderwetse vinyl-langspeelplaten [12].
- *Video-opnamen.* Analoge videobanden worden nog nauwelijks gebruikt. Ze zijn vervangen door de DVD (Digital Versatile Disc). De DVD slaat informatie op met een hoge compressie. Een enkellaags, enkelzijdige DVD kan 35 miljard bits opslaan, genoeg voor twee uur video. De Blu-ray Disc (BD) is een geavanceerde versie van de DVD — beide informatiedragers zijn niet compatibel — en kan tot 128 GB aan informatie opslaan op een vierlaags schijfje. Ook de DVD en de BD worden steeds meer verdrongen door streaming media. Een bekende speler op de markt is NetFlix.
- *Telefonie.* De digitale techniek heeft onmiskenbaar een stempel gedrukt op de ontwikkeling van de telefonie. Tot 1984 waren alle telefoonverbindingen analoog. Ook de diverse coderingen, zoals telefoonnummers en signalering van muntgeld met publieke telefoons, ging via analoge codering. Telefooncentrales werden na 1984 in hoog tempo gedigitaliseerd. In 1987 werd op interlokaal niveau een digitaal telefoonnetwerk gerealiseerd, parallel aan het bestaande analoge netwerk. Vanaf 1989 kwamen ISDN-verbindingen voor de gebruiker beschikbaar [13]. Later werden andere diensten geïntroduceerd, zoals VoIP (Voice over IP). Mobiele telefonie was al beschikbaar via analoge netwerken (ATF1/ATF3). Op 1 juli 1994 werd het eerste GSM-netwerk in werking gesteld [14], vooral voor de zakelijke markt. In 1998 kwam de grote doorbraak toen vijf aanbieders gsm-frequenties wisten te bemachtigen [15]. Hoewel IBM en Nokia al eerder smartphones uitbrachten, veranderde de markt drastisch toen Apple in 2007 de iPhone uitbracht [16]. Naast telefoneren en sms-en was het mogelijk om allerlei programmaatjes te installeren, apps genaamd. Ineens werd de telefoon een kleine computer, met tal van mogelijkheden. Tegenwoordig is de smartphone niet meer uit het straatbeeld weg

te denken. Een onderzoek uit 2018 laat zien dat 93% van de Nederlanders een mobieltje heeft [17].

- *Post.* De postmarkt heeft bijzonder te lijden gehad onder de digitaliseringswoede van bedrijven en consumenten. Tussen 2008 en 2014 is het aantal verstuurde poststukken met 40% teruggelopen [18], voornamelijk als gevolg van minder verstuurde brieven. De reden van deze afname is duidelijk: elektronische post. De aantal verstuurde *e-mails* is gigantisch: in 2015 zijn er wereldwijd 205 miljard e-mails per dag verstuurd [19]. Ook het gebruik van het World Wide Web heeft tot het inzakken van de postmarkt geleid. Een catalogus die vroeger via de post werd verstuurd, is nu online te bekijken. Via de webshop is direct een product te bestellen. Dat heeft trouwens wel geleid tot een toename van het aantal verstuurde pakketjes. Elektronische post bestaat al heel lang. De eerste e-mail is verstuurd in 1971 [20]. Toen is ook het bekende '@'-teken in gebruik genomen. In 1982 is de standaard voor email-adressen opgesteld, de beroemde RFC822 [21].
- *Archivering.* Wie kent het nog: de geur van oude boeken of een stoffig archief. Digitalisering heeft in hoge mate bijgedragen tot het opslaan van gegevens in bestanden en databases. Dat leidt ook wel weer tot problemen: welke versie van het bestand is nou de correcte? Het digitaal opslaan van gegevens heeft er toe geleid dat een nieuw vakgebied binnen de informatica is ontstaan: *data mining*, het uit een grote berg gegevens zinnige informatie destilleren. Voor bedrijven als Google een behoorlijke uitdaging: er wordt gezegd dat het bedrijf in 2014 zo'n 15 exabyte ($15 \cdot 10^{18}$ bytes) aan data heeft opgeslagen [22].
- *Televisie en radio.* Ook televisie en radio zijn in hoge mate gedigitaliseerd. Begin 2016 beschikte 87% van de inwoners van Nederland over digitale televisie [23]. De voordelen zijn dan ook legio: veel zenders, hogere beeldkwaliteit, interactieve tv, elektronische programmagids (EPG) om er maar een paar te noemen. Wat betreft digitale radio is het niet anders. Er is al sprake van het uitschakelen van analoge radiozenders door de opkomst van Digital Audio Broacasting [24].

1.2 Waarom digitaal

Er zijn veel redenen om toepassingen te digitaliseren. Hieronder is een opsomming gegeven met een aantal overwegingen.

- *Gemakkelijk te bewerken.* Digitale data in de vorm van getallen is gemakkelijk te bewerken. Denk hierbij aan *encryptie* (versleutelen) dat een steeds grotere rol speelt in de internetwereld. Een versleutelde verbinding is door een buitenstaander weliswaar af te luisteren, maar de correcte data is niet af te leiden, alleen als we een *sleutel* hebben. Een andere toepassing is het comprimeren van informatie. Compressie is een techniek waarbij gegevens met minder bits worden opgeslagen dan het origineel. Databestanden moeten uiteraard zonder verlies gecomprimeerd worden, het origineel moet immers te reproduceren zijn. Een voorbeeld van *lossless* compressie is de Huffman-codering [25]. Audio- en videobestanden worden met verlies gecomprimeerd (*lossy*) door alle niet-noodzakelijke informatie te verwijderen. Voorbeelden hiervan zijn de MPEG-standaarden zoals MP3 en MPEG-2.

Bij het versturen van digitale data is het noodzakelijk dat de data ongeschonden ontvangen wordt. Zend- en ontvangstsystemen passen *foutdetectie* en *foutcorrectie* toe. Een systeem dat alleen fouten kan detecteren moet na detectie van een fout de zender om een hertransmissie vragen. Bij foutcorrectie is de ontvanger in staat om (tot op zekere hoogte) de foutieve data te herstellen. Algemeen kunnen we zeggen dat met digitale data *complexe signaalbewerkingen* mogelijk zijn.

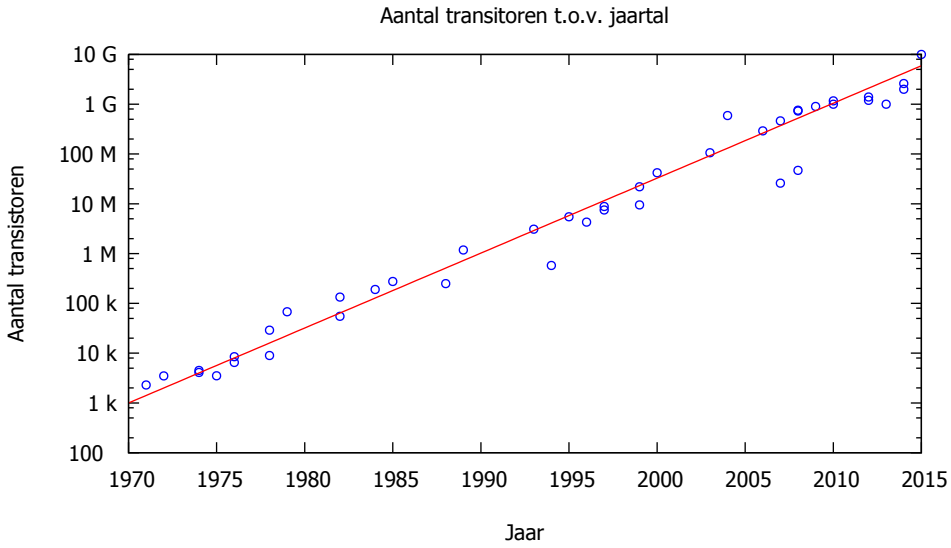
- *Reproduceerbaarheid van informatie*. Een goed ontworpen digitaal systeem zal altijd hetzelfde resultaat produceren bij een gegeven invoer. De uitgangswaarden van analoge systemen variëren met temperatuur, voedingsspanning, veroudering van componenten en andere factoren. In combinatie met een *geheugen* kan de informatie na lange tijd nog goed geproduceerd worden.
- *Betrouwbaarheid m.b.t. storende signalen*. Een goed ontworpen digitaal systeem is in staat om enigszins gestoorde signalen te herstellen. Eventuele ruis en jitter[1] hebben geen invloed op de werking van het systeem. Door toevoeging van foutdetectie en -correctie kan de betrouwbaarheid verhoogd worden. Bij analoge signalen is het stoorsignaal vaak niet te onderscheiden van het originele signaal.
- *Betrouwbare opslag van informatie*. Digitale informatie is eenvoudig op te slaan. Er zijn geen speciale schakelingen nodig om een continu signaalniveau te registreren. Daarnaast kan informatie langere tijd bewaard worden met behoud van kwaliteit. Een en ander hangt natuurlijk wel af van het gebruik en de manier van opslag.
- *Snelheid*. De huidige generatie digitale systemen zijn razendsnel. Ze kunnen schakelen in minder dan 1 nanoseconde. Het heeft ook te maken met parallellisme, iets wat met software niet lukt. Bij software worden de instructies een voor een (d.w.z. sequentieel) uitgevoerd. Hardware daarentegen voert "berekeningen" tegelijk uit. Een gespecialiseerde MPEG-2-decoder kan op een veel lagere snelheid net zo veel bereiken als een videobewerkingsprogramma op een PC.
- *Korte ontwerptijd*. De time-to-market wordt steeds korter en ontwerpers moeten alle zeilen bijzetten om aan de vraag van de consument te voldoen. Met behulp van *tools* zijn digitale schakelingen snel te ontwerpen en testen. Denk hierbij aan beschrijvingstalen als VHDL en Verilog en aan simulatie. Dat betekent dat de ontwerper naast kennis over digitale techniek ook kennis moet hebben van de ontwikkelsoftware. Het is niet noodzakelijk om een chip in de fabriek te produceren. Configureerbare chips zoals FPGA en CPLD verkorten de ontwerpcyclus aanzienlijk.
- *Eenvoudig te leren*. Voor het ontwerpen van digitale schakelingen is geen uitgebreide wiskundige kennis nodig[2]. Kleine schakelingen zijn eenvoudig te analyseren. Bij analoge schakelingen moet de werking van de componenten nauwkeurig worden bestudeerd.
- *Configureerbaarheid*. Een groot aantal digitale systemen wordt ontwikkeld met *reconfigureerbare logica*, zoals FPGA's en CPLD's. In deze chips zijn al diverse

[1] Jitter is het effect van een digitaal signaal dat niet op equidistante (op gelijke afstand gelegen) tijdstippen wordt verzonden.

[2] De wiskunde voor digitale schakelingen is beperkt, maar de te ontwikkelen schakeling kan wel voortkomen uit een wiskundige functie, bv. digitale filtering.

schakelingen (poorten, geheugen) geplaatst. Door het configureren van de verbindingen tussen de schakelingen wordt de functionaliteit van de chip gerealiseerd. Eventuele aanpassingen kunnen snel worden geïmplementeerd door herconfiguratie van de chip. Deze configureerbaarheid stelt de ontwerper ook in staat om de functionaliteit uit te breiden.

- *Hoge integratiedichtheid.* Op een chip kunnen veel transistoren worden aangebracht en dat aantal verdubbelt zich ongeveer elke twee jaar. Dit wordt de wet van Moore genoemd, opgesteld in 1965 [26]. In figuur 1.1 is het aantal transistoren over de jaren heen te zien. De cirkels geven het aantal transistoren van bepaalde microprocessoren aan. De getrokken lijn volgt de wet van Moore. In 1970 konden er ongeveer 1000 transistoren op een chip geplaatst worden, tegenwoordig 10 miljard. Fabrikanten doen trouwens ook hun best om de wet van Moore in stand te houden.



Figuur 1.1: *Grafiek van de wet van Moore met gegevens van enkele microprocessoren.*

- *Goede testvoorzieningen aan te brengen.* Elk schakeling moet getest worden voordat het bij de consument terecht komt. Testen is een serieuze maar lastige aangelegenheid. Als we nagaan dat voor een eenvoudige chip met 100 transistoren al 2^{100} verschillende mogelijkheden moeten worden getest dan begrijpen we dat het testen van een moderne processor een flinke klus is. In veel gevallen neemt het testen de helft van de totale ontwikkeltijd in beslag [27]. Gelukkig maakt de industrie al jaren gebruik van gestandaardiseerde testmethoden voor digitale systemen. Er is software beschikbaar die het een en ander automatiseert. Met onder andere scan-flipflops is een interne teststructuur aan te brengen (Builtin Self Test of BIST) die geactiveerd kan worden door externe signalering. Het testprotocol wordt meestal gecombineerd met JTAG boundary scan [28, 29]. Dat was oorspronkelijk bedoeld voor het testen van verbindingen op een *printed circuit board* (PCB) nadat de componenten geplaatst waren, maar het is prima te gebruiken om componenten te testen (en zelfs te programmeren).

- *Gebruik van standaard blokken.* Fabrikanten van chips stellen kant-en-klare digitale schakelingen beschikbaar, de zogenoemde *IP-cores*. Er zijn onder andere IP-cores te vinden voor Ethernet, USB, processoren, DSP's en RAM. Bovendien zijn ze al grondig getest. Een ontwerper kan zo snel en betrouwbaar een compleet systeem bouwen.
- *Lage kostprijs.* Het produceren van een chip kost lang niet zoveel meer dan pakweg 30 jaar geleden. De bekende drieknops kookwekker is in diverse winkels voor een paar euro te koop. Ook een aantal microcontrollers zoals de AVR ATmega en de ARM Cortex-M0 kosten maar een paar euro.

Hierboven zijn allemaal sterke punten opgesomd om systemen te digitaliseren.

1.3 Korte geschiedenis van de digitale techniek

Een korte geschiedenis van de digitale techniek is wel op zijn plaats. In tabel 1.1 is een aantal belangrijke mijlpalen opgenomen. De lijst is uiteraard niet compleet.

Digitale techniek is onlosmakelijk verbonden met rekenen. Mensen hebben door de eeuwen heen geprobeerd het rekenwerk te mechaniseren. Al rond 2700 voor de jaartelling werd de abacus gebruikt voor het eenvoudige rekenwerk. In 1623 ontwikkelde Schickard een mechanisch apparaat voor optellen en aftrekken. Het bleef bij een papieren exercitie, het apparaat is nooit gebouwd. In 1672 ontwikkelde Leibniz een mechanisch apparaat voor optellen, aftrekken, vermenigvuldigen en delen.

Digitale techniek is ook onlosmakelijk verbonden met de mechanisatie van productieprocessen. De uitvinding van de ponskaart door Jacquard in 1804 voor het besturen van weefgetouwen deed de industrie op zijn grondvesten schudden. Nu waren er immers geen arbeiders meer nodig want machines namen het werk over. Dat leidde tot veel onrust en oproer onder wevers. De machines werkten echter zeer goed zodat er rond 1812 zo'n 11000 van deze machines in gebruik waren. Jacquard gebruikte ponskaarten vergelijkbaar met de muziekkaarten in draaiorgels genomen. De ponskaarten slaan de informatie op als digitale informatie: gat/aan en geen gat/uit. Hiermee worden in een draaiorgel bepaalde instrumenten geactiveerd of niet.

Charles Babbage gebruikte het idee van ponskaarten voor het opslaan van programma's voor zijn Analytical Engine. Deze mechanische machine was programmeerbaar, had een geheugen en een rekeneenheid [30]. De machine is tijdens Babbage's leven nooit gebouwd. Er is een plan om de machine alsnog te bouwen [31]. Herman Hollerith gebruikte ook ponskaarten voor het verwerken van de Amerikaanse volkstelling van 1888. Hij was oprichter van Tabulating Machine Company dat later overging in IBM.

De eerste echte stap richting de digitale techniek is de ontwikkeling van het algebraïsch systeem door George Boole in 1854. De Booleaanse algebra is het wiskundig fundament voor het ontwikkelen van digitale systemen. Claude Shannon liet in 1938 in zijn proefschrift zien hoe de Booleaanse algebra kan worden toegepast voor het realiseren van schakelingen met schakelaars en relais. De ontwikkeling van de schakelalgebra was een belangrijke stap in het ontwerpen van digitale systemen.

Tabel 1.1: *Korte geschiedenis van de digitale techniek*

−2700		Abacus
458	India	Beschrijving van het getal 0
1623	Schickard	Eerste mechanisch apparaat voor optellen en aftrekken (niet gebouwd)
1672	Leibniz	Mechanisch apparaat voor optellen, aftrekken, vermenigvuldigen en delen
1804	Jacquard	Weefgetouw met ponskaartbesturing
1833	Babbage	Difference Engine
1854	Boole	Verhandeling over logica
1888	Hollerith	Ponskaartverwerking volkstelling VS
1906	De Forest	Vacuümbuis
1918	Scherbius	Enigma codeermachine
1936	Turing	Turing-machine
1937	Shannon	Theoretische verhandeling digitaal ontwerpen
1938	Zuse	Z1 (binair, floating-point, Turing-complete)
1939	Atanasoff e.a.	ABC (binair, niet programmeerbaar)
1946	Eckert e.a.	ENIAC
1948	Shockley e.a.	Transistor
1952	IBM	Eerste generatie computers (IBM 701, buizen)
1957	IBM	Tweede generatie computers (IBM 608, transistoren)
1960	Kilby e.a.	Eerste commerciële ic
1964	TI	Eerste TTL-ic (7400)
1964	IBM	Derde generatie computers met ic's
1969	Cerf e.a.	Eerste ontwikkeling van het internet (ArpaNet)
1971	Intel	Eerste microprocessor (4004, 4 bits, 2300 transistoren)
1974	Intel, Motorola	8080/85, 6800 e.a. (8 bits)
1978	MMI	Programmable Array Logic (PAL)
1978	Intel	8086 (16 bits, 29000 transistoren)
1979	Motorola	68000 (32 bits, 68000 transistoren)
1985	Xilinx	Eerste Field Programmable Gate Array (FPGA)
1993	Intel	Pentium (60 MHz)
2010	Sun	1 miljard transistoren op een IC, 16-Core Sparc T3
2014	Intel	15-Core Xeon Ivy Bridge-EX (5 miljard transistoren)
2015	Oracle	Sparc M7 (> 10 miljard transistoren)

De eerste *general-purpose* digitale computer was de ENIAC. Deze computer is gebouwd tussen 1943 en 1946 en was eigenlijk bedoeld voor het berekenen van vluchtbanen van granaten. Het was één van de eerste computers die is opgebouwd met onderdelen die te vinden zijn in elke moderne computer. De machine kon worden geherprogrammeerd en één van de eerste programma's was onderdeel van een haalbaarheidsstudie naar de ontwikkeling van de waterstofbom. De ENIAC is gebouwd met vacuümbuizen in plaats van relais (alhoewel er nog wel relais in zaten). De schakelsnelheid van vacuümbuizen is vele malen groter dan van relais. De betrouwbaarheid liet echter wel te wensen over zodat de machine meerdere keren per dag moest worden gestopt om buizen te vervangen.

Het opgenomen vermogen was rond de 150 kW, waardoor al gauw het gerucht ging dat de lichten in de omgeving dimden als de machine werd gestart.

Met de uitvinding van de transistor in 1947 werd de weg ingeslagen van de verdere verkleining van schakelingen. Betrouwbare halfgeleiderschakelaars vervingen de vacuümbuizen zeer snel. De eerste commercieel verkrijgbare computer met transistoren was de IBM 608 in 1957. Het duurde dus nog 10 jaar voordat de transistor zijn weg vond in de computerindustrie.

Met de uitvinding van de fotolithografie in 1957 werd het mogelijk om meerdere transistoren en verbindingen als één geïntegreerde schakeling te produceren. De betrouwbaarheid en complexiteit van de schakelingen nam enorm toe. Aldus begon de het tijdperk van de geïntegreerde circuits dat resulteerde in de eerste microprocessor: de 4-bits 4004 van Intel werd in 1971 ontwikkeld om gebruikt te worden in een rekenmachine. In de jaren '70 en '80 domineerden Intel en Motorola de wereld van de microprocessors. Er werden veel systemen gebouwd rond de 8080/8085 (Intel) en 6800 (Motorola). Intel won uiteindelijk de wedloop toen IBM de 8086 koos als processor in de IBM PC/XT.

De eerste configureerbare digitale component was de Programmable Array Logic (PAL) van Monolithic Memories. Het bevat een hoeveelheid configureerbare schakelaars in een regelmatige structuur waardoor elke mogelijke schakelfunctie kon worden gerealiseerd. De schakelfunctie kon door de ontwerper zelf in de PAL worden geconfigureerd. Dit leverde een enorme flexibiliteit op voor ontwerpers.

In 1985 introduceerde Xilinx de Field Programmable Gate Array (FPGA), een ic dat telkens weer opnieuw geconfigureerd[3] kan worden. Dit maakt het mogelijk om ontwerpen te veranderen en in-het-veld aan te passen nadat een product is gekocht. Daarnaast kunnen bestaande ontwerpen snel worden aangepast om een nieuw product te realiseren. Zie bijlage A voor een introductie over FPGA's.

1.4 Digitalisering

Onder *digitalisering* wordt het digitaal maken van systemen verstaan. Er zijn twee richtingen binnen de techniek waaruit dit digitaliseren zijn oorsprong heeft: digitaliseren van productieprocessen en van informatie.

De industriële automatisering in de jaren '70 van de vorige eeuw heeft het leven van de mensheid drastisch veranderd. Handmatig werk werd vervangen door machinaal werk. Voordelen waren er legio: goedkoper dan handwerk, geen ziektegevallen en rond-de-klok werken. Er was natuurlijk veel verzet tegen en men dacht dat de werkloosheid enorm zou toenemen. Gelukkig blijkt dat (nu nog) niet op grote schaal zo te zijn. Er komen immers nieuwe beroepen bij zoals operators, onderhoudsmonteurs en ontwerpers. Er zijn echter wel indicaties dat in de toekomst veel werk door robots gedaan gaat worden [32].

Bij de tweede richting denken we gelijk aan administratieve automatisering. De kaartenbakken zijn vervangen door databases en tal van brieven worden automatisch opgemaakt.

[3] Het is wel raar dat een FPGA configureerbaar heet, terwijl de naam suggereert dat ze programmeerbaar zijn.

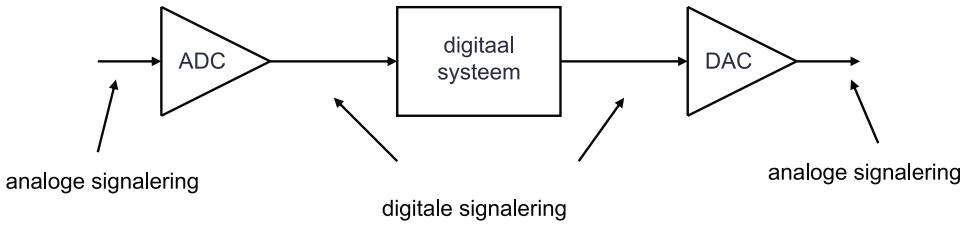
De beroepen worden ook anders. Van de moderne medewerker wordt verwacht dat hij goed kan omgaan met tekstverwerkers en spreadsheetprogrammatuur. Eigenlijk zijn computers bedoeld om te rekenen. Berekeningen van lastige wiskundige formules worden niet meer met de hand gedaan. Computers kunnen dat veel sneller. Maar het zijn nog altijd de mensen die de computer instructies geven hoe iets moet worden uitgerekend.

Een ander voorbeeld is het opslaan en verwerken van audio. We lichten dit in de volgende paragraaf nader toe. Ook hier zijn nieuwe beroepen ontstaan: programmeurs, informatiekundigen, ontwerpers etc.

1.4.1 Digitaliseren van analoge signalen

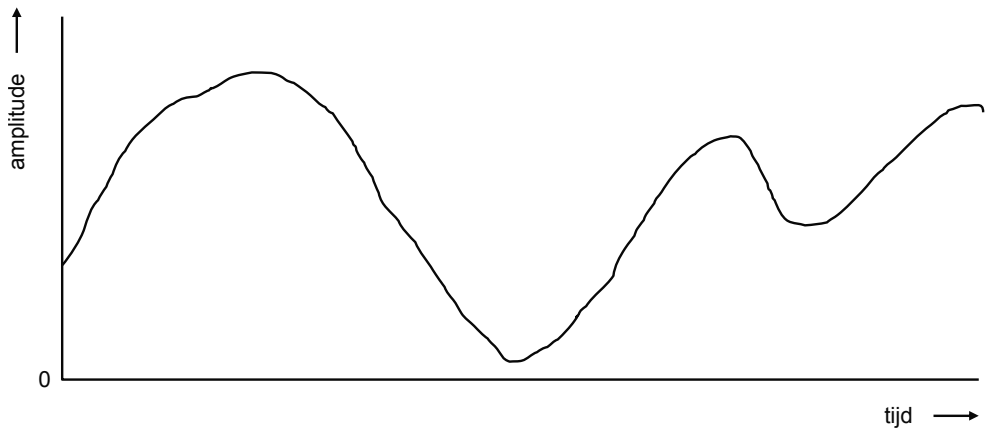
Bij digitalisering van signalen worden bekende fysische grootheden (stroom, spanning, druk, licht etc.) omgezet naar getallen. Het is eigenlijk een transformatie van de fysische grootheden naar getallen die in principe niet binair (tweewaardig) hoeven te zijn. De digitale informatie kan met behulp van elektronica (computers) bewerkt worden en weer naar de fysische grootheden terug getransformeerd worden. Transformatie gebeurt met behulp van een *Analog-Digital Converter* (ADC) en een *Digital-Analog Converter* (DAC).

In figuur 1.2 is het principeschema weergegeven. Het principeschema kan het best worden uitgelegd aan de hand voor een voorbeeld.



Figuur 1.2: *Principeschema digitalisering.*

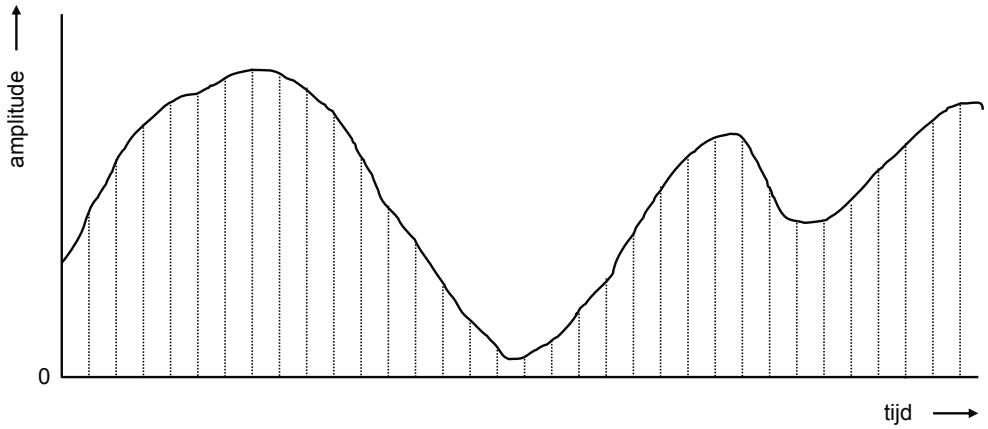
Op de ingang van de ADC wordt een analoog signaal aangeboden, bijvoorbeeld een audiosignaal in de vorm van een spanning. Dit signaal is te zien in figuur 1.3. Het betreft



Figuur 1.3: *Analoog signaal.*

hier een signaal dat louter positieve waarden aanneemt maar dat is niet noodzakelijk. Om een indruk te geven: de *line output level* van een cd-speler ligt tussen −1 V en +1 V (2 V *peak-to-peak*). Het signaal is continu in de tijd en kan alle mogelijke waarden aan tussen een bepaald minimum en maximum aannemen.

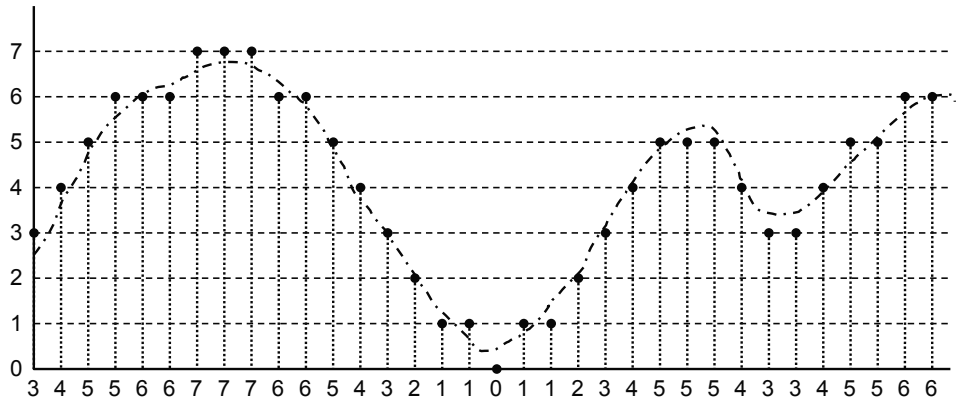
De ADC zorgt ervoor dat de analoge spanning op *equidistante* (op gelijke afstand gelegen) tijdstippen wordt *bemonsterd*, zie figuur 1.4. In veel literatuur wordt de Engelse term *sampling* gebruikt. Het signaal is dan nog steeds analoog maar niet meer continu. We noemen dit een *discreet analoog signaal*. Een belangrijke parameter van het digitale systeem is de bemonsteringsfrequentie die aangeeft hoeveel van deze monsters per seconde worden genomen. Bij de compact disc is de bemonsteringsfrequentie 44,1 kHz, dat wil zeggen 44.100 monsters per seconde. Hogere bemonsteringsfrequentie zijn in professionele toepassingen gebruikelijk, zoals 48 kHz, 96 kHz en 192 kHz. Om geen informatie verloren te laten gaan of te vervormen moet de bemonsteringsfrequentie minstens twee keer zo hoog zijn als de hoogste frequentie die voorkomt in het analoge signaal. Dit wordt de Nyquist-frequentie genoemd [33].



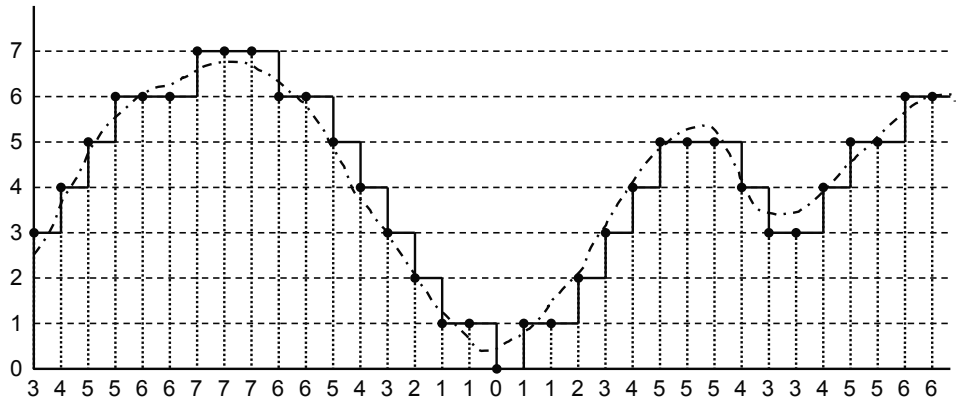
Figuur 1.4: *Analoog signaal bemosterd op equidistante tijdstippen.*

Na bemonstering kwantiseert de ADC het analoge bemonsterde signaal. Kwantiseren is het vertalen van een monster naar een bijbehorende waarde waarbij het aantal mogelijke waarden beperkt is. De waarden worden binair opgeslagen omdat digitale schakelingen verreweg met twee waarden werken. In figuur 1.5 is te zien dat het signaal als voorbeeld in acht niveaus wordt opgedeeld. De waarden kunnen dus met drie bits worden opgeslagen. In de figuur is te zien dat de meeste monsters niet precies op een niveau liggen, maar een beetje erboven of eronder. De ADC rondt het monster af naar het nabij gelegen niveau. Bij het kwantiseren treedt dus een afrondingsfout op. De vraag is of dat problemen oplevert bij verdere verwerking van de digitale data. De acht gebieden in het voorbeeld kwantiseren het signaal nogal grof. Bij een cd worden de monsters met 16 bits opgeslagen, er zijn dan $2^{16} = 65.536$ kwantiseringsniveaus. Uitgaande van de lijnspanning van 2 V is de *resolutie* ongeveer 30 µV.

De data kan nu verwerkt worden in het digitale systeem. Voorbeelden hiervan zijn filtering, versterking of verzwakking of opslag in een geheugen (denk weer aan de cd). Na bewerking wordt de data doorgegeven aan een DAC, die de digitale informatie omzet naar analoge spanningen. Dit is te zien in figuur 1.6. Ook hier is te zien dat de data op



Figuur 1.5: *Kwantisering van het bemonsterde signaal in acht gebieden.*



Figuur 1.6: *Het geanalogiseerde signaal.*

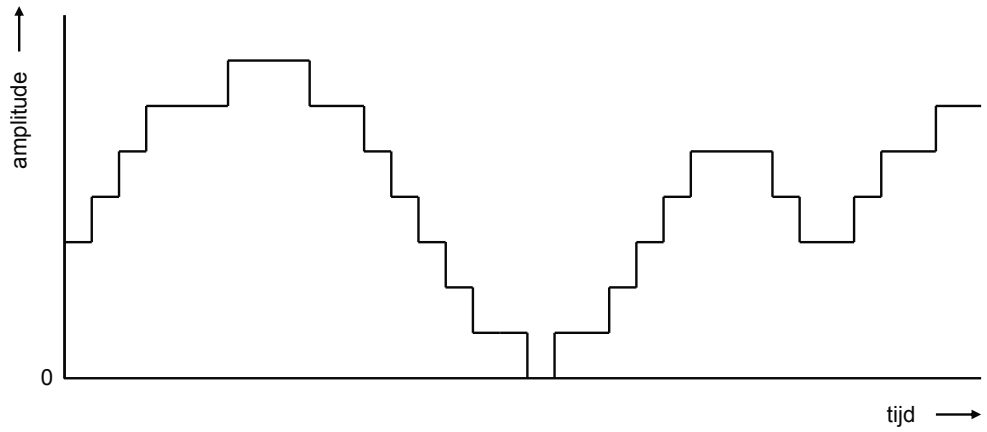
equidistante tijdstippen wordt omgezet.

Bij het kwantiseren treden afrondingsfouten op. Dat houdt in dat het originele signaal niet meer exact te reconstrueren is. In figuur 1.7 is te zien dat het gereconstrueerde signaal veel wegheeft van het originele signaal in figuur 1.3 maar dat het niet volledig overeenkomt met het originele signaal is. De steile flanken in het signaal worden veroorzaakt door het bemonsteren. Aan de uitgang van de DAC wordt daarom een analoog filter geplaatst om de steile flanken weg te filteren zodat een betere reconstructie van het oorspronkelijke signaal wordt verkregen.

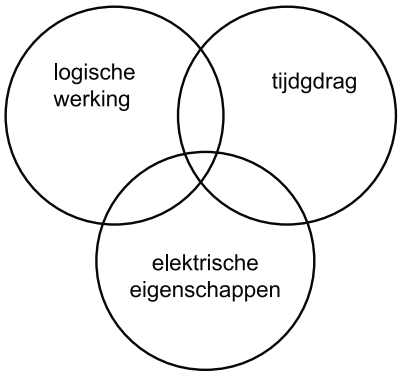
1.5 Het ontwikkelen van digitale schakelingen

Bij het ontwerpen van digitale schakelingen moeten we, voor wat betreft de techniek, rekening houden met de logische werking, tijdgedrag en elektrische eigenschappen. De drie gebieden overlappen elkaar enigszins, zoals te zien is in figuur 1.8.

Bij de meeste schakelingen zijn de drie gebieden onafhankelijk van elkaar te onderzoeken. Dat wil niet zeggen dat de gebieden niet met elkaar verbonden zijn. Maar een ontwerper kan zich eerst op het logisch ontwerp richten en vervolgens het tijdgedrag analyseren. Als blijkt dat de schakeling te langzaam is, dan moet opnieuw naar het logisch niveau



Figuur 1.7: *Reconstructie van het analoge signaal.*



Figuur 1.8: *De drie ontwerpgebieden van digitale schakelingen.*

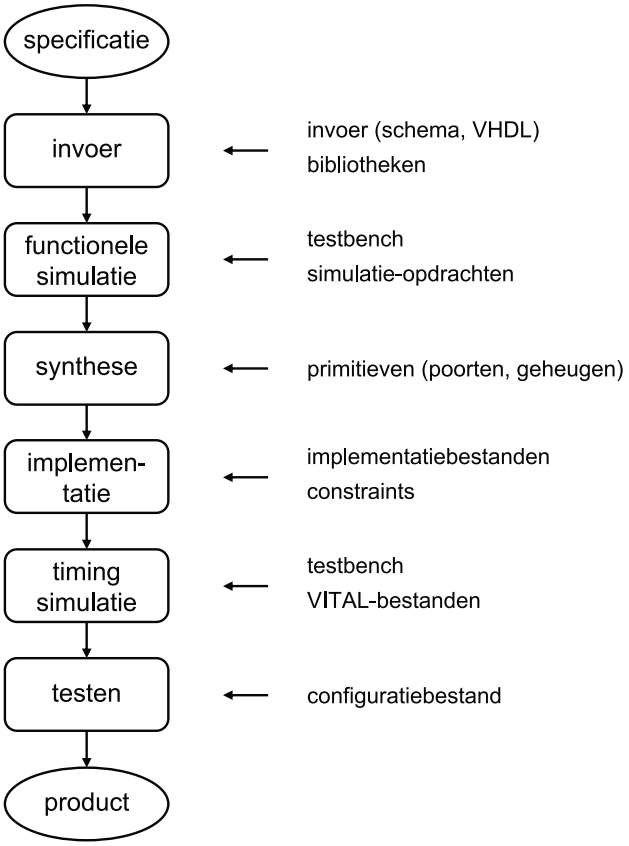
gekeken worden.

Met logische werking bedoelen we de functionaliteit van het systeem. In deze ontwerpfase bepalen we dus wat de schakeling moet doen. We gebruiken technieken zoals vereenvoudigen om de schakeling te ontwerpen.

Er zijn schakelingen waarbij het tijdsaspect minder relevant is. Denk hierbij aan een 7-segment display van een digitale klok. Die informatie hoeft niet met de hoogst haalbare snelheid afgebeeld te worden, dat mag best even duren. Het ligt dan voor de hand om een schakeling te ontwerpen die energiezuinig is.

Aan de andere kant zijn er ook schakelingen waarbij de drie gebieden nauw verbonden zijn, bijvoorbeeld de *asynchrone machines*. Slechts een subtiele verandering in de logische werking heeft direct gevolg voor het tijdgedrag.

Een *ontwerptraject* is een reeks van handelingen die moeten worden uitgevoerd om van de specificatie van het product, meestal in geschreven vorm, tot een eindresultaat te komen. We gaan hierbij ervan uit dat een schakeling in een FPGA of ASIC wordt geïmplementeerd en niet met losse componenten. In figuur 1.9 is het traject te zien. Bij alle stappen biedt programmatuur hulp aan de ontwerper. Een aantal stappen wordt automatisch uitgevoerd zoals synthese en implementatie.



Figuur 1.9: *Het ontwerptraject van een digitale schakeling.*

Het bedenken van de schakeling is een creatief proces. Ervaring en goede kennis van digitale systemen helpt de ontwerper hierbij op weg. Als het probleem niet in één keer te overzien is, worden er deelontwerpen gemaakt die onderling verbonden zijn. We noemen dit *partitioneren*. Elk deelontwerp bevat een schakeling of, als het probleem nog niet te overzien is, weer deelontwerpen. We noemen dit principe *hiërarchisch ontwerpen*. Nadat alle deelontwerpen zijn bedacht wordt overgegaan tot het invoeren van de deelschakelingen. Hier hebben we een verscheidenheid aan keuzes. We kunnen schakelingen invoeren als een schema met behulp van een tekenpakket, maar ook met behulp van een speciale taal die beschrijft wat de schakeling moet doen, bijvoorbeeld VHDL.

Nadat een ontwerp (of deelontwerp) is ingevoerd volgt de functionele simulatie. Tijdens deze simulatie wordt alleen gekeken of de schakeling op logisch niveau werkt. Tijdgedrag en elektrische eigenschappen worden nog buiten beschouwing gelaten. Dit is een eerste test om te zien of de schakeling aan de specificaties voldoet. Eventueel volgt een herontwerp. Het opzetten van een goede simulatie hoort bij de vaardigheden van een ontwerper.

Werkt het ontwerp op functioneel niveau, dan wordt het geheel *gesynthetiseerd*. Synthese wil zeggen dat het ontwerp wordt omgezet naar primitieven. Voorbeelden hiervan zijn logische poorten en geheugens. Deze stap kan geheel automatisch door programmatuur worden uitgevoerd. Zo hoeft de ontwerper zich niet bezig te houden met allerlei details van de gebruikte technologie.

Implementatie wil zeggen dat de primitieven uit de vorige stap worden omgezet naar structuren op een chip. Dat kunnen transistoren zijn bij een *ASIC* (Application Specific Integrated Circuit) of configureerbare blokken op een *FPGA* (Field Programmable Gate Array), maar er zijn ook tussenvormen mogelijk.

Na deze stap is het mogelijk om het ontwerp te simuleren op tijdgedrag. Ook hier kan het zijn dat het ontwerp niet voldoet aan de (tijd-)specificaties. Dan volgt weer een herontwerp. Dat kan zijn bij de invoer van het ontwerp, synthese of tijdens implementatie. Er kan bijvoorbeeld gekozen worden om meer parallellisme door te voeren dat in regel leidt tot een snellere chip ten koste van meer hardware. Er kan ook gekozen worden voor een snellere technologie.

De laatste stap moet niet vergeten worden. Testen van een ontwerp is heel belangrijk. Als blijkt dat de chip in de praktijk niet werkt, dan levert dat voor de fabrikant (en de ontwerper) veel problemen op. Bedenk dat vaak 50% van de ontwikkeltijd in testen gaat zitten.

1.6 Elektrische eigenschappen van digitale schakelingen

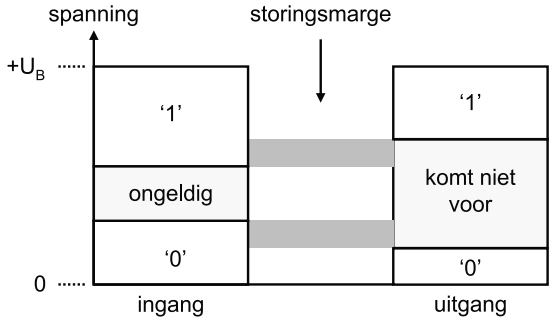
Tot de jaren ’80 waren de meeste elektronische systemen analoog. Denk hierbij aan telefonie, televisie en radio. Analoog wil zeggen dat de signalen (bijvoorbeeld spanningen) in deze systemen elke mogelijke waarde tussen een bepaald minimum en maximum kunnen aannemen. Ook digitale systemen gebruiken analoge spanningen om informatie weer te geven en te bewerken. Digitale signalen zijn dus gewoon analoge spanningen, alleen doen we net alsof ze dat niet zijn. Een digitaal signaal wordt voorgesteld als een signaal dat slechts één van twee waardes aanneemt, die we 0 en 1 noemen (of niet-waar en waar, onjuist en juist, uit en aan of . . .). Deze “digitale abstractie” stelt ons in staat om het analoge gedrag in de meeste gevallen te negeren en net te doen alsof de schakelingen echt met nullen en enen werkt. Dat geldt zeker voor alle schakelingen die we op één chip plaatsen. We hebben dan vooral te maken met de logische werking en het tijdgedrag.

Fabrikanten spreken trouwens liever niet over ‘0’ en ‘1’, maar over ‘L’ (laag) en ‘H’ (hoog). Het wordt dan aan de ontwerper overgelaten hoe de vertaling naar ‘0’ en ‘1’ gebeurt. In de regel vertalen we een ‘L’ naar ‘0’ en een ‘H’ naar ‘1’. Dit wordt *positieve logica* genoemd. Het is echter ook mogelijk om de niveaus om te draaien: een ‘L’ noemen we ‘1’ en een ‘H’ noemen we ‘0’. Dit wordt *negatieve logica* genoemd. In dit boek gebruiken we positieve logica en de notaties ‘0’ en ‘1’. We gebruiken nauwelijks de notaties ‘L’ en ‘H’. De termen ‘laag’ en ‘hoog’ komen wel voor.

Nu is het heel lastig, zelfs onmogelijk, om exact één spanningswaarde te koppelen aan een logische waarde. We kunnen een logische 1 bijvoorbeeld voorstellen met 5 V, maar hoe reageert een schakeling als een spanning van 4,9 V of 5,1 V wordt aangeboden? In de praktijk worden daarom *spanningsbereiken* gebruikt. Een logische waarde wordt dan aangegeven met een minimale en maximale spanning.

In figuur 1.10 is een schematische weergave gegeven van de interpretatie van ingangs- en uitgangsspanningen. De schakeling interpreteert een ingangsspanning als een logische 0

als de aangeboden spanning ligt tussen de referentie[4] en een zekere maximale spanning. In de figuur is dat links te zien in gebied '0'. Voor een logische 1 geldt dat er minimaal een bepaalde spanning moet worden aangeboden, te zien in het gebied '1'. De maximaal aan te bieden spanning is de voedingsspanning (in de praktijk mag dat meestal iets erboven liggen).



Figuur 1.10: *Interpretatie van logische 0 en 1.*

Te zien is dat de gebieden voor een logische 0 en 1 niet aaneengesloten zijn. Er is een gebiedje waarin de spanning als een ongeldig logische waarde wordt gezien. Dit zorgt ervoor dat de interpretatie van de waarden voor 0 en 1 duidelijk gerealiseerd kunnen worden (denk eraan dat elke digitale schakeling in wezen analoog is). Aan de kant van de uitgang worden 0 en 1 ook gerealiseerd met spanningsgebieden. Een logische 0 ligt tussen de referentie en een zekere maximale spanning. Een logische 1 ligt tussen een zeker minimum en de voedingsspanning. Ook tussen deze twee gebieden is enige ruimte, de uitgangsspanning zal zich nooit in dit gebied bevinden.

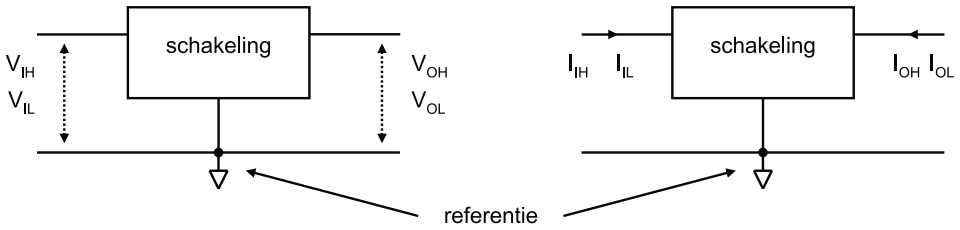
In *statische toestand* zullen alle spanningen zich in een van de twee gedefinieerde gebieden bevinden. In de praktijk kan een spanning nooit direct van het ene gebied naar het andere gebied gaan. Dit kost enige tijd. Er worden aan de *dynamische toestand* eisen gesteld aan de stijg- en daaltijden van de signalen (Engels: rise time en fall time). Een ingangssignaal moet aan deze tijden voldoen anders kan de schakeling kuren vertonen. We mogen er trouwens niet vanuit gaan dat de schakeling in dynamische toestand de bedoelde werking heeft. Die is alleen voor de statische toestand gedefinieerd.

Tussen de ingangs- en uitgangsspanningen zijn *storingssmarges* opgenomen. Dit wordt gedaan zodat fluctuaties van de voedingsspanning, spanningsval op de signaallijnen en storing op de ingangen geen problemen opleveren.

Fabrikanten geven in hun *datasheets* de spanningsniveaus en stromen aan. Daarbij geven ze ook aan hoe de spanningen en stromen moet worden geïnterpreteerd. In figuur 1.11 zijn de definities te zien. Merk op dat de stromen naar binnen toe zijn gedefinieerd.

In tabel 1.2 zijn de spanningsniveaus te zien van de bekende (maar wat ouderwetse) standaard TTL- en CMOS-technologieën. Opgemerkt moet worden dat moderne chips op veel lagere voedingsspanningen werken, bijvoorbeeld 3,3 V, 2,5 V en 1,8 V.

[4] Dit is de spanning die als referentie dient voor alle andere spanningen in de schakeling. Soms wordt dit aangeduid als 0 V of aarde of ground.



Figuur 1.11: *Definities van spanningen en stromen van de schakeling.*

Tabel 1.2: *Spanningsniveaus in volt bij standaard TTL en CMOS.*

	parameter[1]	TTL[2]	CMOS[3]
Minimale ingangsspanning, laag	$V_{IL,min}$	0,0	0,0
Maximale ingangsspanning, laag	$V_{IL,max}$	0,8	$0{,}3 \cdot V_{DD}$
Minimale ingangsspanning, hoog	$V_{IH,min}$	2,0	$0{,}7 \cdot V_{DD}$
Maximale ingangsspanning, hoog	$V_{IH,max}$	V_{CC}	V_{DD}
Minimale uitgangsspanning, laag	$V_{OL,min}$	0,0	0,00
Maximale uitgangsspanning, laag	$V_{OL,max}$	0,4	0,05
Minimale uitgangsspanning, hoog	$V_{OH,min}$	2,4	$V_{DD} - 0{,}05$
Maximale uitgangsspanning, hoog	$V_{OH,max}$	V_{CC}	V_{DD}

1 Deze parameters worden door de fabrikant verstrekt.
2 De nominale voedingsspanning V_{CC} is +5 V.
3 De voedingsspanning V_{DD} ligt tussen +3 V en +15 V.

Ook de stromen die een schakeling in of uit mogen vloeien worden in datasheets aangegeven. In tabel 1.3 zijn de maximale ingangs- en uitgangsstromen te zien. Stromen worden altijd naar binnen toe aangegeven. Een negatief getal geeft aan dat de stroom de schakeling uit stroomt. Een snelle kijk op de kolom TTL laat zien dat één uitgang maximaal tien ingangen kan aansturen. Zie [34] voor een uitgebreide beschrijving van TTL. Een CMOS-uitgang kan veel ingangen aansturen omdat de ingangsstroom zeer laag is. Probleem is wel dat de *capacitieve belasting* toeneemt met het aantal ingangen zodat de vertragingstijden flink toenemen. De twee technologiëen kunnen niet zonder meer door elkaar gebruikt worden. Er zijn speciale *level shifters* nodig waarmee tussen de spanningsniveaus van TTL en CMOS geschakeld kan worden.

Tabel 1.3: *Stromen in milliampères bij standaard TTL en CMOS.*

	parameter[1]	TTL	CMOS[2]
Maximale ingangsstroom, laag	$I_{IL,max}$	$-1{,}6$	10^{-5}
Maximale ingangsstroom, hoog	$I_{IH,max}$	0,04	10^{-5}
Maximale uitgangsstroom, laag	$I_{OL,max}$	16	1,0
Maximale uitgangsstroom, hoog	$I_{OH,max}$	$-0{,}4$	$-1{,}0$

1 Deze parameters worden door de fabrikant verstrekt.
2 Stromen verschillen per fabrikant en per serie.

1.7 Logische schakelingen

Het gebruik van *tweewaardige*, *binaire* of *logische* schakelingen in de digitale techniek is het gevolg van de eenvoud van de schakelingen. Signalen kunnen slechts twee waarden aannemen, bijvoorbeeld *aan* en *uit*, *actief* en *niet-actief*, maar voor ons doel is het handiger om 0 en 1 te gebruiken.

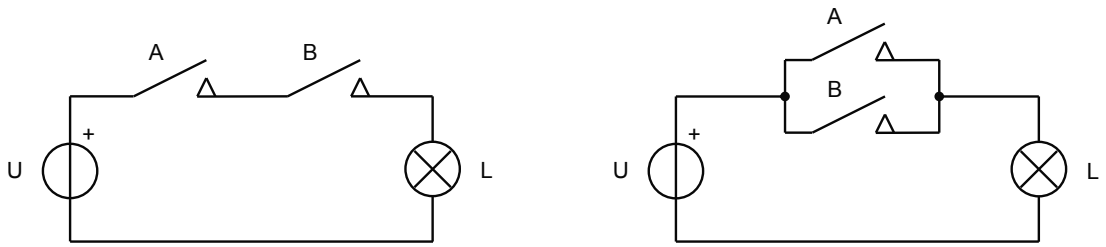
Digitale schakelingen worden verreweg het meest ontworpen met elektronische componenten. De reden hiervoor is dat ze klein, snel en goedkoop zijn. In andere vakgebieden worden pneumatische componenten (lucht), hydraulische componenten (vloeistof) en fotonische componenten (licht) gebruikt.

In de begintijd van de digitale techniek werden de schakelingen opgebouwd uit discrete componenten zoals schakelaars, weerstanden, diodes en transistoren. De schakelingen waren over het algemeen best groot en de complexiteit was laag. De ontwerper was naast het logisch ontwerp een groot deel van zijn tijd bezig om alle elektronica goed te *dimensioneren* (het berekenen van de componentwaarden).

1.7.1 Schakelingen met passieve schakelaars

De eenvoudigste component is een schakelaar die twee standen heeft. In rust is de schakelaar open. In de actieve stand is de schakelaar gesloten. We noemen zo'n schakelaar *normally open*. We kunnen de schakelaars vertegenwoordigen door variabelen die slechts twee waarden kunnen aannemen, 0 en 1. We noemen deze variabelen *binaire* of *logische* variabelen. Een schakelaar die open is geven we aan met een 0, een gesloten schakelaar geven we aan met een 1. Op eenzelfde wijze kunnen we een lamp in de schakeling opnemen. Een lamp die gedoofd is geven we aan met een 0, een lamp die brandt geven we aan met een 1.

Figuur 1.12(a) beeldt een serieschakeling van twee schakelaars uit. De schakelaars zijn in rust getekend. In deze situatie brandt de lamp niet. We geven dit aan door $A = 0$, $B = 0$ en $L = 0$. Nu kunnen we schakelaar B sluiten, dit geven we aan met $B = 1$. Schakelaar A is nog steeds geopend. Ook nu brandt de lamp niet. Deze situatie geven we aan met $A = 0$, $B = 1$ en $L = 0$. In de derde situatie is schakelaar A gesloten en B geopend. De lamp brandt nog steeds niet. We geven dit aan met $A = 1$, $B = 0$ en $L = 0$. Het is duidelijk dat de lamp pas brandt als beide schakelaars gesloten zijn. Dit geven we aan met $A = 1$, $B = 1$ en $L = 1$.



(a) *De logische AND-schakeling.* **(b)** *De logiche OR-schakeling.*

Figuur 1.12: *Twee logische schakelingen.*

We kunnen de vier situaties in een *waarheidstabel* weergeven. Zie hiervoor tabel 1.4. In deze tabel worden de toestanden van de schakelaars en de lamp met een 0 of een 1 aangegeven. Elke rij in de tabel geeft één combinatie van toestanden van de schakelaars aan. Aan elke rij kunnen we een betekenis koppelen.

Tabel 1.4: *Waarheidstabel serieschakeling (AND).*

A	B	L	betekenis
0	0	0	schakelaars open, lamp uit
0	1	0	schak. A open, B gesloten, lamp uit
1	0	0	schak. A gesloten, B open, lamp uit
1	1	1	schakelaars gesloten, lamp aan

Het gedrag van de schakeling kunnen we beschrijven met een *logische functie*:

$$L = A \cdot B \tag{1.1}$$

waarbij geldt dat $L = 1$ als $A = 1$ én $B = 1$, anders is $L = 0$. We noemen de "$\cdot$" de AND-operator en formule (1.1) beschrijft het gedrag van de schakeling in figuur 1.12(a). De "$\cdot$" moet niet verward worden met rekenkundige vermenigvuldiging. We kunnen de punt ook weglaten en dan schrijven we:

$$L = AB \tag{1.2}$$

In figuur 1.12(b) is een parallelschakeling van twee schakelaars te zien. In rust (schakelaars A en B zijn beide 0) brandt de lamp niet ($L = 0$). Als een van de twee schakelaars gesloten is óf als beide schakelaars gesloten zijn, zal de lamp branden. Ook hiervan is een waarheidstabel te maken, zie tabel 1.5.

Tabel 1.5: *Waarheidstabel parallelschakeling (OR).*

A	B	L	betekenis
0	0	0	schakelaars open, lamp uit
0	1	1	schak. A open, B gesloten, lamp aan
1	0	1	schak. A gesloten, B open, lamp aan
1	1	1	schakelaars gesloten, lamp aan

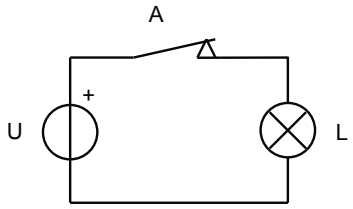
Het gedrag van de parallelschakeling is te beschrijven met de logische functie:

$$L = A + B \tag{1.3}$$

waarbij geldt dat $L = 1$ als $A = 1$ óf $B = 1$ óf $A = B = 1$, anders is $L = 0$. De "+" wordt de OR-operator genoemd en formule (1.3) beschrijft het gedrag van de schakeling in figuur 1.12(b) De "+" moet niet verward worden met de rekenkundige optelling.

De hier gepresenteerde schakelaars zijn in ruststand geopend. Er zijn ook schakelaars die in ruststand gesloten zijn. Dit type wordt *normally closed* genoemd. Daarom spreken we liever niet van open en gesloten maar van een ruststand en actieve stand.

In figuur 1.13 is links een schakeling getekend met een normally closed schakelaar. In de ruststand is de schakelaar gesloten en brandt de lamp. Dit geven we aan met $A = 0$ en $L = 1$. Als de schakelaar geactiveerd wordt, zal de lamp doven. Dit geven we aan met $A = 1$ en $L = 0$. Ook van deze schakeling is een waarheidstabel op te stellen.



A	L	betekenis
0	1	schak. gesloten, lamp aan
1	0	schak. open, lamp uit

Figuur 1.13: *Schakeling en waarheidstabel voor inverseschakeling (NOT).*

Het gedrag van deze schakeling is te beschrijven met de logische functie:

$$L = \overline{A} \tag{1.4}$$

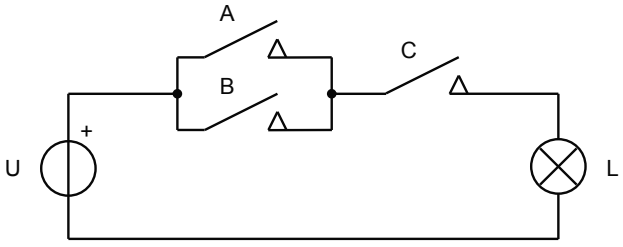
waarbij geldt dat $L = 1$ als $A = 0$, anders is $L = 0$. De waarde van deze functie is de *inverse* van de ingangswaarde. Een veelgebruikte (Engelse) term is *complement*. Een andere veelgebruikte term is de NOT-operatie. Er zijn diverse notatiewijzen voor de NOT-functie. In de bovenstaande functie is een streepje boven A geplaatst (de *overbar*). Deze notatie is waarschijnlijk het beste vanuit het visuele oogpunt. Maar bij het gebruik van computerprogrammatuur is de invoer van het streepje onpraktisch. Daar wordt gebruik gemaakt van de apostrof (Engels: quote) of het uitroepteken of het tilde-karakter. Dus:

$$\overline{A} = A' = !A = \sim A \tag{1.5}$$

Met de functies AND, OR en NOT is het mogelijk om elke logische functie te implementeren. Figuur 1.14 toont hoe drie schakelaars kunnen worden gebruikt om het licht te controleren op een meer complexe manier. Deze serie-parallelschakeling van schakelaars realiseert de logische functie (haakjes geven bewerkingsvolgorde aan):

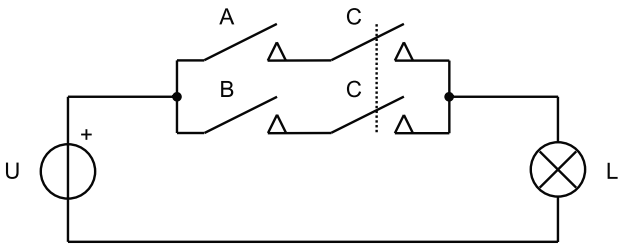
$$L = (A + B) \cdot C \tag{1.6}$$

Het licht brandt als $C = 1$ en tegelijkertijd ten minste één van de schakelaars A of B gelijk is aan 1.



Figuur 1.14: *De logische AND-OR-schakeling.*

In figuur 1.15 is nog een schakeling gegeven. In dit schema komt de schakelaar C twee keer voor. Deze schakelaars zijn op een of andere manier mechanisch gekoppeld zodat



Figuur 1.15: *De logische AND-OR-schakeling.*

bij bediening beide schakelaars tegelijk van stand veranderen. De logische functie die deze schakeling realiseert is:

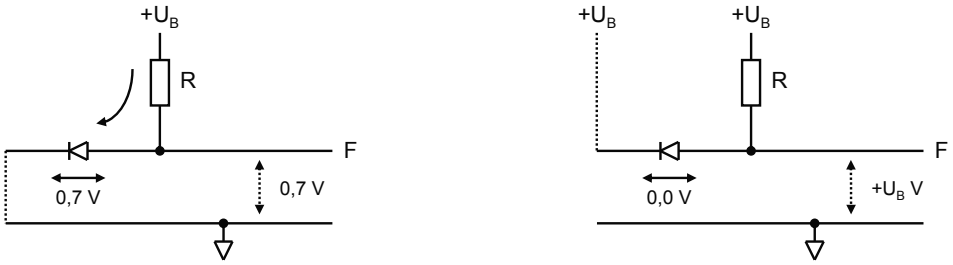
$$L = (A \cdot C) + (B \cdot C) \tag{1.7}$$

De lamp brandt als schakelaars A en C óf B en C geactiveerd zijn.

*1.7.2 Schakelingen met diodes

De *diode* is een elektronische component die de stroom in één richting goed geleidt, maar nauwelijks in de andere richting. Het is te vergelijken met een terugslagventiel voor een water- of luchtstroom. Er is wel een nadeel: de diode gaat pas goed geleiden bij een werkspanning van 0,7 V[5]. Dit wordt de doorlaatspanning genoemd. In het Engels wordt de term *forward voltage drop* gebruikt.

In figuur 1.16(a) is de diode geschakeld in doorlaat. De *kathode*, aan de kant van het streepje in het diodesymbool, is verbonden met de referentie. De *anode* is verbonden met een weerstand. De diode geleidt nu stroom. De spanning over de diode is ongeveer 0,7 V. De spanning op punt F is dus ook 0,7 V. Merk op dat de spanning op punt F nooit lager kan worden dan 0,7 V. In figuur 1.16(b) is de kathode van de diode verbonden met de voedingsspanning. De diode geleidt nu niet want de spanning op punt F kan nooit boven de voedingsspanning uitkomen. We zeggen dan dat de diode spert. Op punt F wordt dus de voedingsspanning gemeten.



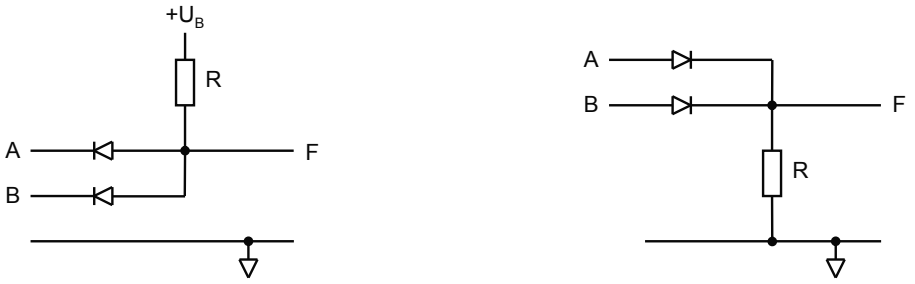
(a) *Diode in doorlaat.* **(b)** *Diode in sper.*

Figuur 1.16: *Diode in doorlaat- en sperrichting.*

Met diodes en weerstanden zijn logische schakelingen te bouwen. We gebruiken de termen laag en hoog om de werking te verklaren. Met laag bedoelen we een spanning van rond de referentie. Met hoog bedoelen we een spanning van rond de voedingsspanning.

[5] Dit geldt voor de siliciumdiode. Er zijn ook diodes die een lagere of hogere werkspanning hebben.

In figuur 1.17(a) is de logische AND-schakeling te zien. Als beide ingangen A en B met de referentie zijn verbonden, dat wil zeggen laag, dan geleiden beide diodes. De uitgangsspanning op F is dan laag, zo'n 0,7 V. Als ingang A laag is en ingang B is hoog, dan geleidt de diode bij ingang A. De diode bij ingang B spert. De uitgang is dan laag. Een zelfde situatie doet zich voor als ingang A hoog is en ingang B is laag. Als beide ingangen hoog zijn dan sperren beide diodes. De uitgang is dan hoog.



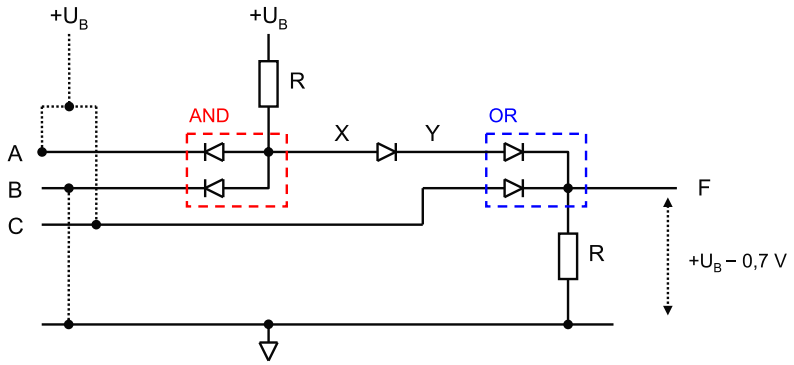
(a) *AND-schakeling met diodes.* **(b)** *OR-schakeling met diodes.*

Figuur 1.17: *AND- en OR-schakeling met diodes.*

Voor de logische OR-schakeling in figuur 1.17(b) geldt dat de uitgangsspanning laag is als beide ingangen laag zijn. Als een van de ingangen of beide ingangen hoog zijn, dan geleiden de betreffende diodes en is de uitgang hoog (zo'n 0,7 V lager dan de voedingsspanning).

Het wordt trouwens nu wel duidelijk waarom diodes nodig zijn in de AND- en OR-schakelingen. Zonder de diodes ontstaat er kortsluiting als de ingangen een verschillend spanningspotentiaal hebben.

We kunnen de AND- en OR-schakelingen combineren tot een grotere schakeling. In figuur 1.18 is een combinatie van beide schakelingen te zien. Ingangen A en B zijn verbonden met de AND-schakeling, ingang C is samen met Y verbonden met de OR-schakeling. In de figuur zijn de ingangen ook aan de voedingsspanning of referentie verbonden. In de geschetste situatie is de spanning op punt X 0,7 V want de diode bij ingang B geleidt. De reden van de diode tussen X en Y wordt nu duidelijk. Zonder deze diode zou de diode tussen Y en F kunnen gaan geleiden want 0,7 V is immers de werkspanning van een diode. Nu geleiden de diodes tussen X en F niet. Ingang C is



Figuur 1.18: *AND-OR-schakeling met diodes.*

verbonden met de voeding, dus de diode tussen C en F geleidt. De uitgangsspanning op F is dan wel lager dan de voedingsspanning, ongeveer 0,7 V lager.

Alle combinaties van A, B en C zijn in een waarheidtabel te zetten, te zien in tabel 1.6. We hebben hier een vertaling gemaakt van spanningen naar logische nullen en enen. Een lage spanning wordt gezien als een logische 0, een hoge spanning wordt gezien als een logische 1 (positieve logica).

Tabel 1.6: *Waarheidstabel AND-OR-schakeling met diodes.*

A	B	C	X	F	geleiden/sperren diodes A, B, C, X, Y
0	0	0	0	0	diodes A, B geleiden, andere sperren
0	0	1	0	1	diodes A, B, C geleiden, andere sperren
0	1	0	0	0	diode A geleidt, andere sperren
0	1	1	0	1	diodes A, C geleiden, andere sperren
1	0	0	0	0	diode B geleidt, andere sperren
1	0	1	0	1	diodes B en C geleiden, andere sperren
1	1	0	1	1	diodes X, Y geleiden, andere sperren
1	1	1	1	1	diodes C, X, Y geleiden, andere sperren

De functie laat zich als volgt omschrijven: de uitgang F is 1 als C 1 is en/of A en B beide 1 zijn. In de andere gevallen is uitgang F gelijk aan 0. In formulevorm:

$$F = (A \cdot B) + C \tag{1.8}$$

We merken zijdelings op dat het niet mogelijk is om een NOT-schakeling te maken met alleen diodes en weerstanden. Zie [35] voor meer informatie over schakelingen met diodes.

*1.7.3 Schakelingen met transistoren

In de praktijk worden elektronische schakelaars gebruikt in de vorm van transistoren[6]. Een transistor is in wezen een stroomversterker. We zullen de transistor beschouwen als een ideale schakelaar. Dat houdt in dat de schakelaar of open is (geleidt geen stroom, weerstand is oneindig groot) of gesloten (geleidt stroom, weerstand is 0 Ω).

In figuur 1.19(a) is een schakeling met een transistor en twee weerstanden te zien. De transistor heeft drie aansluitingen: de *basis* (b), de *collector* (c) en de *emitter* (e). De werking is als volgt. In de getekende situatie is punt A verbonden met de voedingsspanning. Daardoor vloeit er door de basis een stroom. Dit heeft tot gevolg dat de weerstand tussen collector en emitter, dus in de transistor, zeer klein wordt. De transistor geleidt dan stroom tussen collector en emitter. Daardoor neemt de spanning over weerstand R_C toe en de spanning op punt F neemt af. Deze spanning is ongeveer 0 V. Als punt A met de referentie wordt verbonden, vloeit er geen stroom door de basis en dan spert de transistor tussen collector en emitter. Er vloeit dan geen stroom door weerstand R_C en de spanning op punt F is gelijk aan de voedingsspanning. Resumerend kunnen we

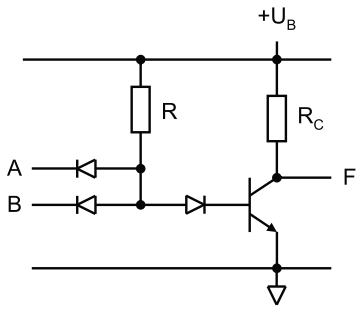
[6] We bespreken alleen de bipolaire NPN-transistor.

(a) *Transistor geschakeld als NOT-poort.* **(b)** *Model van de NOT-poort.*

Figuur 1.19: *Transistorschakelingen.*

zeggen dat een hoge ingangsspanning wordt omgezet in een lage uitgangsspanning en een lage ingangsspanning wordt omgezet in een hoge uitgangsspanning. Het geheel werkt als een NOT-schakeling. Verder moet worden opgemerkt dat in de getekende situatie de spanning op punt b ongeveer 0,7 V is. De basis-emitter-overgang gedraagt zich namelijk als een diode. De gehele schakeling kan gemodelleerd worden zoals te zien is in figuur 1.19(b). In de getekende situatie is er een geleidend pad tussen collector en emitter.

Door toevoegen van drie diodes is een NAND-schakeling te maken, zie figuur 1.20. NAND is een samentrekking van NOT en AND. De diodes bij *A* en *B* vormen samen met weerstand *R* een AND-schakeling. De rest gedraagt zich als een NOT-schakeling. De derde diode is nodig om ervoor te zorgen dat de transistor niet gaat geleiden als een van de ingangen is verbonden of beide ingangen zijn verbonden met de referentie. De spanning over diodes A en/of B is dan 0,7 V en dan kan (zonder de derde diode) de basis-emitter-overgang net gaat geleiden. De werkspanning van de basis-emitter-overgang is namelijk ook 0,7 V. Er vloeit dan stroom door de basis waardoor de transistor geleidt tussen collector en emitter.



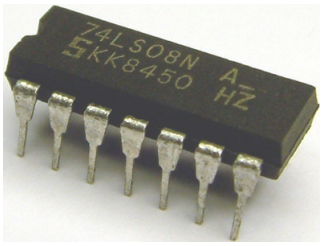
Figuur 1.20: *Een NAND-schakeling.*

Het gebruik van de transistor is drieledig. Ten eerste kan een NOT-schakeling gerealiseerd worden. Ten tweede kan de uitgangsspanning schakelen tussen de referentie en de voedingsspanning. De uitgang heeft geen last van de diodespanningsval. Er zijn nu grotere schakelingen te realiseren met behoud van logische niveaus. We noemen dit *restoring logic*. Ten derde zorgt de transistor voor stroomversterking. Er kunnen nu meerdere ingangen aangestuurd worden of er kan er een zware last geschakeld worden.

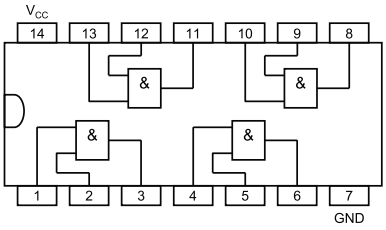
Te denken valt aan een motor, een led en een spoel. Er is wel een nadeel: deze transistorschakeling heeft altijd een inverterende werking. Een AND-schakeling moet dus gerealiseerd worden door een NAND- en een NOT-schakeling in serie te schakelen.

1.8 Symbolen voor logische poorten

Midden jaren '60 van de vorige eeuw kregen ontwerpers de beschikking over *Integrated Circuits* (ic's of *chips*) met daarin enkele *poorten*. Een poort (Engels: gate) is een elektronische schakeling die een bepaalde logische functie vervult. Een voorbeeld daarvan is te zien in figuur 1.21(a). De foto toont een "7408 Quadruple 2-input AND gate" uit de beroemde TTL-serie [36]. Het ic bevat vier AND-poorten met drie aansluitingen elk. Daarnaast zijn nog twee voedingsspanningsaansluitingen nodig zodat het totaal op veertien aansluitingen komt. De pinaansluitingen zijn te zien in figuur 1.21(b).



(a) *Foto 7408 AND.*



(b) *Pinaansluitingen.*

Figuur 1.21: *7408 AND-gate.*

Het voordeel van deze ic's was dat ze kleiner waren dan schakelingen die met discrete componenten waren opgebouwd en de ontwerper hoefde zich minder druk te maken over de dimensionering van de schakeling. Er is een hele reeks van ic's gemaakt, meer dan 100 stuks, waarvan de typenummers allemaal beginnen met 74. Zo zijn er ic's met eenvoudige poorten, maar ook met complexere schakelingen.

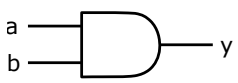
Elke digitale schakeling is te realiseren met de drie basisoperaties AND, OR en NOT. Een complexe schakeling vereist meerdere van deze operaties. Het is handig om een logische schakeling weer te geven met een schakelschema (of schema) met daarin grafische symbolen voor de poorten die de operaties voorstellen.

In de onderstaande paragrafen behandelen we de logische poorten. Van elke poort wordt de werking beschreven in termen van de logische waarden 0 en 1. Daarnaast worden de grafische symbolen gegeven. Deze symbolen zijn gedefinieerd in de normbladen voor IEC/IEEE-symbolen [37, 38]. In de Engelstalige literatuur en CAD-tekenpakketten wordt echter veelvuldig gebruik gemaakt van de "distinctive shape" symbolen (vormsymbolen) die zijn afgeleid van de MIL-STD-806 [39]. In dit boek gebruiken we de IEC/IEEE-symbolen.

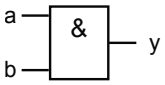
AND-poort

De uitgang van een AND-poort wordt logisch 1 als beide ingangen logisch 1 zijn, anders wordt de uitgang logisch 0. De symbolen zijn te vinden in figuur 1.22, links het

vormsymbool, rechts het IEC/IEEE-symbool. In tabel 1.7 is de waarheidstabel gegeven.



(a) *Vormsymbool.*



(b) *IEC/IEEE-symbool.*

Figuur 1.22: *Symbolen AND-poorten.*

Tabel 1.7: *AND-functie.*

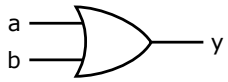
a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

De logische functie is:

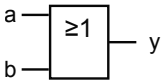
$$y = a \cdot b \tag{1.9}$$

OR-poort

De uitgang van een OR-poort wordt logisch 1 als een of meer ingangen logisch 1 is of zijn, anders wordt de uitgang logisch 0. De symbolen zijn te vinden in figuur 1.23, links het vormsymbool, rechts het IEC/IEEE-symbool. In tabel 1.8 is de waarheidstabel gegeven.



(a) *Vormsymbool.*



(b) *IEC/IEEE-symbool.*

Figuur 1.23: *Symbolen OR-poorten.*

Tabel 1.8: *OR-functie.*

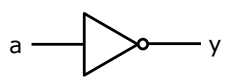
a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

De logische functie is:

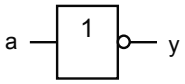
$$y = a + b \tag{1.10}$$

NOT-poort

De uitgang van een NOT-poort (of *inverter*) wordt logisch 1 als de ingang logisch 0 is, anders wordt de uitgang logisch 0. De symbolen zijn te vinden in figuur 1.24, links het vormsymbool, rechts het IEC/IEEE-symbool. In tabel 1.9 is de waarheidstabel gegeven.



(a) *Vormsymbool.*



(b) *IEC/IEEE-symbool.*

Figuur 1.24: *Symbolen NOT-poorten.*

Tabel 1.9: *NOT-functie.*

a	y
0	1
1	0

De logische functie is:

$$y = \overline{a} \tag{1.11}$$

Hiervoor zijn de drie basispoorten besproken. Hiermee kan elke digitale schakeling worden ontworpen en gebouwd. In de praktijk worden nog andere poorten gebruikt. Deze poorten zijn niet strikt noodzakelijk, wel heel handig. Ze worden gebruikt bij veel voorkomende bewerkingen. Al deze poorten kunnen worden opgebouwd uit AND, OR en NOT. We behandelen nog de buffer, EXOR, NAND, NOR en EXNOR.

Buffer

De buffer geeft de ingangswaarde een op een door. Het heeft geen "echte" logische werking. Buffers worden gebruikt om de uitgangsstroom van een schakeling te vergroten of om een tijdvertraging te introduceren. De symbolen zijn te vinden in figuur 1.25. De waarheidstabel is te vinden in tabel 1.10.

(a) *Vormsymbool.* **(b)** *IEC/IEEE-symbool.*

Figuur 1.25: *Symbolen buffers.*

Tabel 1.10: *Buffer-functie.*

a	y
0	0
1	1

De logische functie is:

$$y = a \tag{1.12}$$

EXOR-poort

De uitgang van een EXOR-poort wordt logisch 1 als exact een van de ingangen logisch 1 is, anders wordt de uitgang logisch 0. Anders gezegd: de uitgang wordt logisch 1 als de ingangen ongelijk aan elkaar zijn, anders 0. De symbolen zijn te vinden in figuur 1.26, links het vormsymbool, rechts het IEC/IEEE-symbool. In tabel 1.11 is de waarheidstabel gegeven. EXOR-poorten spelen een grote rol bij rekenschakelingen.

(a) *Vormsymbool.* **(b)** *IEC/IEEE-symbool.*

Figuur 1.26: *Symbolen EXOR-poorten.*

Tabel 1.11: *EXOR-functie.*

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

De logische functie is

$$y = a \oplus b \tag{1.13}$$

NAND-poort

NAND is een samentrekking van NOT en AND. De uitgang van een NAND-poort wordt logisch 0 als beide ingangen logisch 1 zijn, anders wordt de uitgang logisch 1. De symbolen zijn te vinden in figuur 1.27, links het vormsymbool, rechts het IEC/IEEE-symbool. In tabel 1.12 is de waarheidstabel gegeven.

(a) *Vormsymbool.* **(b)** *IEC/IEEE-symbool.*

Figuur 1.27: *Symbolen NAND-poorten.*

Tabel 1.12: *NAND-functie.*

a	b	y
0	0	1
0	1	1
1	0	1
1	1	0

De logische functie is:

$$y = \overline{a \cdot b} \tag{1.14}$$

NOR-poort

NOR is een samentrekking van NOT en OR. De uitgang van een NOR-poort wordt logisch 0 als een of meer ingangen logisch 1 is of zijn, anders wordt de uitgang logisch 1. De symbolen zijn te vinden in figuur 1.28, links het vormsymbool, rechts het IEC/IEEE-symbool. In tabel 1.13 is de waarheidstabel gegeven.

(a) *Vormsymbool.* **(b)** *IEC/IEEE-symbool.*

Figuur 1.28: *Symbolen NOR-poorten.*

Tabel 1.13: *NOR-functie.*

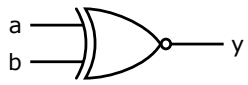
a	b	y
0	0	1
0	1	0
1	0	0
1	1	0

De logische functie is:

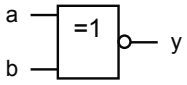
$$y = \overline{a + b} \tag{1.15}$$

EXNOR-poort

EXNOR is een samentrekking van NOT en EXOR. De uitgang van een EXNOR-poort wordt logisch 0 als exact één van de ingangen logisch 1 is, anders wordt de uitgang logisch 1. Anders gezegd: de uitgang wordt logisch 1 als de ingangen gelijk zijn aan elkaar, anders is de uitgang logisch 0. De EXNOR-poort wordt ook wel de gelijkheidspoort (Engels: equivalence gate) genoemd. De symbolen zijn te vinden in figuur 1.29, links het vormsymbool, rechts het IEC/IEEE-symbool. In tabel 1.14 is de waarheidstabel gegeven. EXNOR-poorten spelen een grote rol bij rekenschakelingen.



(a) *Vormsymbool.*



(b) *IEC/IEEE-symbool.*

Figuur 1.29: *Symbolen EXNOR-poorten.*

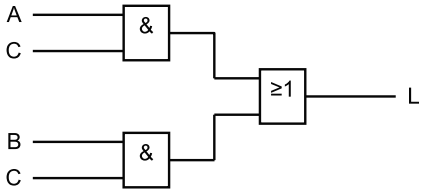
Tabel 1.14: *EXNOR-functie.*

a	b	y
0	0	1
0	1	0
1	0	0
1	1	1

De logische functie is:

$$y = \overline{a \oplus b} \tag{1.16}$$

Met behulp van de symbolen kunnen we een schakeling grafisch weergeven. In figuur 1.30 is het schema te zien van een schakeling die de logische functie van (1.7) vervult. In hoofdstuk 4 gaan we hier dieper op in.



Figuur 1.30: *Schema AND-OR-schakeling met poortsymbolen.*

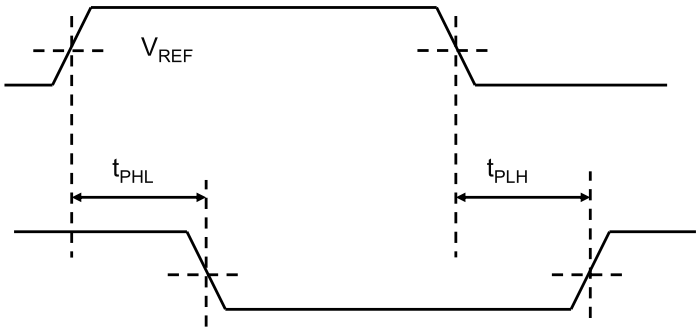
We hebben drie manieren om een schakeling te beschrijven: waarheidstabel, logische functie en schema. Deze drie vormen geven geen informatie over de elektrische eigenschappen en het tijdgedrag.

1.9 Tijdgedrag van digitale schakelingen

Een verandering op de ingang van een poort geeft als resultaat een verandering op de uitgang. Dit gaat niet oneindig snel. Schakelingen hebben enige tijd nodig om op een ingangsverandering te reageren. Dat komt onder andere omdat transistoren parasitaire capaciteiten hebben en die moeten worden opgeladen of ontladen. Het duurt dus even voordat de uitgangsverandering zichtbaar wordt. Deze tijd wordt *vertragingstijd* (Engels: propagation delay) genoemd. De fabrikant geeft in de datasheet op hoe lang deze vertragingstijd duurt met daarbij de meetopstelling en -instellingen. De relaties tussen ingangen en uitgangen worden getekend in een tijdvolgordediagram (timingdiagram).

In figuur 1.31 is een eerste introductie te zien. De tijden worden gemeten tussen een bepaalde referentiespanning V_{REF}. De tijd t_{PHL} (high to low) is de tijd die de schakeling nodig heeft om de *uitgang* van hoog naar laag te krijgen, t_{PLH} (low to high) is de tijd die de schakeling nodig heeft om de uitgang van laag maar hoog te krijgen.

De referentiespanning V_{REF} is bij CMOS in de regel de helft van de voedingsspanning. Bij TTL is de referentiespanning 1,5 V. Nu stelt deze spanning geen logische waarde voor. Thijssen laat in [40] zien dat het tijdmodel uitgebreid moet worden door stijg- en



Figuur 1.31: *Timingdigram.*

daaltijden van de signalen mee te nemen. Wij zullen dat niet doen omdat het tijdmodel dan te ingewikkeld wordt.

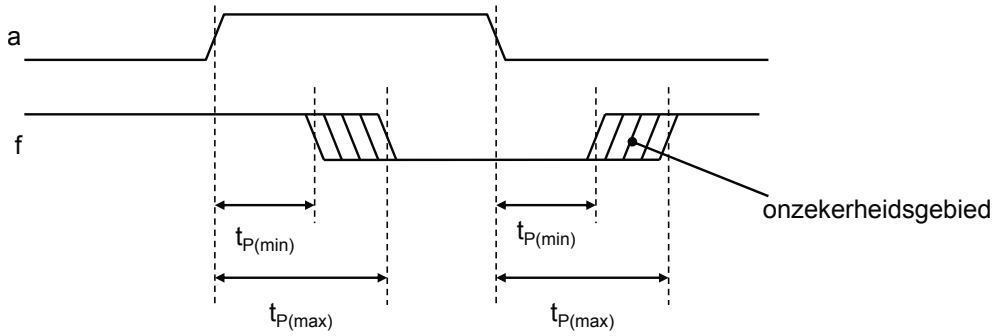
Bij componenten (losse poorten, library-modules) worden de volgende tijden vermeld:

$t_{P(min)}$ De fabrikant garandeert dat de uitgang nooit eerder verandert dan de minimale vertragingstijd.

$t_{P(max)}$ De fabrikant garandeert dat de uitgang nooit later verandert dan de maximale vertragingstijd.

$t_{P(typ)}$ De typische vertragingstijd. Leuk om te weten, maar er kan niet mee ontworpen worden.

We zullen deze tijden toelichten aan de hand van de timing van een NOT-poort, zie figuur 1.32. Als gevolg van de verandering van de ingang zal de uitgang veranderen, maar niet eerder dan $t_{P(min)}$ en niet later dan $t_{P(max)}$. Er is dus een bepaalde onzekerheid over wanneer de uitgang van waarde verandert. Dat wordt in de figuur aangegeven met het gearceerde gebied. We noemen dit het onzekerheidgebied.



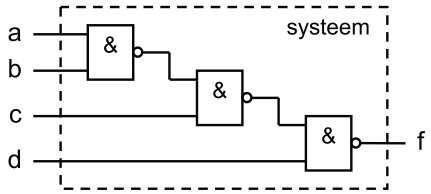
Figuur 1.32: *Timingdiagram van een NOT-poort.*

In tabel 1.15 zijn de timingparameters van de 74F00 NAND-poort te zien. De fabrikant geeft zeer gedetailleerde informatie. Niet alleen worden voedingsspanningvariaties opgegeven, maar ook temperatuurbereik en belasting. Daarnaast worden ook de tijden van hoog naar laag en laag naar hoog opgegeven. Willen we rekening houden met alle mogelijke situaties, dan moeten uitgaan van het slechtste geval. Voor de opgegeven poort geldt dat $t_{P(min)} = 1{,}5$ ns en $t_{P(max)} = 6{,}5$ ns.

Tabel 1.15: *Timingparameters van een 74F00 Quad 2-input NAND [41].*

SYMBOL	PARAMETER	TEST CONDITION	LIMITS							UNITS
			V_{CC} = +5V T_{amb} = +25 °C C_L = 50 pF, R_L = 500 Ω			V_{CC} = +5V ± 10% T_{amb} = 0 °C to +70 °C C_L = 50 pF, R_L = 500 Ω		V_{CC} = +5V ± 10% T_{amb} = –40 °C to +70 °C C_L = 50 pF, R_L = 500 Ω		
			min	typ	max	min	max	min	max	
t_{PLH} t_{PHL}	Propagation delay A, B to $\overline{Q}$	Waveform 1	2.4 2.0	3.7 3.2	5.0 4.3	2.4 2.0	6.0 5.3	2.0 1.5	6.5 6.0	ns

Laten we eens een digitaal systeem met drie van deze NAND-poorten onderzoeken. Het schema is te zien in figuur 1.33. Voor alle poorten gelden de bovengenoemde tijden. We gaan uit van het slechtste geval. Dan is de minimale vertragingstijd van het systeem 1,5 ns. De onderste ingang is namelijk verbonden met de meeste rechtse poort. Dus is een uitgangsverandering niet eerder te zien dan 1,5 ns na een verandering op de ingang. De maximale vertragingstijd is 19,5 ns, namelijk drie maximale poortvertragingen vanaf de bovenste twee ingangen. Samenvat: $t_{P(min)}$(systeem) = 1,5 ns en $t_{P(max)}$(systeem) = 3 × 6,5 = 19,5 ns. We zien dat de globale vertragingstijden een grotere onzekerheid geven dan de vertragingen van de individuele poorten. Naar mate we uitzoomen wordt het onzekerheidsgebied groter.



Figuur 1.33: *Globale tijden van een systeem.*

De huidige generatie ic's is zeer snel. Vertragingstijden liggen in de orde van enkele tienden van een nanoseconde tot enkele nanoseconden. In tabel 1.16 zijn enkele frequenties en bijbehorende tijden gegeven volgens de bekende formule $f = 1/T$. Om de orde van grootte aan te geven: 1 nanoseconde is 10^{-9} seconde. Er gaan een miljard nanoseconden in één seconde.

Tabel 1.16: *Enige frequenties en tijden.*

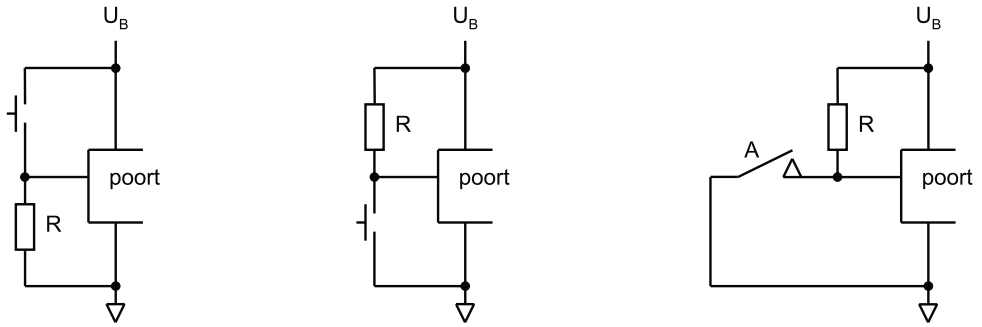
Frequentie	Frequentie	Tijd	Tijd
1 Hz	1 Hz	1 s = 1000 ms	1 s
1 kHz	10^3 Hz	1 ms = 1000 µs	10^{-3} s
1 MHz	10^6 Hz	1 µs = 1000 ns	10^{-6} s
10 MHz	$10 \cdot 10^6$ Hz	100 ns = 0,1 µs	$100 \cdot 10^{-9}$ s
100 MHz	$100 \cdot 10^6$ Hz	10 ns	$10 \cdot 10^{-9}$ s
1 GHz	10^9 Hz	1 ns = 1000 ps	10^{-9} s
1 THz	10^{12} Hz	1 ps	10^{-12} s

We hebben hier slechts een korte introductie gegeven van timing bij schakelingen waarvan de uitgang alleen afhankelijk is van de ingangen. De timing van geheugenschakelingen is ingewikkelder. We gaan daar in hoofdstuk 6 dieper op in.

1.10 Schakelaars en leds aan ingangen en uitgangen

Schakelaars en leds kunnen op verschillende manieren met ingangen en uitgangen van poorten verbonden worden. We bespreken een aantal mogelijkheden die voorkomen in praktische ontwerpen.

In figuur 1.34(a) is een drukknop aangesloten tussen de voedingsspanning en de ingang van een poort. De weerstand zorgt voor een geleidend pad naar de referentie. Dit wordt een *pull-down* weerstand genoemd. In de getekende situatie in de ingang laag, want de drukknop is niet ingedrukt. Bij indrukken wordt de ingang hoog. We noemen dit *actief hoog* (Engels: active high). In figuur 1.34(b) is de drukknop verbonden met de ingang van een poort en de referentie. De weerstand zorgt voor een geleidend pad van de voeding naar de ingang. Dit wordt een *pull-up* weerstand genoemd. In deze situatie is de ingang hoog. Bij indrukken van de drukknop wordt de ingang laag. We noemen dit *actief laag* (Engels: active low). Figuur 1.34(c) laat een ander type schakelaar zien. Het betreft een tuimelschakelaar. Ook hier werkt de schakelaar actief laag.

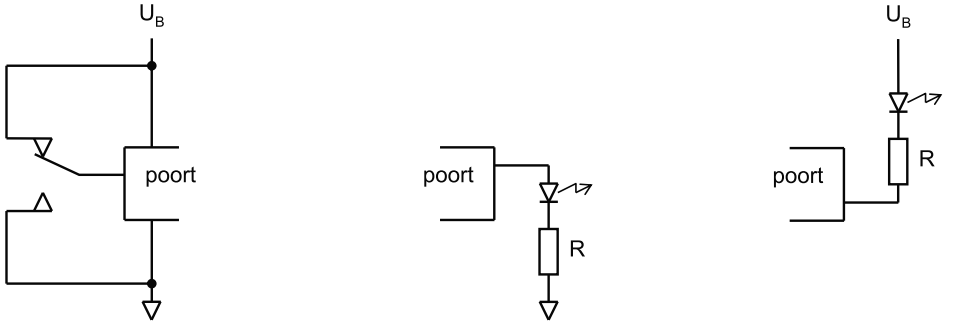


(a) *Drukknop actief hoog.* **(b)** *Drukknop actief laag.* **(c)** *Schakelaar actief laag.*

Figuur 1.34: *Aansluitingen drukknoppen en schakelaar.*

Er zijn ook schakelaars die twee geleidende paden hebben. Dat is te zien in figuur 1.35(a). Nu is er geen weerstand nodig. In de ruststand in de ingang hoog.

In de figuren 1.35(b) en 1.35(c) zijn de twee configuraties om een led aan te sluiten te zien. Ook hier spreken we over actief hoog en actief laag. De weerstanden zorgen voor begrenzing van de stroom door de leds.



(a) *Tweestanden schakelaar.* **(b)** *Led actief hoog.* **(c)** *Led actief laag.*

Figuur 1.35: *Aansluitingen tweestandenschakelaar en leds.*

Historisch gezien kwamen de actief lage schakelingen het meest voor. Dat komt omdat de uitgang van een TTL-poort behoorlijk wat stroom kan opnemen (Engels: sink), maar weinig stroom kan leveren (Engels: source). Zie hiervoor tabel 1.3. Een actief hoge uitgang zou betekenen dat een extra transistor moet worden gebruikt om de benodigde stroom te kunnen leveren. Voor wat betreft de ingang geldt hetzelfde. De ingang kan veel minder stroom opnemen dan leveren. De pull-down weerstand in figuur 1.34(a) moet dan klein gekozen worden om niet boven de spanning van een logische 0 te komen. Dat heeft dan weer tot gevolg dat bij activering van de schakelaar de stroom door de weerstand behoorlijk groot is.

1.11 Een eerste blik op logische schakelingen

Vanuit een waarheidstabel of logische functie kan een schakeling worden opgebouwd. In deze paragraaf behandelen hoe we een schakeling opbouwen. We gebruiken positieve logica en gaan hierbij uit van het principe van “enen sparen”. Door een logische functie te vereenvoudigen kan een schakeling met een minimum aan poorten worden gerealiseerd. We introduceren het begrip dualiteit dat te maken met het uitwisselen van AND- en OR-poorten.

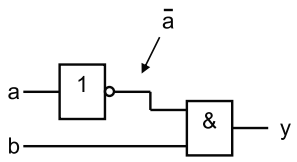
Lezen van een functie en opbouw van een schakeling

In figuur 1.36 is de waarheidstabel van een functie te zien. De functie moet een logische 1 geven voor alle enen in de y-kolom, anders moet de functie een logische 0 geven. We gaan hierbij uit van positieve logica. De enige regel met een 1 lezen we als volgt: $y = 1$ als $a = 0$ én $b = 1$. We kunnen dit in een functie schrijven:

$$y = \overline{a} \cdot b \tag{1.17}$$

Als we de overige drie combinaties invullen wordt y inderdaad logisch 0.

a	b	y
0	0	0
0	1	1
1	0	0
1	1	0



Figuur 1.36: *Waarheidstabel en schema de functie.*

Om de functie te realiseren, hebben we een een NOT-poort en een AND-poort nodig. De AND-poort geeft een logische 1 als beide ingangen logisch 1 zijn. Maar de functie moet een logische 1 geven als a logisch 0 is en b logisch 1 is. We hebben dus een NOT-poort nodig om a te inverteren. Zo zorgen we ervoor dat de AND-poort een logische 1 geeft als a logisch 0 is en b logisch 1. Voor de overige combinaties van a en b geeft de schakeling een logische 0.

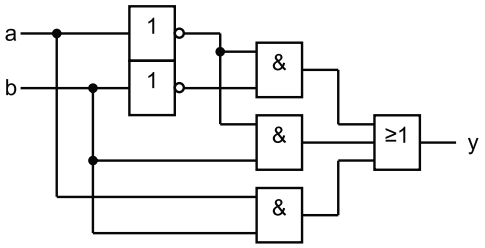
Vereenvoudigen

Stel dat we de schakeling willen bouwen zoals is aangegeven in de waarheidstabel in figuur 1.37. De werking van de schakeling is als volgt: uitgang $y = 1$ als $a = 0$ en $b = 0$ óf $a = 0$ en $b = 1$ óf $a = 1$ en $b = 1$, anders $y = 0$. De logische functie is:

$$y = \overline{a} \cdot \overline{b} + \overline{a} \cdot b + a \cdot b \tag{1.18}$$

Rechts in figuur 1.37 is de schakeling te zien. Dit kost nogal wat poorten: twee NOT-poorten, drie AND-poorten en een OR-poort. Het aantal ingangen is 11. Merk op dat we voor de inverse van a maar één NOT-poort gebruiken.

a	b	y
0	0	1
0	1	1
1	0	0
1	1	1

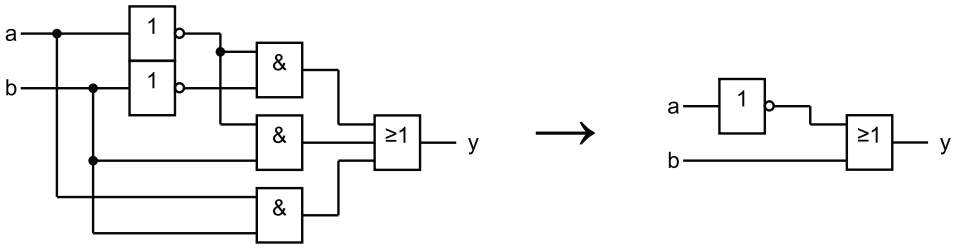


Figuur 1.37: *Waarheidstabel en schema voor de te vereenvoudigen functie.*

We zien dat de functie een logische 1 oplevert als $a = 0$. Dat is bij de eerste twee regels. Dus we kunnen schrijven $y = 1$ als $a = 0$. Ook is te zien dat de uitgang een logisch 1 is als $b = 1$. Dus: $y = 1$ als $b = 1$. Verder geldt dat $y = 0$ als $a = 1$ en $b = 0$. We kunnen de functie nu vereenvoudigen tot $y = 1$ als $a = 0$ en/of $b = 1$ anders geldt $y = 0$. Dus:

$$y = \overline{a} \cdot \overline{b} + \overline{a} \cdot b + a \cdot b = \overline{a} + b \tag{1.19}$$

We kunnen nu de schakeling met veel minder poorten en aansluitingen realiseren, zie figuur 1.38.



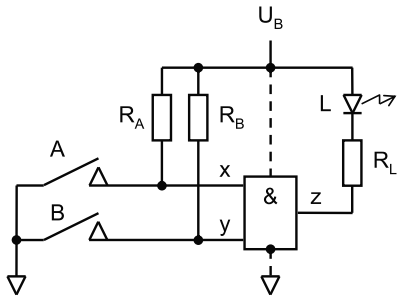
Figuur 1.38: *Vereenvoudigde realisatie van een logische functie.*

Merk op dat beide schakelingen logisch identiek zijn. Ze vervullen dezelfde logische functie.

Dualiteit en De Morgan

Het begrip dualiteit kunnen we het beste uitleggen aan de hand van een voorbeeld. In figuur 1.39 is een schakeling rond een AND-poort te zien. We gaan uit van positieve logica, dus een schakelaar in rust en een led die gedoofd is, worden als logisch 0 gezien.

In de getekende situatie brandt de led niet. De schakelaars zijn open en de ingangen x en y worden door de weerstanden naar een hoog spanningsniveau getrokken. Uitgang z is dan ook hoog zodat er geen stroom door de led vloeit. Als schakelaar A geactiveerd wordt, is ingang x laag en wordt uitgang z ook laag. Nu vloeit er stroom door de led. Alle combinaties van A en B zijn te zien in de waarheidstabel naast de schakeling.



A	B	x	y	z	L
0	0	1	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	0	1

Figuur 1.39: *Schakeling met AND-poort en waarheidstabel.*

Het omkeren van de logische niveaus zorgt ervoor dat de AND-poort zich als een OR-poort gedraagt. Vervangen we de poort door een OR-poort dan gedraagt de hele schakeling zich als een AND-poort. Dit wordt het dualiteitsprincipe genoemd. Als in een schakeling alle logische waarden worden verwisseld en alle AND- en OR-poorten worden verwisseld, dan gedraagt de schakeling zich identiek.

Het dualiteitsprincipe is de basis van de stellingen van De Morgan:

$$\begin{aligned} y = a \cdot b \quad &\longrightarrow \quad \overline{y} = \overline{a} + \overline{b} \\ y = a + b \quad &\longrightarrow \quad \overline{y} = \overline{a} \cdot \overline{b} \end{aligned} \tag{1.20}$$

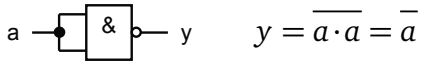
De stellingen laten zien dat AND- en OR-poorten uitgewisseld kunnen worden.

1.12 Universele bouwstenen

De NOT-AND-poort of NAND-poort is een universele digitale bouwsteen. Elke digitale schakeling kan met alleen NAND-poorten worden gemaakt. Dat geldt ook voor de NOT-OR-poort of NOR-poort. Ook deze poort is een universele digitale bouwsteen. Elke digitale schakeling kan met alleen NOR-poorten worden gemaakt. Om dit aan te tonen, moeten we laten zien dat de drie basispoorten kunnen worden gerealiseerd met NAND- of NOR-poorten.

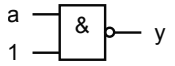
In figuur 1.40 zijn de vier configuraties te zien van NAND- en NOR-poorten geschakeld als NOT-poorten. Bij elke configuratie is de wiskundige beschrijving gegeven. Elk van de vier schakelingen realiseert een NOT-functie. Het maakt niet uit welke gebruikt wordt want de logische werking is identiek.

De schakeling in figuur 1.41 realiseert een AND-poort. De linker NAND-poort realiseert de NAND-functie van a en b. De rechter NAND-poort is geschakeld als NOT-poort. In totaal realiseert de schakeling dus een NOT-NAND-poort. In de wiskundige beschrijving is goed te zien dat de termen $\overline{a \cdot b}$ samengenomen worden. Daarna volgt de dubbele negatie.



$$y = \overline{a \cdot a} = \overline{a}$$

(a) *NOT-poort gemaakt met NAND-poort.*

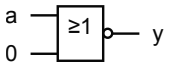


$$y = \overline{a \cdot 1} = \overline{a}$$

(b) *NOT-poort gemaakt met NAND-poort.*

$$y = \overline{a + a} = \overline{a}$$

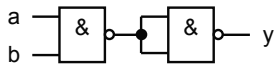
(c) *NOT-poort gemaakt met NOR-poort.*



$$y = \overline{a + 0} = \overline{a}$$

(d) *NOT-poort gemaakt met NOR-poort.*

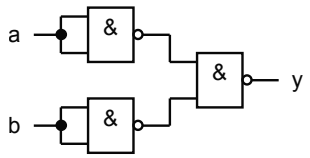
Figuur 1.40: *NOT-poorten gemaakt met NAND- en NOR-poorten.*



$$y = \underbrace{\overline{\overline{a \cdot b} \cdot \overline{a \cdot b}}}_{\text{samennemen}} = \overline{\overline{a \cdot b}} = a \cdot b$$

Figuur 1.41: *AND-poort gemaakt met NAND-poorten.*

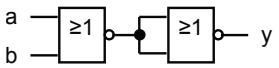
In figuur 1.42 is een schakeling te zien die een OR-poort realiseert met drie NAND-poorten. De NAND-poorten links inverteren de ingangen, de NAND-poort rechts realiseert de NAND-constructie van de geïnverteerde ingangen. In de wiskundige beschrijving is te zien dat eenmaal de stelling van De Morgen is toegepast.



$$y = \overline{\overline{a \cdot a} \cdot \overline{b \cdot b}} = \underbrace{\overline{\overline{a} \cdot \overline{b}}}_{\text{De Morgan}} = a + b$$

Figuur 1.42: *OR-poort gemaakt met NAND-poorten.*

De schakeling in figuur 1.43 realiseert een OR-poort. De linker NOR-poort realiseert de NOR-functie van a en b. De rechter NOR-poort is geschakeld als NOT-poort. In totaal realiseert de schakeling dus een NOT-NOR-poort. In de wiskundige beschrijving is goed te zien dat de termen $\overline{a + b}$ samengenomen worden. Daarna volgt de dubbele negatie.



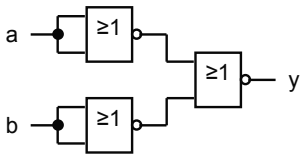
$$y = \underbrace{\overline{\overline{a + b} \cdot \overline{a + b}}}_{\text{samennemen}} = \overline{\overline{a + b}} = a + b$$

Figuur 1.43: *OR-poort gemaakt met NOR-poorten.*

In figuur 1.44 is een schakeling te zien die een AND-poort realiseert met drie NOR-poorten. De NOR-poorten links inverteren de ingangen, de NOR-poort rechts realiseert de NOR-constructie van de geïnverteerde ingangen. In de wiskundige beschrijving is te zien dat eenmaal de stelling van De Morgen is toegepast.

We kunnen nu de NAND-en NOR-poorten gebruiken om elke functie te realiseren. Elke functie in de vorm van $f = a \cdot b + c \cdot d$ kan worden omgezet naar een functie met alleen NAND-poorten:

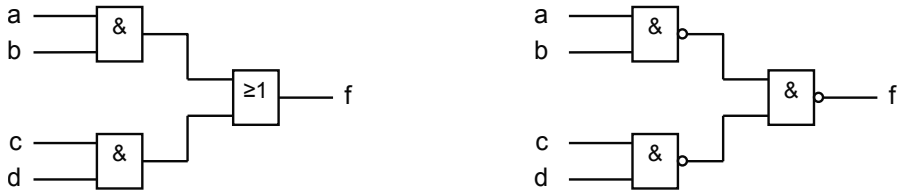
$$f = a \cdot b + c \cdot d = \overline{\overline{a \cdot b} \cdot \overline{c \cdot d}} \tag{1.21}$$



$$y = \overline{\overline{a+a} + \overline{b+b}} = \underbrace{\overline{a}+\overline{b}}_{\text{De Morgan}} = a \cdot b$$

Figuur 1.44: *OR-poort gemaakt met NOR-poorten.*

Dit betekent dat een schakeling die gerealiseerd is in de AND-OR-vorm kan worden omgezet naar een schakeling met alleen NAND-poorten. Dit is te zien in figuur 1.45.

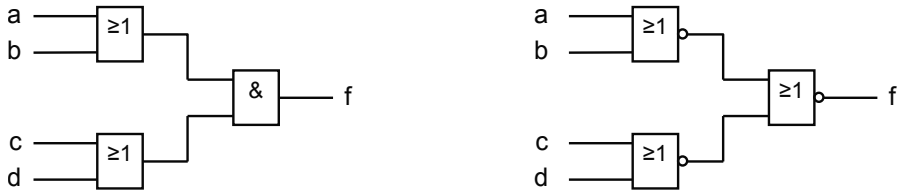


Figuur 1.45: *AND-OR-schakeling omgezet naar NAND-NAND-schakeling.*

Op eenzelfde wijze kan een schakeling in de vorm van $f = (a+b) \cdot (c+d)$ direct worden omgezet naar een schakeling met alleen NOR-poorten:

$$f = (a+b) \cdot (c+d) = \overline{\overline{a+b} + \overline{c+d}} \tag{1.22}$$

Een schakeling gerealiseerd in de OR-AND-vorm is dus om te zetten in een schakeling met alleen NOR-poorten. Dit is te zien in figuur 1.46.



Figuur 1.46: *OR-AND-schakeling omgezet naar NOR-NOR-schakeling.*

In hoofdstuk 4 zullen we nog een universele bouwsteen bespreken: de multiplexer.

1.13 Opgaven

1.1. Ontwerp een omschakelbare buffer/NOT. Dit is een schakeling die onder besturing van een stuursignaal de ingangswaarde doorgeeft (buffer) of inverteert (NOT).

1.2. Ontwerp een NOT-schakeling met een NAND-poort.

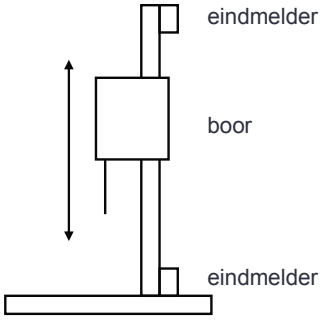
1.3. Ontwerp een NOT-schakeling met een NOR-poort.

1.4. Bepaal de waarheidstabel van een EXOR waarvan één ingang geïnverteerd is.

1.5. De beschrijving van een AND is: de uitgang is 1 als alle ingangen 1 zijn. Hoe zou de beschrijving zijn als er van de nullen wordt uitgegaan?

1.6. Hoeveel verschillende functies zijn er te maken met twee variabelen?

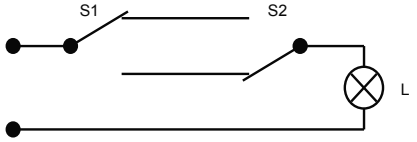
1.7. Een automatische kolomboor (figuur P1.1) heeft twee eindmelders: een aan de bovenkant en een aan de onderkant. Een melder kan open zijn (boor is daar niet) of gesloten zijn (boor is daar wel).



Figuur P1.1: *Kolomboor.*

Hoeveel combinaties van open en gesloten zijn er mogelijk? Welke combinatie komt nooit voor? Stel een tabel op met alle mogelijkheden en geef met een paar woorden aan wat de mogelijkheden inhouden.

1.8. Een elektrisch schema met schakelaars kan ook als een digitaal systeem worden gezien. Een bekende schakeling is de zogeheten wisselschakeling die veel bij trappen voorkomt. Deze schakeling wordt ook wel de hotelschakeling genoemd. Zie figuur P1.2 voor het schakelschema.

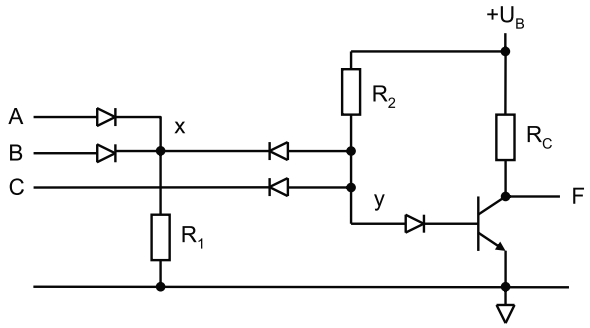


schakelaar in rust: 0
schakelaar geactiveerd: 1
lamp uit: 0
lamp brandt: 1

Figuur P1.2: *Schema wisselschakeling en beschrijving logische signalering.*

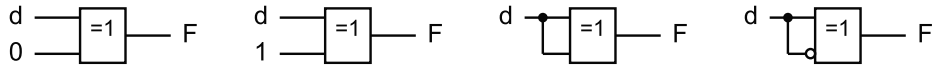
Geef de waarheidstabel van de wisselschakeling.

* **1.9.** In figuur P1.3 is een schakeling gegeven met diodes, weerstanden en een transistor. Stel een waarheidstabel op voor alle signalen in de figuur.



Figuur P1.3: *Transistorschakeling.*

1.10. Aan een EXOR-poort met twee ingangen wordt één ingang als volgt afwisselend aangesloten: 0, 1, normaal of geïnverteerd. Zie hiervoor figuur P1.4. Hieruit is op te maken dat er slechts één ingangssignaal is, namelijk d. Daardoor vereenvoudigd de werking van de poort. Bepaal de werking van deze vier mogelijkheden.



Figuur P1.4: *Vier EXOR-poorten.*

1.11. Als opgave 1.10 maar nu met een EXNOR-poort.

1.12. Toon aan dat een EXOR-poort door middel van een extra NOT-poort kan worden omgezet in een ENXOR-poort.

1.13. Bepaal de eenvoudigste vorm van de functies in tabel P1.1.

Tabel P1.1: *Drie waarheidstabellen.*

a	b	y
0	0	1
0	1	1
1	0	1
1	1	0

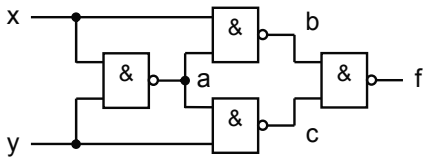
a	b	y
0	0	1
0	1	0
1	0	1
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

1.14. Ontwerp een buffer, NAND, NOR en EXNOR met behulp van NANDs.

1.15. Gegeven de functie: $y = \overline{a} \cdot b + a \cdot \overline{b}$
Ontwerp deze functie met alleen NANDs.

1.16. Gegeven de schakeling in figuur P1.5. Bepaal f als functie van x en y.



Figuur P1.5: *Schakeling met NAND-poorten.*

1.17. Gegeven de volgende functie: $f = (a + b) \cdot (c + d)$
Ontwerp voor deze functie een schakeling met alleen NOR-poorten.

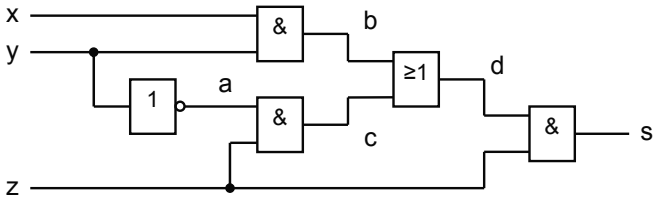
1.18. Gegeven de waarheidstabel in tabel P1.2.

Tabel P1.2: *Waarheidstabel*

a	b	y
0	0	1
0	1	1
1	0	0
1	1	1

Ontwerp de bijbehorende schakeling met alleen NANDs. Ontwerp de bijbehorende schakeling met alleen NORs. Dit mogen ook poorten zijn met meer dan twee ingangen.

1.19. Gegeven onderstaand schema. De vertragingstijden van de poorten is gegeven in tabel ernaast. Bepaal de minimale en maximale vertragingstijd van deze schakeling.



Figuur P1.6: *Een schakeling.*

Tabel P1.3: *Vertragingstijden in ns.*

Poort	$t_{P(min)}$	$t_{P(max)}$
AND	2,4	6,9
OR	3,1	7,2
NOT	1,5	4,3

1.20. Wat is de periodetijd van een signaal met een frequentie van 33,3 MHz? En 3,4 GHz? Wat is de frequentie van van een signaal met een periodetijd van 20 ns? En 104,167 μs?

2

Talstelsels en codes

Het begrip getal stelt ons in staat om allerlei zaken te kunnen tellen en zo een hoeveelheid van iets voor te kunnen stellen. Het voorstellen gebeurt niet altijd op dezelfde wijze. We zouden knikkers kunnen gebruiken om een bepaalde hoeveelheid weer te geven [42] of *cijfers* (symbolen) gebruiken die een bepaalde hoeveelheid voorstellen.

Een getal is een abstractie van een hoeveelheid. Om niet voor elke hoeveelheid een ander cijfer nodig te hebben zijn *talstelsels* bedacht. Het eenvoudigste voorbeeld is het eentallig stelsel [43]. In dit talstelsel worden getallen weergegeven met slechts één cijfer; het cijfer 1. Het getal dertien wordt in dit stelsel weergegeven als 1111111111111 en het getal 7 als 1111111. Berekeningen uitvoeren in dit stelsel is niet eenvoudig, laat staan het weergeven van grote getallen (probeer in dit stelsel maar eens het getal duizend op te schrijven). Om deze reden zijn andere talstelsels bedacht waarmee we eenvoudiger kunnen rekenen, bijvoorbeeld het tientallig stelsel dat we dagelijks gebruikten. Natuurlijk moet een talstelsel zo geconstrueerd zijn dat de bewerkingen zo eenvoudig mogelijk kunnen worden uitgevoerd, iets wat in bijvoorbeeld het Romeinse talstelsel nog knap lastig is [44].

Getallen kunnen ook gebruikt worden om informatie anders dan het getal zelf weer te geven. Een getal kan onderdeel zijn van een *code*. Zo kan een getal gebruikt worden om een letter voor te stellen. Hiermee is het mogelijk om tekst als getallen weer te geven zodat het in een computergeheugen kan worden opgeslagen.

In dit hoofdstuk beginnen we met een korte introductie van het decimale talstelsel. Vervolgens behandelen we de talstelsels die gebruikt worden in de digitale techniek zoals het binaire en hexadecimale talstelsel. Er wordt aandacht besteed aan de constructie van de getallen en het omrekenen tussen de diverse talstelsels. Naast gehele getallen bestaan er ook fracties, of delen van een geheel, en zijn er combinaties mogelijk van gehele getallen en fracties. We besluiten het hoofdstuk met een uiteenzetting over enkele codes die in digitale systemen gebruikt worden.

We behandelen in dit hoofdstuk alleen niet-negatieve getallen, getallen groter dan of

gelijk aan 0. Negatieve getallen worden behandeld in hoofdstuk 5.

2.1 Het decimale talstelsel

In het dagelijks leven werken we met het tientallig of *decimale talstelsel*. Hierin worden de getallen opgebouwd uit de cijfers 0 tot en met 9; het getal 10 is een samenstelling van de cijfers 1 en 0. Elk decimaal getal wordt opgebouwd uit eenheden, tientallen, honderdtallen, duizendtallen, enzovoorts. Zo kan het getal 2539 worden geschreven als:

$$2 \cdot 1000 + 5 \cdot 100 + 3 \cdot 10 + 9 \cdot 1$$

De 9 is als cijfer veel meer waard dan de 2 maar door de *positie* van de 2 stelt dit getal de waarde 2000 voor. Het decimale talstelsel is een zogenoemd *positioneel* talstelsel. De 1, 10, 100 en 1000 worden *gewichten* genoemd en geven de afzonderlijke cijfers meer waarde aan het gehele getal.

Nu zijn 1, 10, 100 en 1000 allemaal machten van 10 zodat het getal ook geschreven kan worden als:

$$2 \cdot 10^3 + 5 \cdot 10^2 + 3 \cdot 10^1 + 9 \cdot 10^0$$

Hierin is het getal 10 het *grondtal* van het decimale talstelsel en de gewichten zijn een macht van 10. De kleine getallen 3, 2, 1, en 0 zijn de exponenten en geven de positie van de afzonderlijke cijfers aan. Het cijfer 2 staat geheel aan de linkerkant. Dit wordt het *meest significante cijfer* genoemd. Geheel rechts staat het cijfer 9. Dit wordt het *minst significante cijfer* genoemd. Merk op dat de macht aan de rechterkant een exponent 0 heeft en volgens de wiskunde is 10^0 gelijk aan 1. In Engelstalige literatuur worden voor grondtal de woorden *radix* en *base* gebruikt.

Algemeen kan een decimaal getal A dat bestaat uit n cijfers als volgt genoteerd worden:

$$A = a_{n-1}a_{n-2}\cdots a_1 a_0 \tag{2.1}$$

waarbij $a_{n-1}\cdots a_0$ de afzonderlijke cijfers voorstellen. De waarde van het getal A is te schrijven als een sommatie (optelling) van de waarden van de afzonderlijke cijfers vermenigvuldigd met hun gewicht:

$$A = a_{n-1} \cdot 10^{n-1} + a_{n-2} \cdot 10^{n-2} \ \ldots \ a_1 \cdot 10^1 + a_0 \cdot 10^0 \tag{2.2}$$

Dit kan met behulp van het sommatieteken ook als volgt worden genoteerd:

$$A = \sum_{i=0}^{n-1} a_i \cdot 10^i \tag{2.3}$$

Hierin is i de *sommatieindex* en begint op de ondergrens 0 ($i = 0$) en wordt steeds met 1 verhoogd tot en met de bovengrens $n-1$. Voor elke waarde van de sommatieindex i moet het product $a_i \cdot 10^i$ uitgerekend worden. Deze producten moeten bij elkaar opgeteld worden (sommatie) om het uiteindelijke resultaat te krijgen.

2.2 Het binaire talstelsel

Elektronische schakelingen voor het bewerken en onthouden van 0 en 1 zijn veel eenvoudiger dan die voor het bewerken en onthouden van 0 t/m 9. Vandaar dat in de computertechniek het tweetallig of *binaire talstelsel* wordt gebruikt. In het binaire talstelsel zijn slechts de cijfers 0 en 1 beschikbaar. Zo zou een willekeurig binair getal dat bestaat uit acht cijfers geschreven kunnen worden als:

11011011

Het is nu echter onduidelijk of dit getal in het binaire of decimale talstelsel genoteerd staat, vandaar dat het getal voorzien wordt van een subscript. Het hiervoor gegeven getal wordt dan geschreven als:

11011011_2

of

11011011_{BIN}

Het omzetten van een binair getal naar het decimale equivalent gaat vrij eenvoudig. Elk binair cijfer (*binary digit* of *bit* genoemd) levert een bijdrage aan het totale getal en moet hiervoor vermenigvuldigd worden met zijn gewicht. Het cijfer kan alleen 0 of 1 zijn en het gewicht is een macht van 2. Zo staat in het bovenstaande getal het linker bit op positie 7 en is het bijbehorende gewicht 2^7. De bijdrage is dus $1 \cdot 2^7$ oftewel 128.

$$\begin{aligned} 11011011_2 &= 1\cdot2^7 + 1\cdot2^6 + 0\cdot2^5 + 1\cdot2^4 + 1\cdot2^3 + 0\cdot2^2 + 1\cdot2^1 + 1\cdot2^0 \\ &= 1\cdot128 + 1\cdot64 + 0\cdot32 + 1\cdot16 + 1\cdot8 + 0\cdot4 + 1\cdot2 + 1\cdot1 \\ &= 128 + 64 + 0 + 16 + 8 + 0 + 2 + 1 \\ &= 128 + 64 + 16 + 8 + 2 + 1 \\ &= 219 \end{aligned}$$

Het binaire getal 11011011_2 is dus gelijk aan het decimale getal 219_{10}. Zoals te zien is worden de gewichten vermenigvuldigd met 1 of 0. Een vermenigvuldiging met 1 levert het gewicht zelf op, een vermenigvuldiging met 0 levert 0 op. Deze nullen worden bij het omzetten dan ook niet genoteerd want de bijdrage is toch 0. Van de enen wordt alleen het gewicht genoteerd.

Binaire getallen worden om de leesbaarheid te bevorderen vaak geschreven in blokken van 4 bits, gescheiden door een punt of een spatie. De reden hiervoor is dat moderne microprocessoren eenheden van 8, 16 of 32 bits verwerken. Leidende nullen worden dan ook geschreven zodat het aantal binaire cijfers een veelvoud van 4 is. In computerterminologie wordt een eenheid van 4 bits een *nibble* genoemd. Een eenheid van 8 bits wordt een *byte* genoemd[1]. Een byte bestaat dus uit twee nibbles. In de datacommunicatie wordt de term *octet* gebruikt om een eenheid van acht bits aan te duiden.

[1] Dat was niet altijd zo. Vroeger wat een byte het aantal bits dat gebruikt werd om een karakter in een computer te coderen.

2.3 Het octale talstelsel

Over het binaire talstelsel kunnen twee opmerkingen gemaakt worden:

- De lengte van een binair getal is ongeveer een factor 3,3 groter dan de lengte van het decimale getal met dezelfde waarde[2].
- Lange binaire getallen zijn lastig te overzien en zijn gevoelig voor fouten.

Vooral de lengte van de binaire getallen levert een probleem op. Het is voor mensen lastig om grote groepen nullen en enen te bewerken [45]. Om binaire getallen goed te kunnen overzien wordt het achttallig of *octale* talstelsel gebruikt. Dit talstelsel gebruikt de cijfers 0 t/m 7. Een voorbeeld van een octaal getal is:

$$7651_8$$

of

$$7651_{\mathrm{OCT}}$$

Omzetten naar een decimaal equivalent gaat als volgt:

$$\begin{aligned} 127_8 &= 1\cdot 8^2 + 2\cdot 8^1 + 7\cdot 8^0 \\ &= 1\cdot 64 + 2\cdot 8 + 7\cdot 1 \\ &= 87_{10} \end{aligned}$$

De keuze voor het octale talstelsel is niet zomaar genomen. Het grondtal 8 is een macht van 2. Dat houdt in dat elk octaal cijfer met precies drie bits is weer te geven. Zo is 5_8 te schrijven als 101_2.

Het octale talstelsel werd vroeger veel gebruikt omdat computers in die tijd 12-bits, 24-bits en 36-bits eenheden gebruikten en deze bitlengtes zijn precies deelbaar door 3. Huidige computers gebruiken veelvouden van 8 bits waardoor het lastig is om octale getallen te gebruiken.

Het octale talstelsel is nu in de vergetelheid geraakt. Sporen ervan zijn terug te vinden in de programmeertaal 'C' en de Unix en Linux besturingssystemen, zoals de commando's `chmod` (change file mode) en `od` (octal dump).

2.4 Het hexadecimale talstelsel

Rond 1970 produceerde Intel de eerste microprocessor, de 4004. De opdracht voor het ontwerp kwam van het Japanse bedrijf Busicom dat de processor in een programmeerbare rekenmachine wilde gebruiken. Deze processor werkte met 4-bits eenheden. De reden hiervoor is dat de rekenmachine rekende in het decimale talstelsel, maar de afzonderlijke cijfers werden met bits opgeslagen [46]. Zie ook paragraaf 2.11. De opvolger van de 4004, de 8008, werkte met 8 bits, zodat twee cijfers in één keer bewerkt konden worden. Latere processors werkten met 16 bits, 32 bits etc.

[2] Dit is uit te rekenen als $^2\log 10$, zie paragraaf 2.9.

Nu kunnen met vier bits zestien combinaties gemaakt worden. Om toch niet met binaire getallen te werken wordt het zestientallig of *hexadecimale* talstelsel gebruikt. In dit talstelsel worden 16 “cijfers” gebruikt met de waarden 0 t/m 15. We gebruiken de eerste tien cijfers van het decimale talstelsel voor de eerste tien cijfers van het hexadecimale talstelsel en voor de overige zes cijfers worden de eerste zes letters van het alfabet gebruikt. De cijfers 0 t/m 9 worden weergegeven als 0 t/m 9. De letters die als hexadecimale cijfers voor de waarden 10 t/m 15 worden gebruikt zijn dus A, B, C, D, E en F. In plaats van hoofdletters kunnen ook kleine letters gebruikt worden. Het hangt vaak af van de persoonlijke voorkeur van de gebruiker. In dit boek worden hoofdletters gebruikt.

Voorbeelden van de hexadecimale notatie zijn:

$3F5A_{16}$ $3F5A_{HEX}$

Het grondtal 16 is een macht van 2. Dat houdt in dat elk hexadecimaal cijfer precies met vier bits is weer te geven, zoals te zien is in tabel 2.1. Uit de tabel blijkt dat de binaire getallen opgesteld zijn volgende de *binaire telcode*.

Tabel 2.1: *Diverse representaties van getallen.*

dec	oct	hex	bin	dec	oct	hex	bin
0	0	0	0000	8	10	8	1000
1	1	1	0001	9	11	9	1001
2	2	2	0010	10	12	A	1010
3	3	3	0011	11	13	B	1011
4	4	4	0100	12	14	C	1100
5	5	5	0101	13	15	D	1101
6	6	6	0110	14	16	E	1110
7	7	7	0111	15	17	F	1111

Hexadecimale getallen worden met dezelfde methode die bij de binaire getallen besproken is, op overeenkomstige wijze geschreven en omgerekend naar het decimale talstelsel. Zo is het getal $3F5A_{16}$

$$\begin{aligned} 3F5A_{16} &= 3\cdot16^3 + 15\cdot16^2 + 5\cdot16^1 + 10\cdot16^0 \\ &= 3\cdot4096 + 15\cdot256 + 5\cdot16 + 10\cdot1 \\ &= 16218_{10} \end{aligned}$$

Een hexadecimaal getal van twee cijfers past precies in een byte. Grotere getallen vergen meer bytes, per twee cijfers één byte. Zo bestaat het 32-bits hexadecimale getal $1234ABCD_{16}$ uit vier bytes met de waarden 12_{16}, 34_{16}, AB_{16} en CD_{16}. Grote hexadecimale getallen kunnen weer in blokken van vier cijfers worden geschreven, gescheiden door een punt. Dit bevordert de leesbaarheid.

Hexadecimale getallen worden o.a. gebruikt om de adresruimte van een computer te beschrijven. Een voorbeeld is een computer met 16-bits adressen dat RAM-geheugen geplaatst heeft op de adressen $C000_{16}$–$FFFF_{16}$.

Internetadressen bij IPv6 bestaan uit 128 bits en worden weergegeven met 32 hexadecimale cijfers. Bij het weergeven wordt een dubbele punt gebruikt om 16-bits eenheden te scheiden. Een voorbeeld van een adres is 0123:4567:90AB:CDEF:0123:4567:90AB:CDEF.

De programmeertaal 'C' gebruikt de *prefix* (voorvoegsel) "0x" om aan te geven dat het om de hexadecimale notatie gaat, bijvoorbeeld `0xFF00FF00`.

2.5 Fracties

Fracties (delen van getallen kleiner dan 1) kunnen op dezelfde wijze worden geschreven als de gehele getallen, alleen zijn de exponenten nu negatief. Dat houdt in dat de gewichten kleiner zijn dan 1. Voorbeeld van een decimale fractie is:

$$0{,}639 = 6\cdot10^{-1} + 3\cdot10^{-2} + 9\cdot10^{-3} = \frac{6}{10} + \frac{3}{100} + \frac{9}{1000}$$

Cijfers na de komma worden uitgedrukt in tienden, honderdsten, duizendsten etc. Binaire fracties worden op vergelijkbare wijze geschreven, maar natuurlijk is het grondtal nu 2. De binaire cijfers worden uitgedrukt in een half, een kwart, een achtste, een zestiende etc. Het omzetten van een binaire fractie naar het decimale equivalent gaat ook op dezelfde wijze als bij gehele getallen:

$$0{,}1101_2 = 2^{-1} + 2^{-2} + 2^{-4} = \frac{1}{2} + \frac{1}{4} + \frac{1}{16} = 0{,}5 + 0{,}25 + 0{,}1625 = 0{,}8125_{10}$$

Op vergelijkbare wijze kan de hexadecimale fractie worden omgezet:

$$\begin{aligned} 0{,}3\text{F2A}_{16} &= 3\cdot16^{-1} + 15\cdot16^{-2} + 2\cdot16^{-3} + 10\cdot16^{-4} \\ &= \frac{3}{16} + \frac{15}{256} + \frac{2}{4096} + \frac{10}{65536} \\ &= 0{,}1875 + 0{,}05859375 + 0{,}00048828125 + 0{,}000152587890625 \\ &= 0{,}246734619140625_{10} \end{aligned}$$

2.6 Conversie tussen binair, hexadecimaal en octaal

Conversie (het omzetten van een getal in het ene talstelsel naar een getal in het andere talstelsel) tussen binaire, hexadecimale en octale getallen is gemakkelijk. Dit komt omdat in deze talstelsels de grondtallen een macht van 2 zijn.

Hexadecimale getallen kunnen eenvoudig naar binaire getallen omgezet worden door elk hexadecimaal cijfer te vervangen met de 4-bits waarde:

$$\text{F7A4.D3C1}_{16} = 1111.0111.1010.0100.1101.0011.1100.0001_2$$

Een voorbeeld met een fractie:

$$\text{3F5A,79F}_{16} = 0011.1111.0101.1010{,}0111.1001.1111_2$$

De punten zijn om het lezen te vergemakkelijken. Bij het omzetten van binair naar hexadecimaal moet het binaire getal eerst met een veelvoud van vier bits worden geschreven.

Bij het gehele deel moeten voor de meest significante bit nullen geplaatst worden. Bij de fractie moeten nullen achter de minst significante bit geplaatst worden:

$$\begin{aligned} 1111011001{,}101011_2 &= 11.1101.1001{,}1010.11_2 \\ &= 0011.1101.1001{,}1010.1100_2 \\ &= 3D9{,}AC_{16} \end{aligned}$$

In het octale talstelsel kunnen groepen van drie bits voorgesteld worden met één octaal cijfer. Een voorbeeld van een conversie naar octaal naar binair:

$$472_8 = 100.111.010_2$$

en met een fractie:

$$345{,}67_8 = 011.100.101{,}110.111_2$$

In deze voorbeelden zijn de binaire getallen opgedeeld in groepjes van drie bits om de overeenkomst tussen de octale cijfers en de bijbehorende binaire getallen goed te laten zien.

Bij conversie tussen octale en hexadecimale getallen wordt gebruik gemaakt van het binaire talstelsel als tussenstap. Een voorbeeld van het omzetten van hexadecimaal naar octaal:

$$\begin{aligned} F4{,}A9_{16} &= 1111.0100{,}1010.1001_2 \\ &= 11.110.100{,}101.010.01_2 \\ &= 011.110.100{,}101.010.010_2 \\ &= 364{,}522_8 \end{aligned}$$

Merk op dat er extra nullen nodig zijn om het gehele deel en de fractie met een veelvoud van drie bits te schrijven. Conversie tussen octale en hexadecimale getallen gaat zonder toevoegen van nullen als de getallen (binair geschreven) een veelvoud zijn van 12 bits.

2.7 Grondtalconversie

Het omrekenen van een getal met een willekeurig grondtal naar het decimale talstelsel is niet zo'n probleem. Willen we een decimaal getal naar een ander talstelsel omzetten, dan moet een andere procedure gevolgd worden. De methoden voor gehele getallen en fracties zijn verschillend en worden apart behandeld.

2.7.1 Conversie gehele getallen

Om te laten zien hoe het converteren werkt, laten we eerst de conversie van een decimaal getal naar een decimaal getal zien. (Ja, dat is wat vreemd, maar het laat zien hoe de conversie werkt). We zetten het getal 25391 om door herhaald te delen door 10 en de rest van de deling te noteren. Daarna gaan we verder met het gehele getal dat na de deling is overgebleven. We doen dit totdat we op 0 zijn uitgekomen.

$$
\begin{array}{rcrcl}
25391 & \div 10 & = & 2539\text{ r }1 & \rightarrow\ 1 \\
2539 & \div 10 & = & 253\text{ r }9 & \rightarrow\ 9 \\
253 & \div 10 & = & 25\text{ r }3 & \rightarrow\ 3 \\
25 & \div 10 & = & 2\text{ r }5 & \rightarrow\ 5 \\
2 & \div 10 & = & 0\text{ r }2 & \rightarrow\ 2 \\
\underline{0} & & & &
\end{array}
$$

Door nu de resten na deling van onder naar boven te lezen, vinden we het getal 25391 terug. We kunnen deze methode gebruiken om een decimaal getal om te zetten naar een binair getal door herhaald te delen door 2. Als voorbeeld zetten we het getal 105_{10} om.

$$
\begin{array}{rcrcl}
105 & \div 2 & = & 52\text{ r }1 & \rightarrow\ 1 \\
52 & \div 2 & = & 26\text{ r }0 & \rightarrow\ 0 \\
26 & \div 2 & = & 13\text{ r }0 & \rightarrow\ 0 \\
13 & \div 2 & = & 6\text{ r }1 & \rightarrow\ 1 \\
6 & \div 2 & = & 3\text{ r }0 & \rightarrow\ 0 \\
3 & \div 2 & = & 1\text{ r }1 & \rightarrow\ 1 \\
1 & \div 2 & = & 0\text{ r }1 & \rightarrow\ 1 \\
\underline{0} & & & &
\end{array}
$$

We lezen de binaire cijfers van onder naar boven af en vinden dat 105_{10} gelijk is aan 1101001_2. Het is natuurlijk ook mogelijk om een decimaal getal om te zetten naar hexadecimaal door herhaald te delen door 16. Let er op dat de resten na deling liggen tussen 0 en 15 en dat de resten 10 t/m 15 vertaald moeten worden naar A t/m F.

$$
\begin{array}{rcrcl}
40810 & \div 16 & = & 2550\text{ r }10 & \rightarrow\ A \\
2550 & \div 16 & = & 159\text{ r }6 & \rightarrow\ 6 \\
159 & \div 16 & = & 9\text{ r }15 & \rightarrow\ F \\
9 & \div 16 & = & 0\text{ r }9 & \rightarrow\ 9 \\
\underline{0} & & & &
\end{array}
$$

De uitkomst moet van beneden naar boven worden uitgelezen, dus 40810_{10} is gelijk aan $9F6A_{16}$.

2.7.2 Conversie fracties

Bij het omrekenen van een decimale fractie naar een binaire fractie moet herhaald vermenigvuldigd worden met 2. Na elke vermenigvuldiging wordt het getal voor de komma (het gehele deel) genoteerd en wordt verder gegaan met de fractie. De conversie stopt als we op 0 zijn uitgekomen.

$$
\begin{array}{rcrcl}
0{,}8125 & \times 2 & = & 1{,}625 & \rightarrow\ 1 \\
0{,}625 & \times 2 & = & 1{,}25 & \rightarrow\ 1 \\
0{,}25 & \times 2 & = & 0{,}5 & \rightarrow\ 0 \\
0{,}5 & \times 2 & = & 1{,}0 & \rightarrow\ 1 \\
\underline{0} & & & &
\end{array}
$$

De cijfers moeten van boven naar beneden gelezen worden. Het decimale getal $0{,}8125_{10}$ is gelijk aan het binaire getal $0{,}1101_2$.

Het is niet altijd mogelijk om een decimale fractie exact te schrijven als een binaire fractie. Zo is het exacte getal $0{,}6_{10}$ niet exact als binaire fractie te schrijven:

$$\begin{array}{llll} 0{,}6 & \times 2 & = & 1{,}2 \rightarrow 1 \\ 0{,}2 & \times 2 & = & 0{,}4 \rightarrow 0 \\ 0{,}4 & \times 2 & = & 0{,}8 \rightarrow 0 \\ 0{,}8 & \times 2 & = & 1{,}6 \rightarrow 1 \\ 0{,}6 & \times 2 & = & \cdots \end{array}$$

Na vier keer vermenigvuldigen komen we weer uit op $0{,}6_{10}$. Dit proces zal nooit stoppen. De vier genoteerde cijfers zullen zich blijven herhalen. Het binaire getal heeft een repeterende binaire breuk. We kunnen dit schrijven door gebruik te maken van een onderstreping die aangeeft welk deel zich herhaald:

$$0{,}6_{10} = 0{,}100110011001\ldots_2 = 0{,}\underline{1001}_2$$

Hieruit volgt een belangrijke ontdekking:

$0{,}6_{10}$ *is niet exact te schrijven in het binaire talstelsel.*

Een computer kan dit decimale getal alleen bij benadering weergeven. We zullen moeten afronden op een aantal bits na de komma[3].

Uiteraard zijn fracties ook als hexadecimale getallen te schrijven.

$$\begin{array}{rlrr} 0{,}615 & \times 16 & = & 9{,}84 \rightarrow 9 \\ 0{,}84 & \times 16 & = & 13{,}44 \rightarrow \mathrm{D} \\ 0{,}44 & \times 16 & = & 7{,}04 \rightarrow 7 \\ 0{,}04 & \times 16 & = & 0{,}64 \rightarrow 0 \\ 0{,}64 & \times 16 & = & 10{,}24 \rightarrow \mathrm{A} \\ 0{,}24 & \times 16 & = & 3{,}84 \rightarrow 3 \\ 0{,}84 & \times 16 & = & \cdots \end{array}$$

Ook dit getal is niet exact te representeren. We schrijven het getal als $0{,}9\underline{\mathrm{D70A3}}_{16}$. De onderstreping geeft aan welk deel zich herhaalt.

2.7.3 Afronden fracties

Afronden vindt plaats op basis van het eerste niet weer te geven cijfer na de komma[4] (het relevante cijfer). Voor binaire getallen is dat:

0 : afronden naar beneden

[3] De vraag is uiteraard na hoeveel bits er moet worden afgerond. Dat hangt af van de beoogde nauwkeurigheid. Het wordt hier niet besproken.

[4] Noot: afronden volgens de Citogroep.

1 : afronden naar boven

Laten we het getal $0{,}6_{10}$ eens afronden op een aantal bits. Hieronder zijn drie afrondingen weergegeven:

$0{,}6_{10} = 0{,}1|\underline{0}\ldots = 0{,}1_2$ (na 1 bit)
$0{,}6_{10} = 0{,}100|\underline{1}\ldots = 0{,}101_2$ (na 3 bits)
$0{,}6_{10} = 0{,}10011001|\underline{1}\ldots = 0{,}10011010_2$ (na 8 bits)

We kunnen de 8-bits fractie weer omrekenen naar het decimale talstelsel:

$$0{,}10011010_2 = 0{,}6015625_{10}$$

De afwijking t.o.v. $0{,}6_{10}$ bedraagt slechts $0{,}0015625_{10}$.

2.8 Bereik van enkele bitbreedtes

In de jaren '70 en '80 van de vorige eeuw kwamen de eerste computers voor de consumenten beschikbaar. Deze computers, zoals de populaire Commodore 64 [47] en ZX Spectrum [48], waren uitgevoerd met eenvoudige processoren waarin eenheden van 8 bits (1 byte) werden verwerkt. In de jaren '90 van de vorige eeuw werd het fabriceren van grote chips gemeengoed waardoor 16-bits en 32-bits processoren (bijvoorbeeld Intel 386) op de markt kwamen. Moderne processoren kunnen eenheden van 64 bits verwerken.

In tabel 2.2 is een aantal voorkomende bitbreedtes met toepassingen opgesomd. Het bereik van een getal is als volgt uit te rekenen:

$$\begin{aligned} &\text{kleinste waarde} = 0 \\ &\text{grootste waarde} = 2^{\text{aantal bits}} - 1 \end{aligned} \tag{2.4}$$

Zo ligt het bereik van een 8-bits getal tussen 0 en 255 ($2^8 - 1$) en van een 16-bits getal tussen 0 en 65535 ($2^{16} - 1$). Het grootste getal dat met 128 bits kan worden weergegeven is 340.282.366.920.938.463.463.374.607.431.768.211.455 ($2^{128} - 1$).

ONEINDIG EN ONEINDIG...

Het getal $0{,}6_{10}$ is binair niet exact te noteren, het levert een repeterende binaire breuk op: $0{,}100110011001\ldots._2$. Uiteraard levert $0{,}3_{10}$ dat ook op. En $0{,}2_{10}$ ook. En $0{,}1_{10}$ en $0{,}05_{10}$ etc. Er zijn dus oneindig veel decimale breuken die niet exact binair te noteren zijn. Laten we die in de verzameling A stoppen.

Er zijn ook getallen die wél binair netjes te schrijven zijn: $0{,}5_{10}$, $0{,}25_{10}$, $0{,}125_{10}$, $0{,}75_{10}$ etc. Ook dit zijn oneindig veel getallen. Laten we deze getallen in de verzameling B stoppen.

Vraag: welke verzameling heeft de meeste elementen, A of B?

Tabel 2.2: *Enige voorkomende bitbreedtes.*

Bits	Toepassing	Bereik
4	Intel 4004	$0 \leq g \leq 15$
8	Diverse processoren (8080, 6800, 6502, ATmega), ADC's en DAC's	$0 \leq g \leq 255$
10	ADC's en DAC's (AVR ATmega)	$0 \leq g \leq 1.023$
12	ADC's en DAC's	$0 \leq g \leq 4.095$
16	CD, processoren	$0 \leq g \leq 65.535$
24	Digitale audio, RGB-kleuren	$0 \leq g \leq 16.777.215$
32	Processoren (Intel 386, AMD, ARM, MIPS), IPv4 adresruimte	$0 \leq g \leq 4.294.967.295$
64	Processoren (Intel, AMD64, ARM)	$0 \leq g \leq 18.446.744.073.709.551.615$
128	IPv6 adresruimte	$0 \leq g \leq 340.282.366.920.938.463.463\ldots$

2.9 Bepaling aantal bits voor een geheel getal

Zoals gezegd wordt in het dagelijks leven gebruik gemaakt van decimale getallen. Dat zijn we nu eenmaal gewend. In de digitale techniek gebruiken we binaire getallen. Het is dan noodzakelijk om te bepalen hoeveel bits er *minimaal* nodig zijn om een bepaald decimaal getal te representeren.

Het aantal bits dat nodig is om een decimaal getal weer te geven kan bepaald worden door de ongelijkheid:

$$M \leq 2^n - 1 \tag{2.5}$$

waarbij M het decimale getal is en n het aantal bits is van het binaire equivalent. De ongelijkheid komt voort uit het feit er meer bits gebruikt mogen worden dan strikt noodzakelijk is.

In de meeste gevallen zoeken we een minimum waarde voor n. Na enig rekenwerk volgt dat:

$$n = \left\lceil \frac{\log(M+1)}{\log 2} \right\rceil \tag{2.6}$$

In de formule wordt gebruik gemaakt van de logaritme-functie. Het maakt niet uit welk grondtal voor de log-functie gebruikt wordt maar de 10-log is voor de hand liggend. Uiteraard moet n een geheel getal zijn. De bijzondere haken geven aan dat het antwoord moet worden afgerond op het laagste gehele getal dat groter is dan of gelijk is aan het oorspronkelijke antwoord. Het getal +1 lijkt overbodig (zeker als M groot is) maar levert het juiste antwoord als M precies een macht van 2 is.

Als voorbeeld berekenen we het minimum aantal bits dat nodig is om het getal 11487 weer te geven:

$$n = \left\lceil \frac{\log(11487+1)}{\log 2} \right\rceil = \left\lceil \frac{4{,}0602\ldots}{0{,}3010\ldots} \right\rceil = \lceil 13{,}4878\ldots \rceil = 14 \tag{2.7}$$

We kunnen ook uitgaan van een decimaal getal dat bestaat uit m cijfers. Er zijn dan 10^m mogelijkheden. Met n bits zijn 2^n mogelijke getallen te representeren. Het aantal mogelijk decimale getallen moet dan kleiner zijn dan of gelijk zijn aan het aantal mogelijke binaire getallen zodat volgt dat:

$$10^m \leq 2^n \tag{2.8}$$

Voor het bepalen van het minimum aantal bits volgt dat:

$$n = \left\lceil \frac{m}{\log 2} \right\rceil \tag{2.9}$$

Merk op dat nu wel de 10-log gebruikt moet worden omdat m de exponent is van een macht van 10. Het aantal bits moet natuurlijk een geheel getal zijn, er moet naar boven worden afgerond.

Stel dat we een kilometerteller met zes (decimale) cijfers willen ontwerpen. De digitale schakeling gebruikt uiteraard binaire getallen. De binaire getallen hebben dan minstens:

$$n = \left\lceil \frac{6}{\log 2} \right\rceil = \left\lceil \frac{6}{0{,}301\ldots} \right\rceil = \lceil 19{,}9315\ldots \rceil = 20 \tag{2.10}$$

bits nodig.

Uit formule (2.9) valt af te leiden dat binaire getallen ongeveer 3,3 keer zoveel cijfers hebben als het equivalente decimale getal.

2.10 Modulo rekenen

Als we op papier (of uit ons hoofd) rekenen met getallen, gebruiken we automatisch genoeg cijfers om de getallen en het resultaat weer te geven. In digitale systemen worden getallen opgeslagen met een eindig aantal bits. Dat betekent dat slechts een eindig aantal getallen kan worden weergegeven. We zetten het even op een rijtje:

- Het bereik van een n-bits getal g is $0 \leq g \leq 2^n - 1$.
- De bijbehorende bitcombinaties zijn $0\ldots0$ t/m $1\ldots1$, alle met n bits.
- Het aantal mogelijke getallen met n bits is 2^n.

Getallen groter dan $1\ldots1$ (met n bits) worden afgebeeld in het bereik $0\ldots0$ t/m $1\ldots1$ (met n bits). In feite worden de overvloedige bits gewoon weggelaten.

Bekijk de getallen 23_{10} en 7_{10} eens. Ze zijn identiek[5] als ze worden afgebeeld met 4 bits:

$$23_{10} \xrightarrow{\text{omzetting}} 10111_2 \xrightarrow{\text{met 4 bits}} 0111_2 \xrightarrow{\text{omzetting}} 7_{10}$$

De getallen worden *modulo* 2^n afgebeeld. Dat houdt in dat de *rest na deling door* 2^n wordt opgeslagen. Bij zelfontwikkelde hardware zal het niet zo veel problemen opleveren

[5] Een mooi woord hiervoor is *congruent modulo*

omdat we van te voren kunnen bepalen hoeveel bits er nodig zijn om de getallen weer te geven. Maar omdat een microprocessor werkt met een bepaalde vaste breedte voor de opslag van getallen, bijvoorbeeld 32 bits, kan het wel problemen geven. Een optelling van twee getallen kan een antwoord opleveren dat te groot is om in 32 bits op te slaan. Er is dan sprake van een zogenoemde *carry*. De processor registreert dat intern wel, maar hogere programmeertalen zoals 'C' kunnen daar niets mee[6]. De programmeur moet het zelf in de gaten houden.

2.11 BCD-code

In digitale systemen wordt voornamelijk gerekend met binaire getallen. Het binaire talstelsel is gekozen omdat elektronica voor het opslaan van binaire getallen en rekenschakelingen vrij eenvoudig is te realiseren. Het nadeel van binaire getallen is dat deze niet direct kunnen worden afgebeeld als decimale getallen, bijvoorbeeld op een beeldscherm. Hiervoor moeten binaire getallen worden omgezet naar decimale getallen. Bij het omzetten van een binair getal naar een decimaal getal moet herhaald gedeeld worden door $10_{10} = 1010_2$ en de restwaarde van de deling stelt dan één decimaal cijfer voor. Een digitale schakeling om een binair getal op deze manier om te zetten naar het decimale talstelsel kost veel poorten.

Het omzetten wordt eenvoudiger door een codering te gebruiken waarbij elk decimaal cijfer wordt opgeslagen in vier bits. Deze codering wordt *Binary Coded Decimal* (BCD) genoemd. Elk BCD-cijfer bestaat uit vier bits (nibble). Van de 16 mogelijkheden worden er 10 gebruikt (0 t/m 9) en 6 niet (10 t/m 15).

Merk op dat de BCD-code een talstelsel is, alleen worden niet alle mogelijke bitcombinaties gebruikt. De bits van een BCD-cijfer hebben wel een gewicht. De meest significante bit heeft de waarde 8, daarna de waarde 4 en 2 en de minst significante bit heeft de waarde 1. De BCD-code wordt daarom ook wel de 8421-code genoemd [49]. Een voorbeeld van een BCD-getal is:

$$1001.0101.0110.0001_{\text{BCD}} = 9561_{10} = 10.0101.0101.1001_2 \tag{2.11}$$

Rekenschakelingen voor BCD-gecodeerde getallen zijn lastig. Het afbeelden op bijvoorbeeld 7-segment displays gaat daarentegen weer heel gemakkelijk.

*2.12 Efficiëntie BCD-codering

Een BCD-cijfer gebruikt vier bits, dus een n-cijferig BCD-getal gebruikt $m_{BCD} = 4 \cdot n$ bits. Het aantal bits voor een n-cijferig decimaal getal (dus niet BCD) is:

$$m_{dec} = {}^2\log 10^n = n \cdot {}^2\log 10 = 3{,}322 \cdot n \tag{2.12}$$

[6] Het kan natuurlijk wel, maar de ontwerpers van 'C' hebben ervoor gekozen om dat niet te doen om de snelheid van de door 'C' gegenereerde uitvoerbare code te maximaliseren. Na elke bewerking controleren op een carry zou een applicatie namelijk minder snel maken en dat vonden de ontwerpers van 'C' niet wenselijk. Een aantal moderne hogere programmeertalen heeft wel mogelijkheden om te controleren op een carry.

De verhouding is:

$$r = \frac{4 \cdot n}{3{,}322 \cdot n} = 1{,}204 \tag{2.13}$$

Hieruit kan geconcludeerd worden dat voor BCD-codering 20% meer bits nodig zijn. Er zit dus nogal wat "lucht" in de BCD-codering. Dit heeft geleid tot onderzoek naar efficiëntere coderingen zoals de Chen-Ho-codering [50].

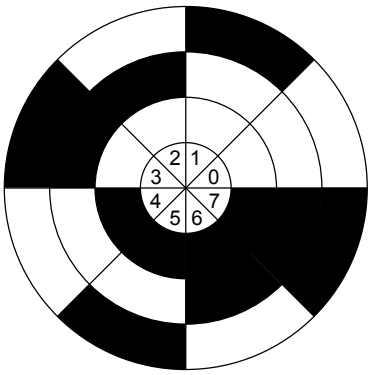
Er kan ook wat gezegd worden over het aantal nuttig gebruikte codecombinaties. Van één BCD-cijfer worden 10 van de 16 mogelijke combinaties gebruikt. Bij twee BCD-cijfers worden $10 \cdot 10 = 100$ van de $16 \cdot 16 = 256$ bitcombinaties nuttig gebruikt. Voor een getal van n BCD-cijfers worden 10^n van de 16^n combinaties nuttig gebruikt. Algemeen gezegd is de efficiëntie:

$$\eta = \frac{10^n}{16^n} = \left(\frac{10}{16}\right)^n \qquad (n > 0) \tag{2.14}$$

Een 8-cijferig BCD-getal (32 bits) heeft dus een efficiëntie van 2,3%.

2.13 Gray-code

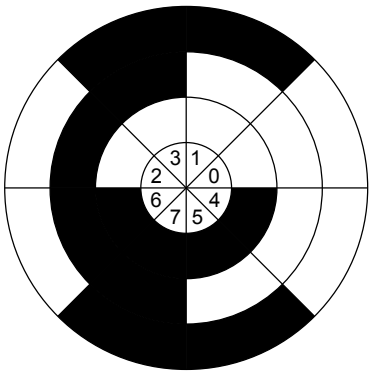
In elektromechanische apparaten, zoals printers en kopieerapparaten, is het soms nodig om de stand te weten van een roterend onderdeel. Dat kan gedaan worden met een zogenoemde *codeerschijf*. De schijf bestaat uit een aantal segmenten waarbij de codering met behulp van lampjes en lichtgevoelige opnemers kan worden bepaald. De witte segmenten laten het licht door, de zwarte segmenten onderbreken de lichtstraal. In figuur 2.1 is een variant te zien met drie bits. De codering is zuiver binair, een zwart segment levert een 1 op en een wit segment levert een 0 op (dit kan in de praktijk ook andersom geregeld worden, afhankelijk van de elektronische uitleesschakeling).



Figuur 2.1: *Een 3-bit binaire codeerschijf [51].*

Deze codeerschijf levert problemen bij de overgangen tussen de zwarte en witte gebieden. Neem als voorbeeld de overgang van 7 naar 0. Als de lampjes niet goed uitgelijnd staan dan kunnen bij de overgang van segment 7 naar segment 0 andere codecombinaties uitgelezen worden. Deze *intermitterende* codes zijn slechts kort zichtbaar maar kunnen wel problemen opleveren voor de achterliggende elektronica.

Het probleem zit in het feit dat er meer dan één bitwisseling tegelijk plaatsvindt. Dat is praktisch nooit goed te regelen. Het probleem kan worden opgelost met de codeerschijf in figuur 2.2. Hierbij wisselt steeds één bit per overgang. De gebruikte codering in de figuur wordt de *binary reflected Gray-code* of *binaire spiegelcode*[7] genoemd [52]. De Gray-code is een vorm van een *progressieve code*. Dat is een code waarbij twee opeenvolgende *codewoorden* (we spreken liever niet van getallen) slechts in één bit verschillen.



Figuur 2.2: *Een 3-bit codeerschijf met Gray-code [53].*

In tabel 2.3 is de variant voor drie bits te zien. Het opstellen van de Gray-code gaat als volgt. Bekijken we de kolom met de minst significante bit, dan zien we dat de eerste twee regels volgens de normale binaire telcode zijn opgesteld. In de volgende twee regels is de minstsignificante bit *gespiegeld* t.o.v. de eerste twee regels. De spiegellijn is aangegeven met een onderstreping. De middelste bit spiegelt om de vier regels. Algemeen geldt dat de n^e bit spiegelt om de 2^n regels.

Tabel 2.3: *De Gray-code voor drie bits*

decimaal	binair	Gray
0	000	000
1	001	00$\underline{1}$
2	010	011
3	011	0$\underline{1}$0
4	100	110
5	101	11$\underline{1}$
6	110	101
7	111	100

De Gray-code wordt in diverse toepassingen gebruikt, zoals bij toestandstoekenning (state assignment) bij asynchrone toestandsmachines, Karnaughdiagrammen (zie hoofdstuk 4) en foutencorrectie bij digitale transmissie.

[7] In zijn patent merkte Gray op dat er nog geen naam was voor de gevonden code maar hij had al wel uitgevonden dat de code “kan worden opgebouwd vanuit de conventionele binaire code door een soort van spiegeling” (vrij vertaald).

2.14 Andere codes voor decimale cijfers

In de begintijd van de computers werd er voornamelijk in het decimale stelsel gerekend. Bekende computers in die tijd waren de ENIAC en de IBM650. In deze computers werden verschillende codes gebruikt voor het weergeven van decimale cijfers. Zo werd in de IBM650 de biquinaire code [54, 55] gebruikt vanwege de opbouw van de gebruikte buizen. De ENIAC gebruikte 10-bits ringtellers om cijfers op te slaan [56].

Voor het representeren van decimale cijfers zijn ten minste vier bits nodig. Daarmee zijn zestien mogelijke binaire codewoorden te maken. Er zijn miljarden verschillende manieren om tien 4-bits codewoorden te kiezen. In tabel 2.4 is een aantal van deze codes opgesomd.

Tabel 2.4: *Enkele codes voor decimale cijfers.*

dec	8421	7421	5421	2421	XS-3	Johnson	biquinair
0	0000	0000	0000	0000	0011	00000	0100001
1	0001	0001	0001	0001	0100	00001	0100010
2	0010	0010	0010	0010	0101	00011	0100100
3	0011	0011	0011	0011	0110	00111	0101000
4	0100	0100	0100	0100	0111	01111	0110000
5	0101	0101	1000	1011	1000	11111	1000001
6	0110	0110	1001	1100	1001	11110	1000010
7	0111	1000	1010	1101	1010	11100	1000100
8	1000	1001	1011	1110	1011	11000	1001000
9	1001	1010	1100	1111	1100	10000	1010000

De bekendste code is de 8421-code, beter bekend als de BCD-code. In feite is dit de binaire telcode voor de cijfers 0 t/m 9. Zie paragraaf 2.11.

De 7421-, 5421- en de 2421-codes zijn net zoals de BCD-code voorbeelden van *gewogen codes*. De posities van de bits hebben een bepaald gewicht. Zo is bij de 2421- of *Aiken*-code het gewicht van links naar rechts 2, 4, 2 en 1. De code voor 7 is gelijk aan 1101 en uit rekenen als $2\cdot1+4\cdot1+2\cdot0+1\cdot1=7$. Merk op dat 7 ook gecodeerd zou kunnen worden als 0111. Er zijn dus afspraken nodig over de precieze invulling van de codes. De Johnson-code is een vorm van een Gray-code. Er is slechts één bitwisseling tussen

ZOEK HET ZELF MAAR UIT

Het aantal mogelijkheden om m cijfers te coderen met n bits wordt gegeven met de formule:

$$(n)_m = \frac{2^n!}{(2^n-m)!} \tag{2.15}$$

Om 10 cijfers met 4 bits te coderen zijn er dus $\frac{16!}{(16-10)!} = 29.059.430.400$ coderingen mogelijk.

twee opeenvolgende codewoorden. De code wordt gebruikt bij snelle telschakelingen.

De representatie van een cijfer in de XS-3-code (*Excess-Three*) of *Stibitz*-code wordt verkregen door 3 bij het cijfer op te tellen. Het voordeel van deze code is dat bij optellen van twee XS-3-cijfers de *uitgaangde carry* (zie hoofdstuk 5) automatisch gegenereerd wordt als de uitkomst groter is dan 9 (decimaal). Het nadeel van de XS-3-code is dat na de optelling een correctie nodig is om het resultaat om te zetten naar een decimaal cijfer.

Sommige codes hebben nog enkele nuttige eigenschappen. Zo heeft de 7421-code een minimum aantal enen in de codering. In systemen waarin de opgeslagen enen vermogen dissiperen levert dat een minimum aan energieverbruik. De 2421-code en XS-3 code zijn symmetrisch ten opzichte van het midden. Dat levert als voordeel op dat de aftrekking $9-N$ te realiseren is door eenvoudigweg de bits van N te inverteren.

2.15 ASCII-code

In de vorige paragrafen hebben we gezien dat binaire coderingen worden gebruikt om numerieke gegevens (of informatie) weer te geven. Maar niet alle gegevens zijn numeriek. Informatie kan ook bestaan uit letters en leestekens. Om er voor te zorgen dat computers informatie kunnen uitwisselen is de ASCII-code bedacht.

De naam ASCII betekent *American Standard Code for Information Interchange* en dat geeft al goed aan waarvoor de code bedoeld is: op een gestandaardiseerde wijze informatie uitwisselen. De code is in 1963 voor het eerst gepubliceerd [58] en in die tijd was er nog geen noodzaak om andere tekens te gebruiken dan de bekende westerse letters, cijfers en leestekens. Vandaar dat het aantal tekens beperkt is.

De ASCII-code bestaat uit 128 7-bits tekens zoals te zien is in tabel 2.5. De tekens zijn verdeeld in leesbare tekens, zoals letters, cijfers en leestekens en zogenoemde *besturingstekens*. De besturingstekens zijn nodig om informatie-overdracht af te bakenen en om een bepaalde *handshake* (uitwisselingsprotocol) te regelen. Zo zijn er codes voor de *carriage return* (CR, code 13_{10}) en de *backspace* (BS, code 8_{10}). De eerste 32 tekens zijn besturingstekens waarvan de meeste tegenwoordig niet meer gebruikt worden [59].

De codes zijn niet willekeurig toegekend wat we goed kunnen zien bij de cijfers en letters. De tabel is zo opgesteld dat de cijfers elkaar opvolgen. Dat is handig bij het afdrukken van een (decimaal) getal (denk hierbij aan de BCD-code). Hetzelfde geldt voor de letters, ook die volgen elkaar op. De makers hebben ook nagedacht over de positie van hoofd- en kleine letters. Deze verschillen in de tabel in slechts één bit (bit b_6 in de tabel). Bij het gebruik van de Caps Lock-toets of Shift-toets hoeft dus maar één bit (in combinatie met een letter) gewijzigd te worden. Een aantal besturingscodes is ook in programmeertalen zoals 'C' direct te gebruiken. In tabel 2.6 zijn de meest voorkomende besturingstekens weergegeven.

2.16 Andere codes voor karakters

De ASCII-code is voor een groot aantal talen niet toereikend. De letters, cijfers en leestekens worden vooral in de westerse talen gebruikt. Daarnaast zijn veel typografische

Tabel 2.5: *De ASCII-code [57].*

ASCII CONTROL CODE CHART

b7 b6 b5 / BITS b4 b3 b2 b1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		SYMBOLS NUMBERS		UPPER CASE		LOWER CASE	
0 0 0 0	0 NUL 0 0	16 DLE 10 20	32 SP 20 40	48 0 30 60	64 @ 40 100	80 P 50 120	96 ‘ 60 140	112 p 70 160
0 0 0 1	1 SOH 1 1	17 DC1 11 21	33 ! 21 41	49 1 31 61	65 A 41 101	81 Q 51 121	97 a 61 141	113 q 71 161
0 0 1 0	2 STX 2 2	18 DC2 12 22	34 ” 22 42	50 2 32 62	66 B 42 102	82 R 52 122	98 b 62 142	114 r 72 162
0 0 1 1	3 ETX 3 3	19 DC3 13 23	35 # 23 43	51 3 33 63	67 C 43 103	83 S 53 123	99 c 63 143	115 s 73 163
0 1 0 0	4 EOT 4 4	20 DC4 14 24	36 $ 24 44	52 4 34 64	68 D 44 104	84 T 54 124	100 d 64 144	116 t 74 164
0 1 0 1	5 ENQ 5 5	21 NAK 15 25	37 % 25 45	53 5 35 65	69 E 45 105	85 U 55 125	101 e 65 145	117 u 75 165
0 1 1 0	6 ACK 6 6	22 SYN 16 26	38 & 26 46	54 6 36 66	70 F 46 106	86 V 56 126	102 f 66 146	118 v 76 166
0 1 1 1	7 BEL 7 7	23 ETB 17 27	39 ’ 27 47	55 7 37 67	71 G 47 107	87 W 57 127	103 g 67 147	119 w 77 167
1 0 0 0	8 BS 8 10	24 CAN 18 30	40 (28 50	56 8 38 70	72 H 48 110	88 X 58 130	104 h 68 150	120 x 78 170
1 0 0 1	9 HT 9 11	25 EM 19 31	41) 29 51	57 9 39 71	73 I 49 111	89 Y 59 131	105 i 69 151	121 y 79 171
1 0 1 0	10 LF A 12	26 SUB 1A 32	42 * 2A 52	58 : 3A 72	74 J 4A 112	90 Z 5A 132	106 j 6A 152	122 z 7A 172
1 0 1 1	11 VT B 13	27 ESC 1B 33	43 + 2B 53	59 ; 3B 73	75 K 4B 113	91 [5B 133	107 k 6B 153	123 { 7B 173
1 1 0 0	12 FF C 14	28 FS 1C 34	44 , 2C 54	60 < 3C 74	76 L 4C 114	92 \ 5C 134	108 l 6C 154	124 | 7C 174
1 1 0 1	13 CR D 15	29 GS 1D 35	45 – 2D 55	61 = 3D 75	77 M 4D 115	93] 5D 135	109 m 6D 155	125 } 7D 175
1 1 1 0	14 SO E 16	30 RS 1E 36	46 . 2E 56	62 > 3E 76	78 N 4E 116	94 ^ 5E 136	110 n 6E 156	126 ~ 7E 176
1 1 1 1	15 SI F 17	31 US 1F 37	47 / 2F 57	63 ? 3F 77	79 O 4F 117	95 _ 5F 137	111 o 6F 157	127 DEL 7F 177

LEGEND: dec CHAR hex oct

Victor Eijkhout
TACC
Austin, Texas, USA

Tabel 2.6: *Enkele veelgebruikte besturingstekens*

Tekens	ASCII-naam	ASCII-code	'C'
Null	NUL	0	`\0`
Hoorbare Bel	BEL	7	`\a`
Horizontale tab	HT	9	`\t`
Line Feed	LF	10	`\n`
Carriage Return	CR	13	`\r`
Escape-toets	ESC	27	`\033`

tekens zoals het copyright-symbool niet beschikbaar. Een poging dit te ondervangen is de Unicode-standaard [60]. De standaard bevat meer dan 110.000 tekens uit meer dan 100 talen en tekensets. Voor het gebruik van Unicode in computerbestanden is UTF-8 (8-bits Unicode Transformation Format) ontwikkeld. UTF-8 is een tekencodering met variabele lengte. Niet elk teken gebruikt evenveel bytes. Afhankelijk van het teken worden 1 tot 4 bytes gebruikt. Voor het coderen van de 128 ASCII-tekens in UTF-8 is slechts één byte nodig. Naast UTF-8 bestaan er ook andere varianten zoals UTF-16 en UTF-32.

Een mooi voorbeeld van een oude binaire code is de Morse-code[8]. In deze code worden letters, cijfers en enkele leestekens uitgedrukt in punten ("dit") en strepen ("dah"). Een streep duurt drie keer zo lang als een punt. De Morse-code is vooral bedoeld voor transmissie (het oversturen van berichten). In de Morse-code is trouwens sprake van datacompressie: de E wordt gecodeerd met één punt, I met punt-punt en de A met punt-streep. De Y daarentegen met streep-punt-streep-streep. De E is de meest voorkomende letter in het Engels [61] en in het Nederlands [62].

*2.17 Getallen met willekeurig grondtal

Een willekeurig getal A in een positioneel talstelsel met grondtal R bestaande uit n cijfers voor de komma en k cijfers na de komma kan genoteerd worden als:

$$A_R = \{a_{n-1} \ldots a_0, a_{-1} \ldots a_{-k}\}_R \tag{2.16}$$

waarbij $a_{n-1} \ldots a_{-k}$ de afzonderlijke cijfers voorstellen. De (decimale) waarde van A_R kan dan uitgerekend worden door:

$$A_R = a_{n-1} \cdot R^{n-1} + \ldots + a_0 \cdot R^0 + a_{-1} \cdot R^{-1} + \ldots + a_{-k} \cdot R^{-k} \tag{2.17}$$

of, als sommatie van machten:s

$$A_R = \sum_{i=-k}^{n-1} a_i \cdot R^i \qquad \text{met } a_i \in \{0, \cdots, R-1\} \tag{2.18}$$

[8] Strikt genomen is dit niet waar. De stiltes tussen de karakters hebben ook betekenis. Dit wordt gebruikt om woordgrenzen aan te geven.

*2.18 Uitwerking bepaling aantal bits

Het aantal bits n dat minimaal nodig is om een decimaal getal M te representeren is:

$$M \leq 2^n - 1 \tag{2.19}$$

Om deze ongelijkheid uit te werken beginnen we met het omdraaien van de twee termen zodat n aan de linkerkant van de ongelijkheid staat:

$$2^n - 1 \geq M \tag{2.20}$$

Vervolgens tellen we links en rechts er 1 bij op:

$$2^n \geq M + 1 \tag{2.21}$$

Aan beide kanten nemen we nu de logaritme. Dat mag want de termen $M+1$ en 2^n zijn altijd groter dan 0. We nemen uiteraard de 2-log:

$${}^2\log 2^n \geq {}^2\log(M+1) \tag{2.22}$$

Nu kan n voor de logaritme gezet worden:

$$n \cdot {}^2\log 2 \geq {}^2\log(M+1) \tag{2.23}$$

Nu geldt dat ${}^2\log 2 = 1$ zodat volgt dat:

$$n \geq {}^2\log(M+1) \tag{2.24}$$

Voor het vinden van een minimum waarde van n wordt de ongelijkheid omgezet in een gelijkheid:

$$n = {}^2\log(M+1) \tag{2.25}$$

Uiteraard moet n, het minimum aantal bits dat nodig is om M weer te geven, een geheel getal zijn zodat er naar boven moet worden afgerond (tenzij het al een geheel getal is):

$$n = \lceil {}^2\log(M+1) \rceil \tag{2.26}$$

Op de meeste rekenmachines is de 2-log niet beschikbaar, dus we moeten de vergelijking omzetten naar een logaritme met een ander grondtal bijvoorbeeld 10-log. We maken gebruik van de gelijkheid:

$${}^a\log b = \frac{\log b}{\log a}. \tag{2.27}$$

zodat we uiteindelijk krijgen:

$$n = \left\lceil \frac{\log(M+1)}{\log 2} \right\rceil \tag{2.28}$$

Om een decimaal getal dat bestaat uit m cijfers met n bits te representeren moet voldaan worden aan de ongelijkheid:

$$10^m \leq 2^n \tag{2.29}$$

Het uitwerken met behulp van de 10-log gaat als volgt:

$$\begin{aligned} \log 2^n &\geq \log 10^m \\ n \cdot \log 2 &\geq m \\ n &\geq \frac{m}{\log 2} \end{aligned} \tag{2.30}$$

Voor het bepalen van het minimum aantal bits volgt dat:

$$n = \left\lceil \frac{m}{\log 2} \right\rceil \tag{2.31}$$

*2.19 Optimaal talstelsel

Mahler toont in [63] aan dat het binaire talstelsel efficiënt is voor wat betreft het ontwerpen van digitale schakelingen. Dat is als volgt te verklaren. We gaan uit van het representeren van het getal M. Om een getal M in een R-tallig talstelsel te representeren zijn N cijfers nodig met:

$$N = {}^R\log M \tag{2.32}$$

Gegeven een getal M, dan kunnen we stellen dat als R groot is N klein is en als R klein is N groot is. Als maat voor de complexiteit van een schakeling nemen we het aantal cijfers van het getal M vermenigvuldigd met het aantal mogelijke cijfers R. Dus:

$$f(R) = R \cdot N = R \cdot {}^R\log M \tag{2.33}$$

De waarde van R waarbij $R \cdot {}^R\log M$ minimaal is levert de optimale oplossing. Deze vinden we door de afgeleide van $f(R)$ gelijk te stellen aan 0:

$$\frac{\mathrm{d}f(R)}{\mathrm{d}R} = 0 \tag{2.34}$$

De afgeleide is:

$$\frac{\mathrm{d}f(R)}{\mathrm{d}R} = \frac{\ln N \cdot (\ln R - 1)}{(\ln R)^2} \tag{2.35}$$

Om de afgeleide op 0 te krijgen moet de teller 0 zijn en de noemer ongelijk aan 0. We kunnen dan schrijven dat:

$$\ln N \cdot (\ln R - 1) = 0 \qquad \wedge \qquad (\ln R)^2 \neq 0 \tag{2.36}$$

Hieruit volgt dat

$$\ln R - 1 = 0 \quad \rightarrow \quad \ln R = 1 \quad \rightarrow \quad R = \mathrm{e} \tag{2.37}$$

Het e-tallig talstelsel (e $\approx$ 2,71828) is dus het meest efficiënt. In de praktijk wordt voor een tweetallig talstelsel gekozen worden vanwege de eenvoud van de schakelingen.

2.20 Opgaven

2.1. Zet om van binair naar decimaal:

1111_2 110011_2 01111111_2 1010010110100101_2

2.2. Zet om van decimaal naar binair:

25_{10} 10_{10} 128_{10} 27543_{10}

2.3. Met tien vingers is het mogelijk om van 0 t/m 1023 te tellen door binaire codering te gebruiken. Een 0 is dan een gebogen vinger en een 1 is een gestrekte vinger. Waarom levert het getal 132_{10} toch problemen op in het dagelijks gebruik?

2.4. Zet om van hexadecimaal naar decimaal:

$3FF_{16}$ $4D52_{16}$ $CAFE_{16}$

2.5. Zet om van decimaal naar hexadecimaal:

255_{10} 57005_{10} 32768_{10}

2.6. Het nieuwe internet protocol IPv6 gebruikt 128 bits om computers een uniek IP-adres te geven. Hoeveel unieke adressen zijn mogelijk met IPv6? Voor meer informatie over IPv6 zie [64].

2.7. Hoeveel unieke IPv6-adressen zijn er mogelijk per vierkante meter aardoppervlakte?

2.8. Zet de volgende binaire breuken om in decimale breuken:

$0{,}1001_2$ $0{,}11111111_2$

2.9. Zet de volgende decimale breuken om in een binaire breuken:

$0{,}75_{10}$ $0{,}3_{10}$ $0{,}8_{10}$

2.10. Zet de volgende getallen om:

$11{,}7_{10} \rightarrow$ hex $1F{,}3C_{16} \rightarrow$ dec $3F{,}EA_{16} \rightarrow$ bin $110110{,}110111_2 \rightarrow$ hex

2.11. Hoewel zelden toegepast, komt in de praktijk ook het octale of achttallig talstelsel voor. Hierbij worden alleen de cijfers 0 t/m 7 gebruikt en elk cijfer wordt weergegeven met precies drie bits. Wat is de decimale waarde van 377_8 en 127_8? Wat is de binaire waarde van deze twee octale getallen?

2.12. In een bepaald talstelsel blijkt het getal 41 gelijk te zijn aan het kwadraat van 5. In welk talstelsel staan deze getallen geschreven?

2.13. Op de planeet Mars is een getal ontdekt in een onbekend talstelsel. Na veel puzzelen komen de onderzoekers er achter dat het marsiaanse getal 234 overeenkomt met aardse getal 123_{10}. In welk talstelsel zijn de marsiaanse getallen genoteerd?

2.14. Een ontwerper van een digitale schakeling heeft een getal opgeslagen in een 8 bits variabele. Bij het aanzetten van de schakeling wordt de variabele gereset (op 0 gezet). Wat is de waarde van deze variabele als hij 653 maal met 1 is verhoogd?

* **2.15.** Geef een algemene uitdrukking voor de verzameling getallen die opgeslagen in n bits het getal M oplevert met $0 \leq M \leq 2^n - 1$.

2.16. Hoeveel bits zijn er minimaal nodig om een 8-cijferig decimaal getal op te slaan?

2.17. Bepaal voor elk van de decimale getallen 365, 1024, 7987 en 176890 het aantal bits dat minimaal nodig is om de getallen binair weer te geven.

2.18. Hoeveel procent van de codecombinaties van een twee-cijferig BCD-getal kan nuttig worden gebruikt? En bij drie-cijferig?

* **2.19.** Ontwerp een schakeling die een (zuiver) 3-bits binaire code omzet in een 3-bits Gray-code. Hint: EXOR rules.

* **2.20.** Ontwerp een schakeling die een 3-bits Gray-code omzet in een (zuiver) 3-bits binaire code. EXOR rules again?

2.21. Laat zien dat het eenvoudig is om in de ASCII-code hoofdletters te veranderen in kleine letters (en andersom).

2.22. Met een 7-segment decoder kunnen de cijfers 0 t/m 9 worden gedecodeerd voor aansturing van een 7-segment display. De zes niet gebruikte combinaties kunnen toch zinvol worden gebruikt voor het afbeelden van A t/m F zodat ook hexadecimale cijfers kunnen worden afgebeeld. Hoe zouden die letters er op een 7-segment display uitzien?

3

Schakelalgebra

In hoofdstuk 1 hebben we kennis gemaakt met logische schakelingen. De werking van deze schakelingen kunnen we uitdrukken door middel van een waarheidstabel of een logische functie. Deze schakelingen zijn eenvoudig van aard. Naar mate de schakelingen complexer worden, hebben we behoefte aan een meer formelere en systematische manier om de schakelingen te beschrijven. We doen dat aan de hand van een wiskunde waarvan de fundamenten al zo'n 150 jaar geleden zijn gelegd.

Twee belangrijke publicaties hebben bijgedragen aan de ontwikkeling van de digitale techniek. De eerste, "An Investigation of the Laws of Thought" uit 1854 van George Boole [65] beschrijft hoe de bewerkingen op klassen en objecten wiskundig kunnen worden geformuleerd. Deze *Booleaanse algebra* is later uitgebreid en verbeterd door onder andere Edward Huntington [66]. De tweede publicatie is "A Symbolic Analysis of Relay and Switching Circuits" uit 1938 van Claude Shannon [67]. Shannon toonde aan dat de Booleaanse algebra toegepast kan worden bij het analyseren en ontwerpen van contactschakelingen met behulp van relais. Zo ontstond de *schakelalgebra* die de basis vormt van de digitale techniek. In Shannon's schakelalgebra kan een relais open of gesloten zijn en dat kan gerepresenteerd worden door een variabele die slechts twee waarden kan hebben, 0 of 1. Vanwege de sterke overeenkomsten worden de termen Booleaanse algebra en schakelalgebra als synoniemen gebruikt.

In dit hoofdstuk behandelen we de schakelalgebra. We beginnen met een korte introductie van de Booleaanse algebra. Daarna introduceren we de schakelalgebra op een wat formelere manier. Er worden elementaire rekenregels gegeven en een aantal wetten wordt besproken. Daarna worden diverse beschrijvings- en notatievormen van logische functies behandeld. We sluiten dit hoofdstuk af met het begrip don't care: situaties waarbij de waarde van een variabele niet van belang is.

3.1 Booleaanse algebra

De Booleaanse algebra verschilt in veel opzichten van de gewone algebra. Zo zijn er slechts twee constanten, 0 en 1, en kunnen variabelen (ook wel Booleaanse variabelen of logische variabelen genoemd) slechts een van de twee waarden aannemen.

Booleaanse algebra maakt gebruik van *proposities*. Een propositie is een uitspraak die op een bepaald moment waar of niet waar is. Waar en niet waar zijn de zogenoemde waarheidswaarden. Een propositie die waar is, heeft waarheidswaarde 1. Een propositie die niet waar is, heeft waarheidswaarde 0[1].

Een propositie:

"De zon schijnt"

is waar of niet waar. Duidelijk is trouwens wel dat de propositie niet echt handig geformuleerd is, de zon schijnt namelijk altijd. Een wat beter geformuleerde propositie is:

"De temperatuur in Delft is hoger dan 23 °C"

Naast proposities en de waarheidswaarden 0 en 1 zijn drie *operatoren* gedefinieerd:

- Conjunctie (en), weergegeven met $x \wedge y$, voldoet aan $x \wedge y = 1$ als $x = 1$ en $y = 1$, anders geldt $x \wedge y = 0$.
- Disjunctie (of), weergegeven met $x \vee y$, voldoet aan $x \vee y = 0$ als $x = 0$ en $y = 0$, anders geldt $x \vee y = 1$.
- Negatie (niet), weergegeven met $\neg x$, voldoet aan $\neg x = 0$ als $x = 1$ en $\neg x = 1$ als $x = 0$.

Door proposities met operatoren te combineren kunnen we nieuwe proposities samenstellen. Stel dat een kind graag een attractie in een amusementspark wil bezoeken. Daar gelden de volgende regels voor: een kind moet 12 jaar of ouder zijn, of een kind jonger dan 12 jaar moet begeleid worden door een volwassene. We stellen dan de volgende proposities op:

a: "De attractie mag bezocht worden"

l: "De leeftijd van het kind is groter dan of gelijk aan 12 jaar"

b: "Het kind wordt begeleid door een volwassen begeleider"

We kunnen de proposities opnemen in een *vergelijking*. Uit het bovenstaande kunnen concluderen dat:

$$a = l \vee (\neg l \wedge b) \tag{3.1}$$

We spreken de vergelijking uit als "a is waar als l is waar **of** l is **niet** waar **en** b is waar, anders is a niet waar". Merk trouwens op dat we bij het uitspreken wel goed de volgorde

[1] Er zijn boeken waarin de waarheidswaarde T (true) en F (false) worden genoemd. Dat is onjuist: T is een propositie die altijd waar is, F is een propositie die altijd niet waar is. Zie [68] voor meer uitleg.

van de bewerkingen moeten aangeven: **en** heeft dus voorrang boven **of**. We kunnen deze proposities in een zogenoemde waarheidstabel plaatsen, zie tabel 3.1. De tabel stellen we op door alle combinaties van de waarheidswaarden van l en b onder elkaar op te schrijven en te koppelen aan de bijbehorende waarheidswaarde van a.

Tabel 3.1: *Tabel voor vergelijking* (3.1).

l	b	a
0	0	0
0	1	1
1	0	1
1	1	1

De regel met $l = 0$ en $b = 0$ leest als "De leeftijd van het kind is **niet** groter dan of gelijk aan 12 jaar **en** het kind wordt **niet** begeleid door een volwassen begeleider". Daar hangt mee samen dat "De attractie **niet** bezocht mag worden".

Bekijken we tabel 3.1 eens goed dan valt op dat propositie a waar is als propositie l waar is of propositie b waar is of als beide proposities waar zijn. Dit is te noteren als:

$$a = l \vee b \vee (l \wedge b) \tag{3.2}$$

Verder impliceert de constructie $l \vee b$ dat de propositie ook waar is als l en b beide waar zijn. De term $l \wedge b$ is dus overbodig. De gehele vergelijking kan nu in de meest eenvoudige vorm geschreven worden als:

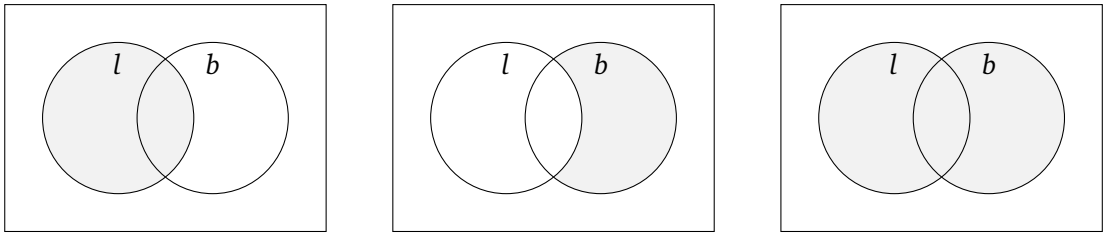
$$a = l \vee b \tag{3.3}$$

Hieruit blijkt dat er meerdere beschrijvingen zijn die *logisch equivalent* zijn. De vergelijking in (3.3) is de meest eenvoudige. Er is een minimum aan bewerkingen nodig.

Het voorstellen van en het uitbeelden van bewerkingen op proposities kan inzichtelijk worden gemaakt met *Venndiagrammen*. Een Venndiagram bestaat uit een rechthoek en een aantal cirkels. De rechthoek wordt het *universum* genoemd en omvat alle mogelijkheden. Voor elke propositie wordt één cirkel getekend. Het gebied binnen de cirkel staat voor het waar zijn van een propositie. Het gebied buiten een cirkel staat voor het niet waar zijn van een propositie.

De Venndiagrammen in figuur 3.1 laten de proposities l en b zien. De cirkel voor l staat voor het waar zijn van propositie l. De cirkel voor b staat voor het waar zijn van propositie b. De cirkels overlappen elkaar. Dit gebied staat voor het waar zijn van beide proposities.

In figuur 3.1(a) is het gebied $l = 1$ uitgebeeld. Dit houdt in dat een kind 12 jaar of ouder is. Figuur 3.1(b) beeld het het gebied uit waarvoor geldt dat een kind niet 12 jaar of ouder is én dat het kind begeleid wordt door een volwassene, met andere woorden $l = 0 \wedge b = 1$. Figuur 3.1(c) beeldt de vereniging uit van beide proposities. Hiervoor geldt dat een kind 12 jaar of ouder is óf dat het kind wordt begeleid door een een volwassene. Deze figuur komt overeen met vergelijking (3.3).



(a) $l = 1$. **(b)** $\neg l \wedge b = 1$. **(c)** $l \wedge b = 1$.

Figuur 3.1: *Het uitbeelden van bewerkingen op proposities l en b.*

3.2 Schakelalgebra

De schakelalgebra is naast variabelen en de constanten 0 en 1 gebaseerd op drie *operatoren*: AND, OR en NOT. Variabelen worden Booleaanse of logische variabelen genoemd. Hetzelfde geldt voor functies, formules en vergelijkingen.

In tabel 3.2 zijn de functies van AND, OR en NOT weergegeven.

Tabel 3.2: *De drie basisfuncties van de schakelalgebra.*

(a) *AND-functie.*

0	0	0
0	1	0
1	0	0
1	1	1

(b) *OR-functie.*

0	0	0
0	1	1
1	0	1
1	1	1

(c) *NOT-functie.*

0	1
1	0

In de schakelalgebra wordt de AND vervangen door de halfhoge punt "$\cdot$" (niet te verwarren met de rekenkundige vermenigvuldiging) en de OR vervangen door de plus "+" (niet te verwarren met de rekenkundige optelling). De NOT wordt vervangen door een negatiestreep $\overline{\ }$ (Engels: overbar) boven de propositie. Het isgelijkteken wordt gebruikt om de twee delen in een functie of formule aan elkaar gelijk te stellen. Een voorbeeld van een schakelfunctie is:

$$s = x + \overline{y} \cdot z \tag{3.4}$$

In de bovenstaande functie wordt $\overline{y} \cdot z$ het *logisch product* van $\overline{y}$ en z genoemd vanwege de gelijkenis met het rekenkundig product. Een AND-term wordt een *productterm* genoemd. De $x + \ldots$ wordt de *logische som* genoemd vanwege de gelijkenis met de rekenkundige optelling. Een OR-term wordt een *somterm* genoemd.

Volgorde van bewerkingen

In de schakelalgebra wordt de onderstaande *prioriteitvolgorde* gebruikt:

- NOT
- AND
- OR

Haakjes kunnen gebruikt worden om de volgorde van bewerking aan te passen. Stel dat we de volgende functie moeten uitwerken:

$$s = a \cdot (b + \overline{c}) \tag{3.5}$$

Dan is de volgorde van bewerken:

- eerst $\overline{c}$ uitwerken;
- dan b met de uitkomst van $\overline{c}$ uitwerken: $(b + \overline{c})$;
- dan a met de uitkomst van $(b + \overline{c})$ uitwerken: $a \cdot (b + \overline{c})$.

Gegeven dat $a = 1$, $b = 0$ en $c = 0$ dan is de uitkomst:

$$\overline{c} = 1$$
$$b + \overline{c} = 0 + 1 = 1$$
$$a \cdot (b + \overline{c}) = 1 \cdot 1 = 1$$

Het is mogelijk om de negatiestreep over een term te plaatsen. Dan moet eerst de term onder de negatiestreep berekend worden en daarna de inverse bepaald worden. Het is dan niet nodig om haakjes te gebruiken:

$$\overline{(a + b)} \quad \rightarrow \quad \overline{a + b} \tag{3.6}$$

3.3 Rekenregels voor logische variabelen

In deze paragraaf introduceren we de rekenregels voor logische variabelen. Met behulp van deze regels is het mogelijk om schakelfuncties te bewerken. Deze *wetten* zijn eenvoudig te bewijzen m.b.v. de definities van de drie basisbewerkingen die gegeven zijn in tabel 3.2. Zo is de wet $a \cdot a = a$ te bewijzen door aan beide kanten van het isgelijkteken a te vervangen door 0 respectievelijk 1 en te kijken of de vergelijking klopt:

$$\begin{aligned} a = 0 \quad &\xrightarrow{a=0 \text{ invullen}} \quad 0 \cdot 0 = 0 \\ a = 1 \quad &\xrightarrow{a=1 \text{ invullen}} \quad 1 \cdot 1 = 1 \end{aligned} \tag{3.7}$$

Wetten voor één variabele

Deze wetten laten zien hoe functies of schakelingen met één variabele moeten worden geïnterpreteerd.

$$a \cdot a = a \tag{3.8a}$$
$$a + a = a \tag{3.8b}$$

$$a \cdot \overline{a} = 0 \tag{3.9a}$$
$$a + \overline{a} = 1 \tag{3.9b}$$

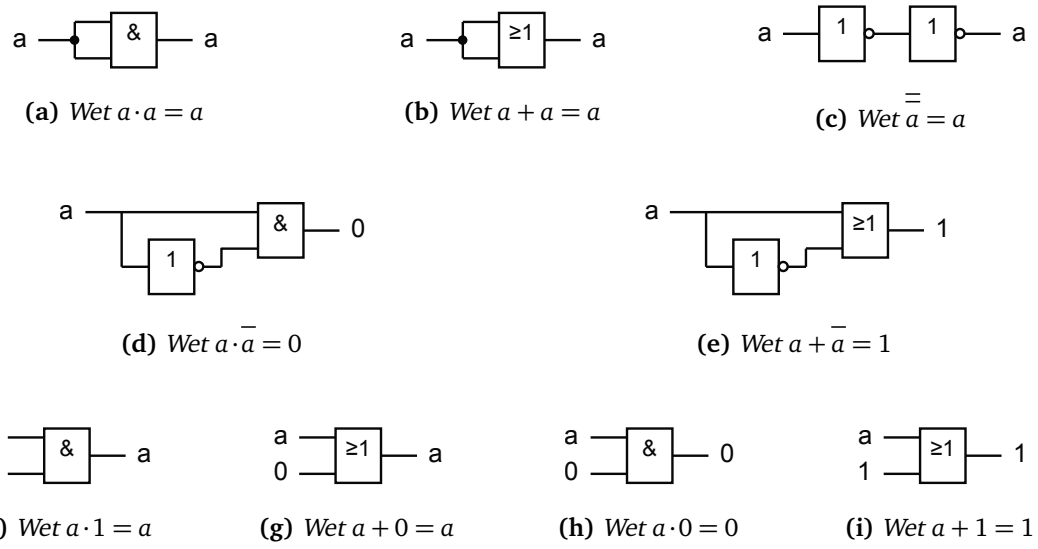
$$\overline{\overline{a}} = a \tag{3.10}$$

$$a \cdot 1 = a \tag{3.11a}$$
$$a + 0 = a \tag{3.11b}$$

$$a \cdot 0 = 0 \tag{3.12a}$$
$$a + 1 = 1 \tag{3.12b}$$

De wetten in (3.8) worden gelijkheidswetten genoemd. Zij geven aan dat in een logische som of logisch product identieke variabelen mogen worden geschrapt. De wetten in (3.9) worden negatiewetten genoemd. Ze geven aan hoe functies met negaties moeten worden geïnterpreteerd. De wet in (3.10) laat zien dat dubbele negatie van een variabele leidt tot de variabele zelf. De wetten in (3.11) en (3.12) worden moduluswetten genoemd. Zij geven aan hoe functies met constanten moeten worden geïnterpreteerd.

We kunnen de wetten demonstreren aan de hand van poortschakelingen. In figuur 3.2 zijn de wetten (3.8) tot en met (3.12) uitgebeeld.



(a) *Wet* $a \cdot a = a$ **(b)** *Wet* $a + a = a$ **(c)** *Wet* $\overline{\overline{a}} = a$

(d) *Wet* $a \cdot \overline{a} = 0$ **(e)** *Wet* $a + \overline{a} = 1$

(f) *Wet* $a \cdot 1 = a$ **(g)** *Wet* $a + 0 = a$ **(h)** *Wet* $a \cdot 0 = 0$ **(i)** *Wet* $a + 1 = 1$

Figuur 3.2: *Schema's wetten voor één variabele.*

Uit de de figuur blijkt dat in alle gevallen deze wetten leiden tot vereenvoudigde schakelingen; de poorten kunnen worden vervangen door de variabele zelf of de logische constanten 0 of 1.

Associatieve wetten

De associatieve wetten laten zien dat de volgorde van dezelfde logische bewerkingen gewijzigd mogen worden.

$$(a \cdot b) \cdot c = a \cdot (b \cdot c) \tag{3.13a}$$
$$(a + b) + c = a + (b + c) \tag{3.13b}$$

Commutatieve wetten

Volgens de commutatieve wetten mogen de variabelen in een bewerking verwisseld worden.

$$a \cdot b = b \cdot a \tag{3.14a}$$
$$a + b = b + a \tag{3.14b}$$

Distributieve wetten

De distributieve wetten geven aan hoe haakjes binnen of buiten een deel van een schakelfunctie kunnen worden gebruikt.

$$(a + b) \cdot (a + c) = a + b \cdot c \tag{3.15a}$$
$$a \cdot b + a \cdot c = a \cdot (b + c) \tag{3.15b}$$

Absorptiewetten

De absorptiewetten zijn vereenvoudigingswetten. Ze geven aan wanneer een variabele in een functie overbodig is.

$$a \cdot b + a \cdot \overline{b} = a \tag{3.16a}$$
$$(a + b) \cdot (a + \overline{b}) = a \tag{3.16b}$$

$$a + a \cdot b = a \tag{3.17a}$$
$$a \cdot (a + b) = a \tag{3.17b}$$

$$a + \overline{a} \cdot b = a + b \tag{3.18a}$$
$$a \cdot (\overline{a} + b) = a \cdot b \tag{3.18b}$$

De onderstaande wetten worden ook wel de *consensustheorema's* genoemd. Ze spelen een belangrijke rol bij het voorkomen van *hazards* (zie hoofdstuk 4).

$$a \cdot b + \overline{a} \cdot c + b \cdot c = a \cdot b + \overline{a} \cdot c \tag{3.19a}$$
$$(a + b) \cdot (\overline{a} + c) \cdot (b + c) = (a + b) \cdot (\overline{a} + c) \tag{3.19b}$$

Stellingen van De Morgan

De Morgan was een tijdgenoot van Boole. Deze stellingen laten zien dat AND en OR kunnen worden uitgewisseld als de variabelen worden geïnverteerd.

$$c = a \cdot b \quad \longleftrightarrow \quad \overline{c} = \overline{a} + \overline{b} \tag{3.20a}$$
$$c = a + b \quad \longleftrightarrow \quad \overline{c} = \overline{a} \cdot \overline{b} \tag{3.20b}$$

Meestal worden ze als volgt geschreven:

$$\overline{a \cdot b} = \overline{a} + \overline{b} \tag{3.21a}$$
$$\overline{a + b} = \overline{a} \cdot \overline{b} \tag{3.21b}$$

De Morgan speelt een belangrijke rol bij het realiseren van schakelingen met NAND- en NOR-poorten.

Uitvermenigvuldigen

Op basis van herhaald toepassen van wet (3.15b) kunnen we het volgende opschrijven:

$$\begin{aligned}(a+b)\cdot(c+d) &= a\cdot(c+d)+b\cdot(c+d)\\ &= a\cdot c+a\cdot d+b\cdot c+b\cdot d\end{aligned} \tag{3.22}$$

Dit wordt uitvermenigvuldigen genoemd.

Voorbeeld schakelalgebra

Het is mogelijk om met behulp van schakelalgebra wetten te bewijzen. We laten er twee zien. Eerst de distributieve wet (3.15a):

$$\begin{aligned}(a+b)\cdot(a+c) &= a\cdot a+a\cdot c+b\cdot a+b\cdot c && \text{via (3.22)}\\ &= a\cdot a+a\cdot b+a\cdot c+b\cdot c && \text{via (3.14)}\\ &= a+a\cdot b+a\cdot c+b\cdot c && \text{via (3.8a)}\\ &= a\cdot 1+a\cdot b+a\cdot c+b\cdot c && \text{via (3.11a)}\\ &= a\cdot(1+b+c)+b\cdot c && \text{via (3.15b)}\\ &= a\cdot 1+b\cdot c && \text{via (3.12b)}\\ &= a+b\cdot c && \text{via (3.11a)}\end{aligned} \tag{3.23}$$

Met behulp van het bovenstaande bewijs kan de absorptiewet (3.18a) worden bewezen:

$$\begin{aligned}a+\overline{a}\cdot b &= (a+\overline{a})\cdot(a+b) && \text{via (3.15a)}\\ &= 1\cdot(a+b) && \text{via (3.9b)}\\ &= a+b && \text{via (3.11a)}\end{aligned} \tag{3.24}$$

Het bewerken van logische functies kost enige moeite. Sommige stappen kunnen overgeslagen worden. Grondige kennis van wetten en regels is noodzakelijk.

3.4 Dualiteit

De wetten in (3.8) tot en met (3.21) (met uitzondering van wet (3.10)) zijn in paren opgeschreven. De b-variant van een wet is te vinden door in de a-variant $\cdot$ met $+$ te verwisselen en, indien aanwezig, 0 met 1. Haakjes moeten worden gebruikt om de oorspronkelijke volgorde van bewerkingen te handhaven. We kunnen nu een *metawet*, een wet over wetten, geven: elke vergelijking in de schakelalgebra blijft waar als $\cdot$ met $+$ en 0 met 1 verwisseld worden. Dit wordt het *dualiteitsprincipe* genoemd. Dualiteit is een belangrijke eigenschap in de schakelalgebra. Het maakt het mogelijk om AND- en OR-schakelingen uit te wisselen. Als we eenmaal geleerd hebben om een AND-OR-schakeling te realiseren vanuit de som-van-producten-vorm, dan kunnen we ook een OR-AND-schakeling realiseren vanuit de product-van-sommen-vorm. Zie paragraaf 3.9. We nemen als voorbeeld de functie uit (3.4) en zijn duale vorm:

$$s = x+\overline{y}\cdot z \qquad s^D = x\cdot(\overline{y}+z) \tag{3.25}$$

Hierin is s^D de duale vorm van s. De haakjes zorgen ervoor dat de originele bewerkingsvolgorde gehandhaafd blijft. Merk op dat de gewone vorm en de duale vorm elkaars inverse *kunnen* zijn. Een voorbeeld is:

$$s = \overline{x} \cdot y + x \cdot \overline{y} \qquad s^D = (\overline{x} + y) \cdot (x + \overline{y}) \tag{3.26}$$

3.5 Waarheidstabellen

De meest basale weergave van een logische functie is de waarheidstabel. In tabel 3.3 is de algemene opbouw van een waarheidstabel gegeven, in dit geval voor drie variabelen. In de waarheidstabel komt elke ingangscombinatie exact één keer voor en de tabel bevat alle mogelijke ingangscombinaties. De rijen zijn volgens de normale binaire telcode oplopend genummerd.

Tabel 3.3: *Algemene opzet waarheidstabel voor drie variabelen.*

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

De variabelen f_0 t/m f_7 zijn de bijbehorende *functiewaarden*. De subscript volgt uit de binaire waarde van de bijbehorende ingangscombinatie.

De constructie van een waarheidstabel is eenvoudig. Voor een tabel met n variabelen zijn 2^n regels nodig om alle mogelijke ingangscombinaties te noteren. Schrijf bovenaan de tabel eerst de variabelenamen op. Begin daarna van achter af (de laatst opgeschreven variabelenaam) naar beneden te nummeren afwisselend 0 en 1. De variabelenaam daarvoor moet om de twee regels afwisselend genummerd worden met 0 en 1, daarvoor om de vier etc. Waarheidstabellen zijn trouwens praktisch te gebruiken tot vijf variabelen, daarna zijn de tabellen niet echt hanteerbaar. Bedenk dat een tabel van tien variabelen 1024 regels heeft.

TO BE OR NOT TO BE...

"To be or not to be, that is the question" is de bekende uitspraak uit het stuk Hamlet van William Shakespeare. Het antwoord op de vraag levert 1 op, want

$$2B + \overline{2B} = 1$$

De schrijver was zijn tijd ver vooruit...

Waarheidstabellen kunnen worden gebruikt bij het bewijzen van de eerder genoemde wetten. In feite kan een waarheidstabel gebruikt worden om aan te tonen of twee schakelfuncties logisch equivalent zijn door alle mogelijke combinaties van de ingangswaarden uit te rekenen. Als voorbeeld nemen we de stellingen van De Morgan. Zie tabel 3.4.

Tabel 3.4: *Bewijs van de stellingen van De Morgan met behulp van waarheidstabellen.*

a	b	$\overline{a+b}$	$\overline{a}\cdot\overline{b}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

a	b	$\overline{a\cdot b}$	$\overline{a}+\overline{b}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Een functie met twee variabelen heeft vier functiewaarden. Deze functiewaarden kunnen allemaal 0 of 1 zijn zodat er in totaal zestien verschillende functies kunnen worden gerealiseerd. In tabel 3.5 staan deze opgesomd.

Tabel 3.5: *Alle mogelijke functies met twee variabelen.*

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Sommige van deze functies zijn triviaal, zoals de functies F_0 (alle functiewaarden 0), F_1 (x AND y), F_3 (x), F_7 (x OR y), F_{12} (NOT x) en F_{15} (alle functiewaarden zijn 1). Merk op dat alle functies met twee variabelen te realiseren zijn met (een combinatie van) AND, OR en NOT, d.w.z. dat de verzameling van operatoren *functioneel compleet* is [69]. Er zijn geen andere operatoren nodig. Dit geldt ook voor functies met een willekeurig aantal variabelen.

3.6 De mintermvorm

Elke regel in tabel 3.3 levert een bijdrage aan de functie. De eerste regel leest als volgt:

Als $x = 0$ en $y = 0$ en $z = 0$, dan geldt $s = f_0$

of

$s = f_0$ als $\overline{x}\cdot\overline{y}\cdot\overline{z} = 1$

De bijdrage van deze regel aan de functie s is:

$s = \overline{x}\cdot\overline{y}\cdot\overline{z}\cdot f_0 + \ldots$

De productterm zorgt ervoor dat s gelijk is aan f_0 als $x = 0$, $y = 0$ en $z = 0$. De overige zeven regels worden op vergelijkbare wijze gelezen en leveren op vergelijkbare wijze een

bijdrage aan de functie van s:

$$
\begin{aligned}
s &= f_1 \quad \text{als } \overline{x}\cdot\overline{y}\cdot z = 1\\
s &= f_2 \quad \text{als } \overline{x}\cdot y\cdot\overline{z} = 1\\
s &= f_3 \quad \text{als } \overline{x}\cdot y\cdot z = 1\\
s &= f_4 \quad \text{als } x\cdot\overline{y}\cdot\overline{z} = 1\\
s &= f_5 \quad \text{als } x\cdot\overline{y}\cdot z = 1\\
s &= f_6 \quad \text{als } x\cdot y\cdot\overline{z} = 1\\
s &= f_7 \quad \text{als } x\cdot y\cdot z = 1
\end{aligned}
$$

Voor een gegeven combinatie van x, y en z wordt de waarde van s dus bepaald door bijbehorende functiewaarde. We kunnen de functie nu schrijven als een logische som van deze producttermen:

$$
\begin{aligned}
s = \overline{x}\cdot\overline{y}\cdot\overline{z}\cdot f_0 + \overline{x}\cdot\overline{y}\cdot z\cdot f_1 + \overline{x}\cdot y\cdot\overline{z}\cdot f_2 + \overline{x}\cdot y\cdot z\cdot f_3 +\\
x\cdot\overline{y}\cdot\overline{z}\cdot f_4 + x\cdot\overline{y}\cdot z\cdot f_5 + x\cdot y\cdot\overline{z}\cdot f_6 + x\cdot y\cdot z\cdot f_7
\end{aligned}
\tag{3.27}
$$

De termen $\overline{x}\cdot\overline{y}\cdot\overline{z}$ tot en met $x\cdot y\cdot z$ worden *mintermen* genoemd. Mintermen zijn producttermen waarin elke variabele slechts één keer voorkomt, gewoon of geïnverteerd. Merk op dat voor een gegeven combinatie van x, y en z slechts één minterm logisch 1 is. De mintermen sluiten elkaar wederzijds uit.

Mintermen worden doorgaans afgekort door een m met een subscript. Zo is de minterm die hoort bij $\overline{x}\cdot\overline{y}\cdot\overline{z}$ gelijk aan m_0 en $x\cdot y\cdot z$ gelijk aan m_7. De functie kan dus ook geschreven worden als:

$$
s = m_0\cdot f_0 + m_1\cdot f_1 + m_2\cdot f_2 + m_3\cdot f_3 + m_4\cdot f_4 + m_5\cdot f_5 + m_6\cdot f_6 + m_7\cdot f_7 \tag{3.28}
$$

De vorm van deze functie wordt de *som van mintermen* genoemd. Dit is een van de twee standaardvormen (ook wel canonieke vormen genoemd).

3.7 De maxtermvorm

Er bestaat nog een tweede vorm waarin schakelfuncties kunnen wordt uitgedrukt: in een logisch product van sommen. De functie is:

$$
\begin{aligned}
s = (x+y+z+f_0)\cdot(x+y+\overline{z}+f_1)\cdot(x+\overline{y}+z+f_2)\cdot(x+\overline{y}+\overline{z}+f_3)\cdot\\
(\overline{x}+y+z+f_4)\cdot(\overline{x}+y+\overline{z}+f_5)\cdot(\overline{x}+\overline{y}+z+f_6)\cdot(\overline{x}+\overline{y}+\overline{z}+f_7)
\end{aligned}
\tag{3.29}
$$

De termen $x+y+z$ tot en met $\overline{x}+\overline{y}+\overline{z}$ worden *maxtermen* genoemd. Maxtermen zijn somtermen waarin elke variabele slechts één keer voorkomt, gewoon of geïnverteerd. Merk op dat voor een gegeven combinatie van x, y en z slechts één maxterm logisch 0 is, de andere maxtermen zijn logisch 1. De maxtermen sluiten elkaar wederzijds uit.

Maxtermen worden doorgaans afgekort door een M met een subscript. Zo is de maxterm die hoort bij $x+y+z$ gelijk aan M_0 en $\overline{x}+\overline{y}+\overline{z}$ gelijk aan M_7. De functie kan dus ook geschreven worden als:

$$\begin{aligned} s = {} & (M_0+f_0)\cdot(M_1+f_1)\cdot(M_2+f_2)\cdot(M_3+f_3)\cdot \\ & (M_4+f_4)\cdot(M_5+f_5)\cdot(M_5+f_5)\cdot(M_7+f_7) \end{aligned} \tag{3.30}$$

De vorm van deze functie wordt de *product van maxtermen* genoemd. Dit is een van de twee standaardvormen.

3.8 Verband tussen mintermvorm en maxtermvorm

Het verband tussen de mintermvorm en maxtermvorm laat zich het beste zien door een voorbeeld. Gegeven de waarheidstabel in tabel 3.6 voor de functie s van drie variabelen x, y en z:

Tabel 3.6: *Waarheidstabel voor drie variabelen.*

x	y	z	s
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

De logische functie weergegeven in tabel 3.6 wordt als volgt in de mintermvorm geschreven:

$$\begin{aligned} s = {} & \overline{x}\cdot\overline{y}\cdot\overline{z}\cdot 0+\overline{x}\cdot\overline{y}\cdot z\cdot 1+\overline{x}\cdot y\cdot\overline{z}\cdot 0+\overline{x}\cdot y\cdot z\cdot 1+ \\ & x\cdot\overline{y}\cdot\overline{z}\cdot 0+x\cdot\overline{y}\cdot z\cdot 1+x\cdot y\cdot\overline{z}\cdot 1+x\cdot y\cdot z\cdot 1 \end{aligned} \tag{3.31}$$

of door gebruik te maken van de mintermnotatie als:

$$s = m_0\cdot 0+m_1\cdot 1+m_2\cdot 0+m_3\cdot 1+m_4\cdot 0+m_5\cdot 1+m_6\cdot 1+m_7\cdot 1 \tag{3.32}$$

Meestal worden de producttermen met een 0 volledig geschrapt en producttermen met een 1 worden gereduceerd tot alleen de bijbehorende minterm. De functie wordt dan geschreven als:

$$s = \overline{x}\cdot\overline{y}\cdot z+\overline{x}\cdot y\cdot z+x\cdot\overline{y}\cdot z+x\cdot y\cdot\overline{z}+x\cdot y\cdot z \tag{3.33}$$

of door gebruik te maken van de mintermnotatie als:

$$s = m_1+m_3+m_5+m_6+m_7 \tag{3.34}$$

Verder bestaat er de zogenoemde verkorte of somnotatie waarbij de subscripts van de mintermen die een logische 1 opleveren genoteerd worden:

$$s = \sum m(1,3,5,6,7) \tag{3.35}$$

Dezelfde functie kan ook in de maxtermvorm geschreven worden:

$$\begin{aligned} s = {} & (x+y+z+0)\cdot(x+y+\overline{z}+1)\cdot(x+\overline{y}+z+0)\cdot(x+\overline{y}+\overline{z}+1)\cdot \\ & (\overline{x}+y+z+0)\cdot(\overline{x}+y+\overline{z}+1)\cdot(\overline{x}+\overline{y}+z+1)\cdot(\overline{x}+\overline{y}+\overline{z}+1) \end{aligned} \tag{3.36}$$

of door gebruik te maken van de maxtermnotatie als:

$$\begin{aligned} s = {} & (M_0+0)\cdot(M_1+1)\cdot(M_2+0)\cdot(M_3+1)\cdot \\ & (M_4+0)\cdot(M_5+1)\cdot(M_5+1)\cdot(M_7+1) \end{aligned} \tag{3.37}$$

In deze functie worden alle somtermen met een 1 volledig geschrapt en somtermen met een 0 gereduceerd tot de bijbehorende maxterm. De functie wordt dan geschreven als:

$$s = (x+y+z)\cdot(x+\overline{y}+z)\cdot(\overline{x}+y+z) \tag{3.38}$$

of door gebruik te maken van de verkorte of maxtermnotatie als:

$$s = M_0 \cdot M_2 \cdot M_4 \tag{3.39}$$

Verder bestaat er de zogenoemde productnotatie waarbij de subscripts van de maxtermen die een logische 0 opleveren genoteerd worden:

$$s = \prod M(0,2,4) \tag{3.40}$$

De functie s wordt beschreven door zowel (3.35) als (3.40) zodat geldt dat:

$$s = \sum m(1,3,5,6,7) = \prod M(0,2,4) \tag{3.41}$$

Vergelijken we de functiewaarden in tabel 3.6 en de functie in (3.41), dan zien we direct dat de mintermvorm overeenkomt met alle enen in de tabel en de maxtermvorm met alle nullen.

Laten we de mintermen en maxtermen voor een functie van drie variabelen naast elkaar opsommen. Dat is te zien in tabel 3.7.

Uit de tabel blijkt direct dat mintermen en maxtermen elkaars inverse zijn. Dat kan aangetoond worden met behulp van De Morgen. Zo is:

$$m_0 = \overline{x}\cdot\overline{y}\cdot\overline{z} \tag{3.42}$$

We inverteren de minterm en passen de stellingen van De Morgan toe:

$$\overline{m_0} = \overline{\overline{x}\cdot\overline{y}\cdot\overline{z}} = x+y+z = M_0 \tag{3.43}$$

Tabel 3.7: *Waarheidstabel met mintermen en maxtermen.*

x	y	z	minterm	maxterm
0	0	0	$\overline{x}\cdot\overline{y}\cdot\overline{z}$	$x+y+z$
0	0	1	$\overline{x}\cdot\overline{y}\cdot z$	$x+y+\overline{z}$
0	1	0	$\overline{x}\cdot y\cdot\overline{z}$	$x+\overline{y}+z$
0	1	1	$\overline{x}\cdot y\cdot z$	$x+\overline{y}+\overline{z}$
1	0	0	$x\cdot\overline{y}\cdot\overline{z}$	$\overline{x}+y+z$
1	0	1	$x\cdot\overline{y}\cdot z$	$\overline{x}+y+\overline{z}$
1	1	0	$x\cdot y\cdot\overline{z}$	$\overline{x}+\overline{y}+z$
1	1	1	$x\cdot y\cdot z$	$\overline{x}+\overline{y}+\overline{z}$

3.9 SOP- en POS-vormen

Elke schakelfunctie wordt volledig gespecificeerd door de mintermvorm. In de regel kan zo'n functie geschreven worden in een vorm waarbij meerdere mintermen gecombineerd worden tot productermen waarin niet elke variabele (gewoon of geïnverteerd) voorkomt. We spreken dan van de *som-van-producten-vorm* (SOP-vorm). Laten we vergelijking (3.33) nog eens nader bekijken:

$$s = \overline{x}\cdot\overline{y}\cdot z + \overline{x}\cdot y\cdot z + x\cdot\overline{y}\cdot z + x\cdot y\cdot\overline{z} + x\cdot y\cdot z \tag{3.44}$$

Te zien is dat er vier mintermen zijn waarin de variabele z voorkomt en één minterm waarin de inverse van z voorkomt. Als we z buiten haakjes halen wordt de functie:

$$s = (\overline{x}\cdot\overline{y} + \overline{x}\cdot y + x\cdot\overline{y} + x\cdot y)\cdot z + x\cdot y\cdot\overline{z} \tag{3.45}$$

Binnen de haakjes komen alle mogelijke combinaties met x en y voor. In feite staan hier alle mintermen van een functie met twee variabelen genoteerd. Dit deel van de functie levert de constante 1 op en kan verwijderd worden uit de functie:

$$\begin{aligned} s &= (\underbrace{\overline{x}\cdot\overline{y} + \overline{x}\cdot y + x\cdot\overline{y} + x\cdot y}_{1})\cdot z + x\cdot y\cdot\overline{z} \\ &= z + x\cdot y\cdot\overline{z} \end{aligned} \tag{3.46}$$

Met behulp van absorptieregel (3.18a) uit de schakelalgebra kan $\overline{z}$ verwijderd worden zodat de functie uiteindelijk wordt:

$$s = z + x\cdot y \tag{3.47}$$

De functies uit (3.44) en (3.47) zijn *logisch equivalent*: voor elke mogelijke ingangscombinatie leveren de functies dezelfde uitgangswaarden op. Het mag duidelijk zijn dat de korte vorm met veel minder operaties de juiste waarden oplevert. Een schakeling voor deze functie is met veel minder poorten te realiseren.

Het is over het algemeen mogelijk om een functie die in een maxtermvorm geschreven is, met minder en kleinere somtermen te schrijven door maxtermen te combineren. We

spreken dan van een product-van-sommen-vorm (POS-vorm). We nemen als voorbeeld de functie:

$$s = (x + y + z) \cdot (x + \overline{y} + z) \cdot (\overline{x} + y + z) \tag{3.48}$$

De functie kan als volgt gereduceerd worden:

$$\begin{aligned} s &= (x + y + z) \cdot (x + \overline{y} + z) \cdot (\overline{x} + y + z) \\ &= (x + y + z) \cdot (x + y + z) \cdot (x + \overline{y} + z) \cdot (\overline{x} + y + z) \\ &= (x + y + z) \cdot (x + \overline{y} + z) \cdot (\overline{x} + y + z) \cdot (x + y + z) \\ &= (x + (y \cdot \overline{y}) + z) \cdot ((x \cdot \overline{x}) + y + z) \\ &= (x + z) \cdot (y + z) \end{aligned} \tag{3.49}$$

Overigens zijn niet alle functies compacter te schrijven. Een voorbeeld is de functie:

$$s = \overline{x} \cdot \overline{y} \cdot z + \overline{x} \cdot y \cdot \overline{z} + x \cdot \overline{y} \cdot \overline{z} + x \cdot y \cdot z \tag{3.50}$$

Geen enkele minterm is met een andere minterm te combineren.

3.10 Functies met don't cares

Tot nu toe hebben we gezien dat elke ingangscombinatie een bijbehorende uitgangswaarde 0 of 1 oplevert. Dat is niet altijd nodig. Soms komt in de praktijk een bepaalde ingangscombinatie nooit voor of is de uitgangswaarde bij een bepaalde ingangscombinatie niet belangrijk. We spreken dan van een *don't care*. In een waarheidstabel wordt dit aangegeven met een halfhoog streepje (—). Er zijn ook andere notaties in omloop zoals x, X, d, D of Φ.

Als voorbeeld is de waarheidstabel in tabel 3.8 gegeven. In de tabel zijn twee don't cares te zien.

Tabel 3.8: *Waarheidstabel met don't cares.*

x	y	z	s
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	—
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	—

De functie wordt in de somnotatie genoteerd als:

$$s = \sum m(2) + d(3,7) \tag{3.51}$$

en in de productnotatie als:

$$s = \prod M(0,1,4,5,6) \cdot D(3,7) \tag{3.52}$$

Merk op dat de don't cares bij beide notaties voorkomen, hierbij geven d en D aan of het om product- of somtermen gaat. Verder valt op te merken dat don't cares alleen kunnen worden gespecificeerd bij de verkorte notaties.

Don't cares bestaan alleen bij de specificatie van een functie, niet bij de realisatie van een functie. Bij realisatie worden ze omgezet in een 0 of 1. Er zijn twee don't cares die elk afzonderlijk als 0 of 1 kunnen worden ingevuld, dus er zijn vier functies te realiseren:

$$\begin{aligned}
&s = \overline{x} \cdot y \cdot \overline{z} && m_3 = 0,\ m_7 = 0 \\
&s = \overline{x} \cdot y && m_3 = 1,\ m_7 = 0 \\
&s = \overline{x} \cdot y \cdot \overline{z} + x \cdot y \cdot z && m_3 = 0,\ m_7 = 1 \\
&s = \overline{x} \cdot y + y \cdot z && m_3 = 1,\ m_7 = 1
\end{aligned} \tag{3.53}$$

Alle functies voldoen aan de specificatie.

*3.11 Nog meer booleaans

Een ander mooi voorbeeld van het gebruik van Booleaanse algebra is het vinden van informatie bij zoekmachines. Meestal is één zoekterm niet voldoende om het aanbod van resultaten binnen redelijke grenzen te houden. Met gebruik van de termen AND, OR en NOT zijn meerdere zoektermen te combineren. Willen we zoeken naar ontmoetingen tussen Einstein en Bohr, dan moeten we zoeken met de term "meeting AND Einstein AND Bohr". De drie zoektermen leveren een verzameling resultaten op en de *doorsnede* van deze verzamelingen levert het gewenste resultaat.

We kunnen ook zoeken op twee bekenden uit de kwantummechanica, Bohr en Planck. Hierbij levert de zoekterm "Bohr OR Planck" de *vereniging* van de verzameling resultaten van beide zoektermen. Merk op dat deze verzameling ook resultaten bevat waarin Bohr én Plank voorkomen.

Stel nu dat we alle ontmoetingen tussen zowel Einstein en Planck als Einstein en Bohr willen opzoeken. Dat kan met de zoekterm "(meeting AND Einstein AND Bohr) OR (meeting AND Einstein AND Planck)" waarbij de *haakjes* prioriteitsvolgorde afdwingen. Dat levert twee deelverzamelingen op (twee maal AND) die weer verenigd worden (OR). Deze zoekterm is *logisch equivalent* met "meeting AND Einstein AND (Bohr OR Planck)" (let op de haakjes). Ook hier kan weer worden opgemerkt dat in de zoekresultaten ontmoetingen zijn opgenomen waar alle heren aanwezig zijn geweest.

3.12 Opgaven

3.1. Gegeven dat $a = 1$, $b = 0$ en $c = 0$. Werk de volgende functies uit. De waarde van d is niet gegeven.

$s = (a + b) \cdot (\overline{b} + c)$ $\quad$ $s = \overline{(\overline{a + b}) \cdot (\overline{c + d})}$ $\quad$ $s = \overline{b} \cdot (c + d \cdot (c + a))$

3.2. Toon aan met behulp van de schakelalgebra dat de absorptiewet $a + a \cdot b = a$ klopt.

3.3. Toon aan met behulp van de schakelalgebra en met behulp van waarheidstabellen dat de consensuswet $(a+b)\cdot(\overline{a}+c)\cdot(b+c) = (a+b)\cdot(\overline{a}+c)$ klopt.

3.4. Gegeven de functie: $s = x \cdot (y + z \cdot \overline{y}) + z \cdot y$
Bepaal de mintermvorm van deze functie. Bepaal de waarheidstabel van deze functie.

3.5. Schrijf de functie $s_{x,y,z} = \sum m(1,3,4,7)$ uit als een som van mintermen.

3.6. Schrijf de functie $s_{x,y,z} = \prod M(2,6,7)$ uit als een product van maxtermen.

3.7. Schrijf de functie $s_{x,y,z} = \sum m(1,2,3,6)$ uit als een product van maxtermen.

3.8. Schrijf de functie $s_{x,y,z} = \prod M(0,1,3,5)$ uit als een som van mintermen.

3.9. Schrijf elk van de functies, gedefinieerd in tabel P3.1, als een som van mintermen (indien mogelijk) en in de somnotatie.

Tabel P3.1: *Waarheidstabel bij opgave 3.9 en 3.10.*

x	y	z	S_1	S_2	S_3	S_4	S_5	S_6	S_7
0	0	0	0	1	0	0	0	1	1
0	0	1	1	1	0	1	1	0	–
0	1	0	0	1	0	1	1	1	1
0	1	1	1	0	0	1	1	0	–
1	0	0	0	1	0	0	1	1	1
1	0	1	1	0	0	1	1	0	0
1	1	0	1	0	0	1	1	1	–
1	1	1	1	1	1	1	1	0	0

3.10. Schrijf elk van de functies, gedefinieerd in tabel P3.1, als een product van maxtermen (indien mogelijk) en in de productnotatie.

3.11. Gegeven een functie van drie variabelen waarvoor geldt dat de functie voor $xyz = 000$ en $xyz = 111$ don't care is en dat de functie een logische 1 geeft als $x = 0$ terwijl $z = 1$, anders is de functie logisch 0. Geef de waarheidstabel.

4

Combinatorische schakelingen

In de vorige hoofdstukken hebben we de logische poorten en schakelalgebra behandeld. We hebben nu genoeg gereedschap om *combinatorische schakelingen* te ontwikkelen. De uitgangswaarden van combinatorische schakelingen zijn alleen afhankelijk van de huidige ingangswaarden en niet van een voorafgaande reeks ingangswaarden. Combinatorische schakelingen hebben geen geheugenwerking.

In dit hoofdstuk gaan we de schakelalgebra en poorten gebruiken om schakelingen te analyseren en synthetiseren. We introduceren diverse methodes om schakelfuncties te *minimaliseren* zodat schakelingen met een minimum aan poorten en signalen gerealiseerd kunnen worden.

Realiseren van de geminimaliseerde functie kan met de bekende poorten AND, OR en NOT maar het is ook mogelijk om een schakeling met alleen NAND- of NOR-poorten te bouwen. We laten zien dat elke functie kan worden gerealiseerd met multiplexers en ROMs.

Om de theorie te ondersteunen werken we een aantal voorbeelden uit. We beginnen met een geschreven specificatie en werken dit uit tot een schakeling met poorten. Daarbij wordt een aantal stappen doorlopen: opstellen van een waarheidstabel, minimaliseren van de logische functies, eventueel functies omwerken naar NAND- of NOR-vorm en tekenen van de schakeling met poorten.

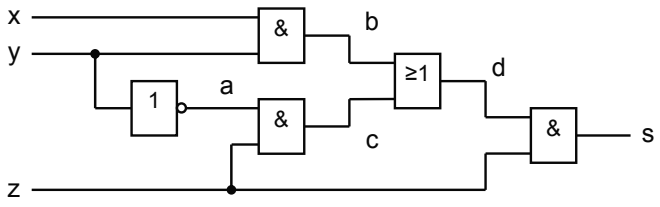
Ook de elektrische eigenschappen moeten niet vergeten worden. We laten een aantal ingangs- en uitgangsschakelingen zien waarmee een koppeling met de fysieke buitenwereld wordt gemaakt. Tot slot bespreken we de timing van combinatorische schakelingen en in het bijzonder *timing hazards*.

4.1 Analyse van combinatorische schakelingen

Met *analyse* bedoelen we het bepalen van de schakelfunctie van een schakeling met logische poorten. Het doel is te achterhalen hoe de schakeling reageert op de ingangs-

combinaties. Naderhand kunnen we de resultaten gebruiken om een uitspraak te doen over de uitgangswaarde bij een bepaalde ingangscombinatie of om de schakeling op een andere manier te realiseren, bijvoorbeeld met minder poorten. Het bepalen van de functie kan op twee manieren: met behulp van de schakelalgebra en door middel van waarheidstabellen.

In figuur 4.1 is een schakeling gegeven met de ingangen x, y en z en uitgang s. De schakeling bestaat uit vijf poorten. Verder zijn in de schakeling de interne signalen a, b, c en d opgenomen. Deze zijn niet strikt noodzakelijk maar vergemakkelijken het analyseproces.



Figuur 4.1: *Schakeling met poorten.*

De functies van de poorten zijn bekend. Van elk van de interne signalen en de uitgang kunnen we de functie bepalen uitgedrukt in de ingangssignalen en de interne signalen. Zo is a de inverse van y en d de logische som van b en c. In vergelijking (4.1a) t/m (4.1e) zijn alle functies gegeven:

$$a = \overline{y} \tag{4.1a}$$
$$b = x \cdot y \tag{4.1b}$$
$$c = a \cdot z \tag{4.1c}$$
$$d = b + c \tag{4.1d}$$
$$s = d \cdot z \tag{4.1e}$$

Vervolgens werken we de functie uit door de interne variabelen te vervangen door hun functie totdat de functie s volledig is beschreven door de ingangen (algebraïsche substitutie):

$$\begin{aligned} s &= d \cdot z \\ &= (b + c) \cdot z \\ &= (x \cdot y + a \cdot z) \cdot z \\ &= (x \cdot y + \overline{y} \cdot z) \cdot z \end{aligned} \tag{4.2}$$

We hebben nu een, wat later zal blijken, niet-vereenvoudigde schakelfunctie verkregen. Het vinden van een uitgangswaarde voor een specifieke ingangscombinatie kost toch nog enige moeite. Voor $xyz = 101$ vinden we:

$$s = (1 \cdot 0 + \overline{0} \cdot 1) \cdot 1 = (0 + 1 \cdot 1) \cdot 1 = (0 + 1) \cdot 1 = 1 \tag{4.3}$$

We kunnen een schakelfunctie ook bepalen door middel van het opstellen van een waarheidstabel. In de waarheidstabel nemen we alle signalen op. Links plaatsen we de

ingangen x, y en z, gevolgd door de interne signalen a, b, c en d. Geheel rechts plaatsen we uitgang s. De waarden van de interne signalen zijn makkelijk te bepalen, de functies van de poorten zijn eenvoudig van aard. Uiteindelijk krijgen we de waarden zoals te zien is in tabel 4.1.

Tabel 4.1: *Waarheidtabel voor functie s.*

x	y	z	a	b	c	d	s
0	0	0	1	0	0	0	0
0	0	1	1	0	1	1	1
0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	1	0	1	1	1
1	1	0	0	1	0	1	0
1	1	1	0	1	0	1	1

Met behulp van de tabel is het makkelijk om een uitgangswaarde te bepalen bij een gegeven ingangscombinatie. Stel dat $xyz = 101$ is, dan vinden we in de zesde rij dat de uitgang logisch 1 is.

Vanuit de waarheidstabel is direct de mintermvorm van de functie op te stellen:

$$s = \overline{x} \cdot \overline{y} \cdot z + x \cdot \overline{y} \cdot z + x \cdot y \cdot z = m_1 + m_5 + m_7 = \sum m(1,5,7) \tag{4.4}$$

Het gebruik van waarheidstabellen bij analyse beperkt zich tot maximaal vijf variabelen. Bij meer variabelen worden de tabellen vanwege de grootte niet meer hanteerbaar en moet voor de schakelalgebra gekozen worden.

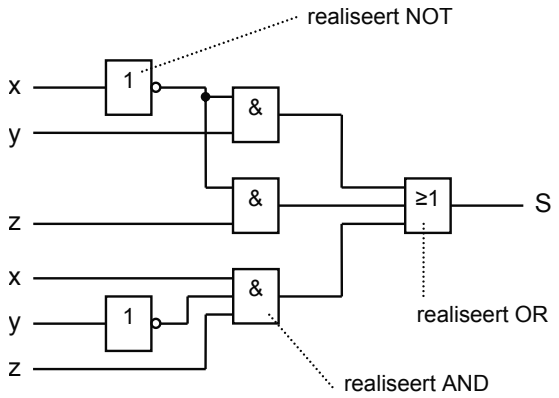
4.2 Synthese van combinatorische schakelingen

Met *synthese* bedoelen we het *samenstellen* van een schakeling die is opgebouwd uit kleinere delen. Al eerder hebben we gezien dat elke schakelfunctie te beschrijven is met de operatoren AND, OR en NOT. Bij het realiseren van een schakeling kunnen we dus gebruik maken van de drie bijbehorende poorten.

Als voorbeeld nemen we de functie in de som-van-producten-vorm (SOP-vorm):

$$S = \overline{x} \cdot y + \overline{x} \cdot z + x \cdot \overline{y} \cdot z \tag{4.5}$$

Het realiseren van een schakeling gaat als volgt. Als eerste moeten de inversen van x en y gerealiseerd worden. Hiervoor worden NOT-poorten gebruikt. Daarna worden de producttermen gerealiseerd door middel van AND-poorten. De sommatie van de producttermen wordt gerealiseerd door middel van een OR-poort. Het resultaat is te zien in figuur 4.2. Merk op dat voor de inverse van x slechts één NOT-poort nodig is. Deze schakeling heeft zes poorten en twaalf ingangen.

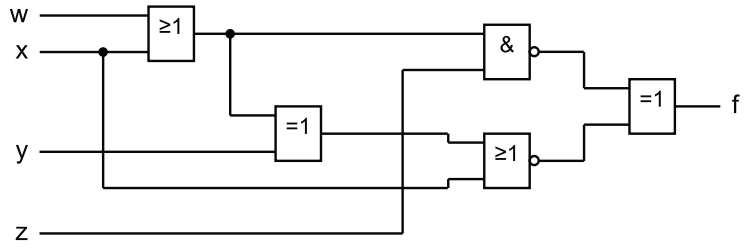


Figuur 4.2: *Synthese van een schakeling met poorten.*

In de praktijk komen nog wel andere poorten voor zoals de NAND-, NOR- en EXOR-poorten. Als voorbeeld nemen we de functie:

$$f = \overline{(w+x)\cdot z} \oplus \overline{((w+x)\oplus y)+x} \tag{4.6}$$

De schakeling is te zien in figuur 4.3.



Figuur 4.3: *Synthese van een schakeling met poorten.*

Deze schakeling kost vijf poorten en tien ingangen. Nu rijst de vraag: kan het niet eenvoudiger?

4.3 Minimalisatie

Uit het oogpunt van kosten, oppervlakte en energieverbruik (wat ook kosten met zich meebrengt) is het wenselijk om een schakelfunctie te *minimaliseren*. Er wordt gezocht naar de eenvoudigste vorm van een functie. Het is niet eenvoudig om aan te geven wat de eenvoudigste vorm is. We kunnen zoeken naar een vorm die het minste aantal operaties (poorten) en variabelen (ingangen) bevat. Hoe minder poorten, hoe minder oppervlak op een ic. Een ingang betekent bedrading (wires) op een ic en waar bedrading is, kan geen logische poort worden geplaatst. Het is dus van belang om het aantal poorten en ingangen te verkleinen. Maar het kan ook zijn dat een vorm wordt gezocht die het eenvoudigst *afbeeldbaar* is op poorten of transistoren op ic's. Denk hierbij aan NAND- en NOR-poorten. Deze zijn makkelijker en goedkoper (lees: minder transistoren) te realiseren dan AND- en OR-poorten. Al met al zijn er meerdere manieren om te minimaliseren. Aangezien we steeds spreken over het logische poorten hanteren we de definitie dat de functie in de meest eenvoudige som-van-producten-vorm (SOP-vorm) is geschreven met:

- een zo klein mogelijk aantal producttermen;
- per productterm een zo klein mogelijk aantal variabelen.

Bij minimalisatie wordt vooral gebruik gemaakt van de wetten (3.16a) en (3.8b). Hieronder zijn ze nog een keer gegeven:

$$a \cdot b + a \cdot \overline{b} = a \tag{4.7a}$$

$$a + a = a \tag{4.7b}$$

De wet in (4.7a) laat zien dat de logische som van twee producttermen die slechts in één variabele verschillen, d.w.z. de variabele komt bij de ene productterm voor in de gewone vorm en bij de andere productterm in geïnverteerde vorm, kunnen worden samengenomen. De variabele die verschilt in de twee producttermen verdwijnt. Zo zijn de producttermen $x \cdot y \cdot z$ en $x \cdot y \cdot \overline{z}$ te combineren tot de productterm $x \cdot y$. De wet in (4.7b) zegt dat we in het vereenvoudigingsproces een productterm meer dan één keer mogen gebruiken. We lezen de wet in dit geval van rechts naar links. De productterm $x \cdot y \cdot \overline{z}$ mag dus geschreven worden als $x \cdot y \cdot \overline{z} + x \cdot y \cdot \overline{z}$.

We zullen achtereenvolgens minimalisatie van functies laten zien aan de hand van schakelalgebra, een grafische methode en een tabellarische methode. De laatste twee methoden leveren direct vereenvoudigde functies in SOP- en POS-vorm. We sluiten af met korte verhandeling over andere minimalisatiemethoden.

4.3.1 Schakelalgebra

Als voorbeeld nemen we het schema uit figuur 4.1. De functie die direct uit de schakeling volgt is:

$$s = (x \cdot y + \overline{y} \cdot z) \cdot z \tag{4.8}$$

Met behulp van de schakelalgebra kunnen we de functie bewerken. We proberen een zo eenvoudig mogelijke uitdrukking in som-van-productenvorm (SOP-vorm) te vinden, een functie met een minimum aan operatoren en variabelen.

$$s = (x \cdot y + \overline{y} \cdot z) \cdot z \qquad \text{de functie} \tag{4.9a}$$

$$= x \cdot y \cdot z + \overline{y} \cdot z \cdot z \qquad \text{haakjes wegwerken} \tag{4.9b}$$

$$= x \cdot y \cdot z + \overline{y} \cdot z \qquad z \cdot z = z \tag{4.9c}$$

$$= (x \cdot y + \overline{y}) \cdot z \qquad \text{buiten haakjes halen} \tag{4.9d}$$

$$= (x + \overline{y}) \cdot z \qquad \text{absorptiewet} \tag{4.9e}$$

$$= x \cdot z + \overline{y} \cdot z \qquad \text{minimale SOP-vorm} \tag{4.9f}$$

Als we functies (4.8) en (4.9f) vergelijken dan zien we dat de eerste vorm meer variabelen en operatoren bevat dan de tweede vorm. De oplettende lezer ziet trouwens dat functie (4.9e) nóg minder variabelen en operatoren bevat in vergelijking met (4.9f). Dit is echter een product-van-sommen-vorm (POS-vorm) en is niet gevraagd.

Het is echter niet altijd duidelijk of de meest eenvoudige vorm van een functie is gevonden. Laten we eens de volgende functie onderzoeken waarvan zowel de waarheidstabel als de functie in mintermvorm is gegeven. Deze zijn te vinden in figuur 4.4.

x	y	z	s
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$\begin{aligned} s &= \sum m(0,1,2,5,6,7) \\ &= m_0 + m_1 + m_2 + m_5 + m_6 + m_7 \\ &= \overline{x}\cdot\overline{y}\cdot\overline{z} + \overline{x}\cdot\overline{y}\cdot z + \overline{x}\cdot y\cdot\overline{z} + x\cdot\overline{y}\cdot z + x\cdot y\cdot\overline{z} + x\cdot y\cdot z \end{aligned} \tag{4.10}$$

Figuur 4.4: *Waarheidstabel en functie.*

We minimaliseren de functie door het samennemen van m_0 met m_1, m_0 met m_2, m_5 met m_7 en m_6 met m_7. Dat levert:

$$\begin{aligned} s &= \overline{x}\cdot\overline{y}\cdot(\overline{z}+z) + \overline{x}\cdot(\overline{y}+y)\cdot\overline{z} + x\cdot(\overline{y}+y)\cdot z + x\cdot y\cdot(\overline{z}+z) \\ &= \overline{x}\cdot\overline{y} + \overline{x}\cdot\overline{z} + x\cdot z + x\cdot y \end{aligned} \tag{4.11}$$

De functie in (4.11) heeft vier producttermen en acht variabelen. We noemen deze functie *niet-reduceerbaar* omdat de functie niet verder te vereenvoudigen is door toepassing van schakelalgebra zonder terug te gaan naar de oorspronkelijke vorm.

Nu combineren we de mintermen m_0 met m_1, m_2 met m_6 en m_5 met m_7. Dit levert:

$$\begin{aligned} s &= \overline{x}\cdot\overline{y}\cdot(\overline{z}+z) + (\overline{x}+x)\cdot y\cdot\overline{z} + x\cdot(\overline{y}+y)\cdot z \\ &= \overline{x}\cdot\overline{y} + y\cdot\overline{z} + x\cdot z \end{aligned} \tag{4.12}$$

De functie in (4.12) heeft drie producttermen en zes variabelen en is niet verder te vereenvoudigen. De vorm is wel eenvoudiger dan de functie in (4.11). We weten alleen nog steeds niet of de functie de *minimale representatie* van (4.10) is (dat is het wel, zie verderop).

Soms is het nodig om een functie te expanderen naar meer mintermen om de functie te vereenvoudigen. Neem als voorbeeld de functie:

$$s = a\cdot b\cdot\overline{c} + \overline{a}\cdot b\cdot c + a\cdot b\cdot c \tag{4.13}$$

Door productterm $a\cdot b\cdot c$ twee keer te gebruiken krijgen we:

$$\begin{aligned} s &= a\cdot b\cdot\overline{c} + \overline{a}\cdot b\cdot c + a\cdot b\cdot c \\ &= a\cdot b\cdot\overline{c} + \overline{a}\cdot b\cdot c + a\cdot b\cdot c + a\cdot b\cdot c \\ &= a\cdot b\cdot\overline{c} + a\cdot b\cdot c + \overline{a}\cdot b\cdot c + a\cdot b\cdot c \\ &= a\cdot b\cdot(\overline{c}+c) + b\cdot c\cdot(\overline{a}+a) \\ &= a\cdot b + b\cdot c \end{aligned} \tag{4.14}$$

Het mag duidelijk zijn dat minimalisatie met behulp van algebra erg lastig is en dat veel ervaring vereist is. De beste manier is om de functie geheel uit te schrijven in mintermen en dan de wetten uit (4.7) te gebruiken.

4.3.2 Karnaughdiagrammen

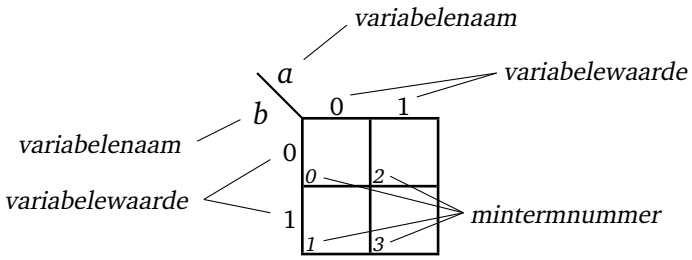
Al eerder hebben we gezien dat twee mintermen te combineren zijn tot een productterm met minder variabelen als de mintermen in slechts één variabele verschillen. We noemen deze mintermen *logisch aangrenzend* (Engels: logic adjacent). In een waarheidstabel is dit niet altijd goed te zien. Als we de mintermen van een functie van twee variabelen als volgt in een vierkant rangschikken:

$$\begin{array}{cc} \overline{a}\cdot\overline{b} & a\cdot\overline{b} \\ \overline{a}\cdot b & a\cdot b \end{array}$$

dan staan alle aangrenzende mintermen naast elkaar, zowel in horizontale als verticale richting: $\overline{a}\cdot\overline{b}$ is logisch aangrenzend met $a\cdot\overline{b}$ en met $\overline{a}\cdot b$. We kunnen deze vorm gebruiken bij het minimaliseren van functies.

Een Karnaughdiagram [70] is een visueel hulpmiddel voor het vereenvoudigen van schakelfuncties. Het is vooral geschikt voor mensen omdat die goed in staat zijn patronen te herkennen. Het is gebaseerd op de wetten in (4.7) en het idee van het bovenstaande vierkant.

Het diagram bestaat uit een aantal vierkanten of hokjes (Engels: cells), voor elke functiewaarde één. In figuur 4.5 is de opbouw van een Karnaughdiagram voor twee variabelen te zien[1]. Elk hokje vertegenwoordigt één minterm van de variabelen a en b. De variabelenamen worden boven en onder de diagonale lijn bij de linkerbovenhoek geschreven. De waarden van variabelen worden boven en naast de hokjes geschreven. De waarden aan de bovenkant behoren tot variabele a, die aan de linkerkant tot variabele b. In de hokjes zijn de bijbehorende mintermnummers geschreven.

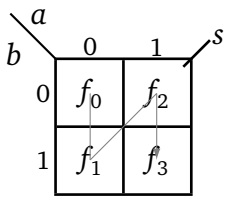


Figuur 4.5: *Opbouw van een Karnaughdiagram voor twee variabelen.*

Om de relaties tussen mintermen en functiewaarden aan te geven, plaatsen we de functiewaarden in een Karnaughdiagram. In figuur 4.6 is links het Karnaughdiagram gegeven en rechts de bijbehorende waarheidstabel. Het is goed te zien dat mintermen die logisch

[1] Er zijn veel varianten mogelijk van een Karnaughdiagram. Dit boek gebruikt de variant die in Engelstalige literatuur gebruikt wordt.

aangrenzend zijn horizontaal of verticaal gerangschikt zijn. Het invullen van de functiewaarden begint links, van boven naar beneden, dan schuin omhoog naar rechts en dan weer van boven naar beneden. Dit is te zien aan de grijze gebroken pijl in de figuur.



a	b	s
0	0	f_0
0	1	f_1
1	0	f_2
1	1	f_3

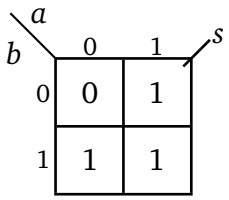
Figuur 4.6: *Karnaughdiagram voor twee variabelen.*

Het vinden van de SOP-vorm van een functie

Laten we eens de functie:

$$s = \overline{a} \cdot b + a \cdot \overline{b} + a \cdot b \tag{4.15}$$

invullen in een Karnaughdiagram. Dit is weergegeven in figuur 4.7 met links het ingevulde Karnaughdiagram en rechts de waarheidstabel. Uit de waarheidstabel blijkt dat het om een eenvoudige OR-functie gaat.



a	b	s
0	0	0
0	1	1
1	0	1
1	1	1

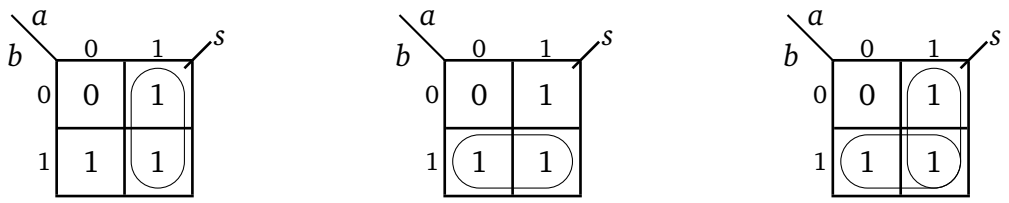
Figuur 4.7: *Karnaughdiagram en waarheidstabel voor twee variabelen.*

In de figuur is goed te zien dat de enen van functiewaarden f_2 en f_3 aangrenzend boven elkaar geplaatst zijn. Dat betekent dat de mintermen te combineren zijn tot een productterm met minder variabelen. In figuur 4.8(a) is dit weergegeven met de omranding van de twee functiewaarden. Duidelijk is te zien dat de enen liggen in het gedeelte waar $a = 1$. Dus als $a = 1$ dan is $f = 1$ ongeacht of b gelijk aan 0 of 1 is. De functiewaarden f_1 en f_3 staan ook aangrenzend, maar nu horizontaal en zijn dus ook te combineren. Dit is weergegeven in figuur 4.8(b), het gedeelte waar $b = 1$. De combinatie van de twee figuren levert uiteindelijk de omranding op van alle enen uit de functie. Dit is te zien in figuur 4.8(c). Opvallend is dat functiewaarde f_3 bij beide omrandingen gebruikt wordt (dit mag op basis van de wet (4.7b)) en dat de omrandingen elkaar overlappen.

De omrandingen geven aan dat de functie te schrijven is als:

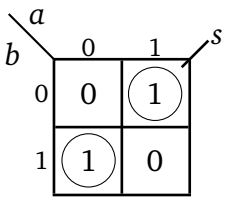
$$s = a + b \tag{4.16}$$

Het is niet altijd mogelijk om een functie te vereenvoudigen. Bekijk de waarheidstabel in figuur 4.9 eens. In het Karnaughdiagram zijn de enen diagonaal geplaatst en zijn niet te combineren.



(a) *Omranding $a = 1$.* **(b)** *Omranding $b = 1$.* **(c)** *Gecombineerd.*

Figuur 4.8: *Karnaughdiagrammen voor twee variabelen.*



a	b	s
0	0	0
0	1	1
1	0	1
1	1	0

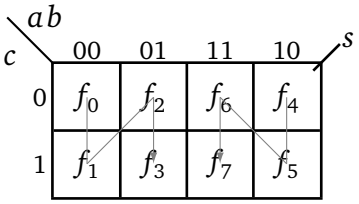
Figuur 4.9: *Karnaughdiagram en waarheidstabel voor twee variabelen.*

De functie is:

$$s = \overline{a} \cdot b + a \cdot \overline{b} \tag{4.17}$$

We herkennen hier de EXOR-functie in.

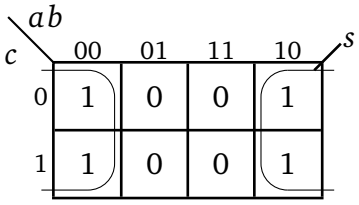
In figuur 4.10 is het Karnaughdiagram getekend voor een functie van drie variabelen. Aan de bovenrand zijn de variabelen a en b geplaatst met de bijbehorende variabelewaarden, genoteerd volgens de *Gray-codering* (zie paragraaf 2.13). Dit is noodzakelijk omdat de aangrenzende mintermen in slechts één variabele mogen verschillen. Zo is minterm f_2 ($abc = 010$) te combineren met minterm f_6 ($abc = 110$). Links zijn de variabele c en de bijbehorende variabelewaarden te zien. Het invullen van de functiewaarden f_0 t/m f_3 gaat op dezelfde wijze als bij een Karnaughdiagram voor twee variabelen. De functiewaarden f_4 t/m f_7 worden als het ware gespiegeld t.o.v. de eerste vier functiewaarden ingevuld zoals is aangegeven met de grijze pijlen.



a	b	c	s
0	0	0	f_0
0	0	1	f_1
0	1	0	f_2
0	1	1	f_3
1	0	0	f_4
1	0	1	f_5
1	1	0	f_6
1	1	1	f_7

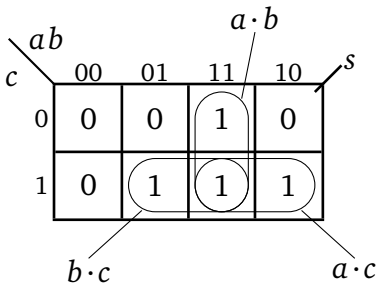
Figuur 4.10: *Opbouw van het Karnaughdiagram voor drie variabelen.*

Merk op dat de linkerrand (f_0, f_1) "vastzit" aan de rechterrand (f_4, f_5). Dit is het gedeelte waar $b = 0$. Zie figuur 4.11.



Figuur 4.11: *De linker- en rechterrand van het Karnaughdiagram zitten aan elkaar vast.*

Een uitgewerkt voorbeeld is te zien in figuur 4.12. Om alle enen van deze functie af te dekken zijn drie omrandingen nodig. De twee enen in de kolom $ab = 11$ vormen de productterm $a \cdot b$. De enen van f_5 en f_7 liggen in het gedeelte waar $ab = 11$ en $ab = 10$ wat resulteert in $a = 1$. Samen met $c = 1$ vormt dit de productterm $a \cdot c$. De enen van f_3 en f_7 vormen samen de productterm $b \cdot c$. Dat is het gedeelte waar $ab = 01$ en $ab = 11$ en $c = 1$.



a	b	c	s
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

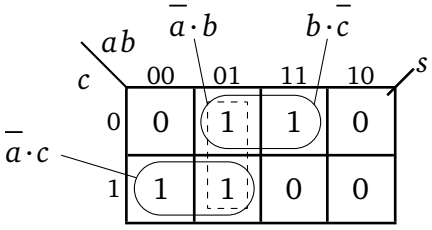
Figuur 4.12: *Karnaughdiagram en waarheidstabel voor drie variabelen.*

De volledige functie, geschreven in de minimale SOP-vorm, is dus:

$$s = a \cdot b + a \cdot c + b \cdot c \tag{4.18}$$

Deze functie is niet verder te vereenvoudigen. Merk op dat minterm $a \cdot b \cdot c$ drie keer wordt omrand.

Bij het uitwerken van een Karnaughdiagram mogen de omrandingen elkaar overlappen, maar alleen als daardoor de gevonden functie minimaal is. Het is dus niet altijd vanzelfsprekend dat omrandingen elkaar overlappen. In figuur 4.13 is een Karnaughdiagram getekend.



Figuur 4.13: *Karnaugdiagram.*

Er zijn slechts twee omrandingen nodig om alle enen van de functie af te dekken. De omranding van de gestippelde rechthoek (productterm $\bar{a} \cdot b$) is overbodig omdat de enen

al door de andere omrandingen zijn afgedekt. De functie is:

$$s = \overline{a} \cdot c + b \cdot \overline{c} \tag{4.19}$$

Soms leidt het vereenvoudigingsproces tot meerdere minimale functies. De Karnaughdiagrammen in figuur 4.14 kunnen op twee manieren uitgewerkt worden. Beide leveren een minimale functie. De functies zijn logisch equivalent en bevatten net zoveel variabelen en bewerkingen dus beide functies zijn een minimale POS-vorm van dezelfde logische functie.

(a) *Karnaugdiagram.* **(b)** *Karnaugdiagram.*

Figuur 4.14: *Karnaughdiagram voor drie variabelen.*

Het Karnaughdiagram in figuur 4.14(a) levert de functie:

$$s = \overline{a} \cdot \overline{b} + a \cdot c + b \cdot \overline{c} \tag{4.20}$$

Die van figuur 4.14(b) levert (merk op dat de zijranden aan elkaar vastzitten):

$$s = a \cdot b + \overline{a} \cdot \overline{c} + \overline{b} \cdot c \tag{4.21}$$

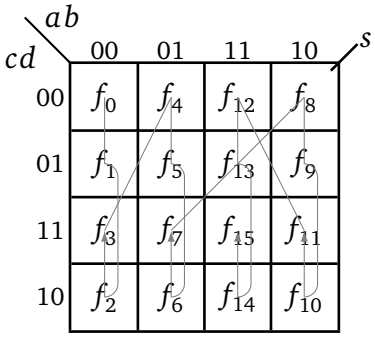
We kunnen dit noteren als:

$$s = \frac{\overline{a} \cdot \overline{b} + a \cdot c + b \cdot \overline{c}}{a \cdot b + \overline{a} \cdot \overline{c} + \overline{b} \cdot c} \tag{4.22}$$

De deelstreep tussen de twee functies geeft aan dat een van de twee functies kan worden gebruikt. In vergelijking (4.20) herkennen we de eerder op algebraïsche wijze geminimaliseerde functie uit (4.12).

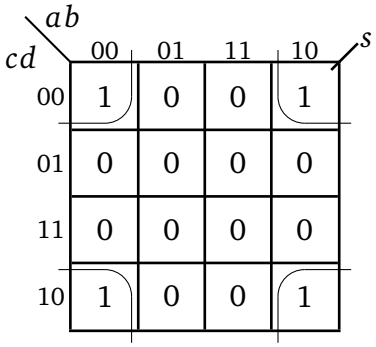
Figuur 4.15 laat de structuur en de plaats van de functiewaarden van het Karnaughdiagram voor vier variabelen zien. De twee meest significante variabelen (a en b) worden boven de schuine streep geplaatst en de bijbehorende variabelewaarden worden, net als bij een Karnaughdiagram voor drie variabelen, boven de kolommen genoteerd. Onder de schuine streep worden de twee minst significante variabelen geplaatst (c en d). Bij de rijen worden weer de bijbehorende variabelewaarden genoteerd uiteraard weer volgens de Gray-codering. Het invullen gaat weer kolomsgewijs maar let erop dat de functiewaarden van f_0 t/m f_3 nu in één kolom geplaatst worden. Het invullen wordt in de figuur aangegeven door de grijze lijnen.

De bovenrand zit vast aan de onderrand en de linkerrand zit vast aan de rechterrand. Dit houdt in dat de vier hoekpunten één geheel vormen. Dit is te zien in figuur 4.16.



a	b	c	d	s
0	0	0	0	f_0
0	0	0	1	f_1
0	0	1	0	f_2
0	0	1	1	f_3
		⋮		
1	1	0	0	f_{12}
1	1	0	1	f_{13}
1	1	1	0	f_{14}
1	1	1	1	f_{15}

Figuur 4.15: *Karnaughdiagram en waarheidstabel voor vier variabelen.*



Figuur 4.16: *De hoekpunten van het Karnaughdiagram zitten aan elkaar vast.*

We kunnen dit aantonen met behulp van schakelalgebra:

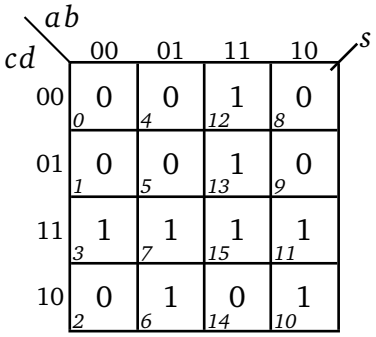
$$\begin{aligned} s &= \sum m(0,2,8,10) \\ &= \overline{a}\cdot\overline{b}\cdot\overline{c}\cdot\overline{d} + \overline{a}\cdot\overline{b}\cdot c\cdot\overline{d} + a\cdot\overline{b}\cdot\overline{c}\cdot\overline{d} + a\cdot\overline{b}\cdot c\cdot\overline{d} \\ &= (\overline{a}\cdot\overline{c} + \overline{a}\cdot c + a\cdot\overline{c} + a\cdot c)\cdot\overline{b}\cdot\overline{d} \\ &= \overline{b}\cdot\overline{d} \end{aligned} \tag{4.23}$$

Laten we eens de functie:

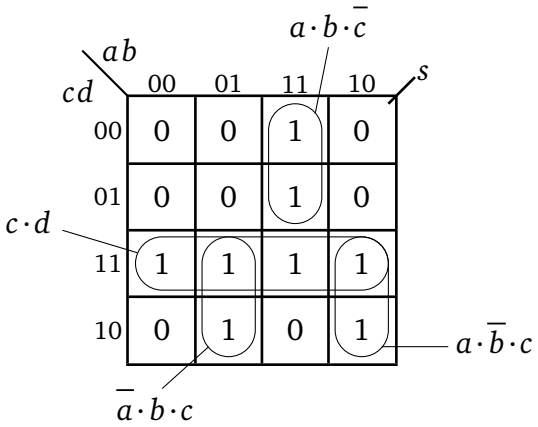
$$s = \sum m(3,6,7,10,11,12,13,15) \tag{4.24}$$

minimaliseren door middel van een Karnaughdiagram. Dit is weergegeven in figuur 4.17. In de figuur zijn ook de mintermnummers te zien.

Het uitgewerkte Karnaughdiagram is te zien in figuur 4.18. We gaan als volgt te werk. We zoeken in het diagram eerst de enen die omrand worden door veel nullen, zoals minterm m_{12}. Deze minterm valt niet te combineren met de mintermen m_4, m_8 en m_{14} want dat zijn allemaal nullen. Minterm m_{12} is alleen te combineren met minterm m_{13}. Samen levert dat de productterm $a\cdot b\cdot\overline{c}$ op.



Figuur 4.17: *Ingevuld Karnaughdiagram voor een functie van vier variabelen.*



Figuur 4.18: *Uitgewerkt Karnaughdiagram voor een functie van vier variabelen.*

Hetzelfde geldt voor minterm m_6. Die kan alleen worden gecombineerd met minterm m_7. De andere omringende mintermen zijn allemaal 0. De combinatie van m_6 en m_7 levert de productterm $\overline{a} \cdot b \cdot c$ op. Verder is te zien dat minterm m_{10} alleen gecombineerd kan worden met minterm m_{11}. Dit levert de productterm $a \cdot \overline{b} \cdot c$ op.

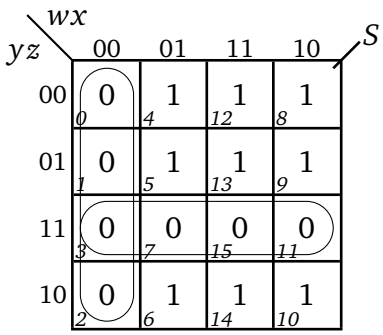
Er zijn nog twee mintermen over die niet gecombineerd zijn, namelijk minterm m_3 en minterm m_{15}. Te zien is dat deze enen op één rij liggen met twee andere enen, de mintermen m_7 en m_{11}. We mogen deze enen samennemen tot de productterm $c \cdot d$ omdat alle mogelijke combinaties van a en b hierin voorkomen. Dat is goed te zien in het Karnaughdiagram. In de omranding van $c \cdot d$ komen alle mogelijke combinaties van a en b voor: 00, 01, 10 en 11. De gevonden functie is:

$$s = a \cdot b \cdot \overline{c} + a \cdot \overline{b} \cdot c + \overline{a} \cdot b \cdot c + c \cdot d \tag{4.25}$$

Het vinden van de POS-vorm van een functie

Het is ook mogelijk om de schakelfunctie in POS-vorm (product van sommen) te vinden. Hiervoor omranden we niet de enen maar de nullen. We vinden hierdoor in eerste instantie de inverse van de functie in SOP-vorm. Door toepassen van de stellingen van De Morgan krijgen we de POS-vorm.

In figuur 4.19 is een Karnaughdiagram gegeven voor een functie van vier variabelen. Na het omranden van de nullen volgt de inverse functie in SOP-vorm:



Figuur 4.19: *Karnaughdiagram.*

$$\overline{S} = \overline{w} \cdot \overline{x} + y \cdot z \tag{4.26}$$

Inverteren van de functie levert dan:

$$S = \overline{\overline{S}} = \overline{\overline{w} \cdot \overline{x} + y \cdot z} \tag{4.27}$$

Twee keer toepassen van de stellingen van De Morgan geeft:

$$\begin{aligned} S &= \overline{\overline{w} \cdot \overline{x} + y \cdot z} \\ &= \overline{\overline{w} \cdot \overline{x}} \cdot \overline{y \cdot z} \\ &= (w + x) \cdot (\overline{y} + \overline{z}) \end{aligned} \tag{4.28}$$

De functie staat nu in de POS-vorm.

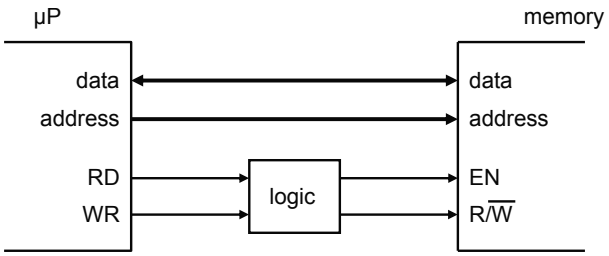
Don't cares

Soms is de uitgangswaarde van een functie bij een bepaalde ingangscombinatie niet gespecificeerd. We spreken dan van een don't care. In een Karnaughdiagram geven we dit aan met een halfhoog streepje (–). Don't cares kunnen bij minimalisatie als een logische 0 of 1 ingevuld worden als dat de functie vereenvoudigt.

Voorbeeld

Een bepaalde microprocessor wordt verbonden met een bepaalde geheugenchip. Een principeschema is te zien in figuur 4.20. De processor (in de figuur aangegeven met µP) heeft twee stuursignalen RD en WR waarmee wordt aangegeven of er uit het geheugen gelezen ($RD = 1$) of naar het geheugen geschreven ($WR = 1$) moet worden. Via de *adresbus* wordt aangegeven op welk adres er gelezen of geschreven moet worden. Via de *databus* wordt de data verstuurd. Merk op dat de databus *bidirectioneel* is: de data kan twee kanten op.

De geheugenchip heeft twee stuursignalen EN en $R/\overline{W}$. Om het geheugen te activeren voor lezen of schrijven moet $EN = 1$ zijn. Het signaal $R/\overline{W}$ heeft een dubbele werking. Bij $R/\overline{W} = 1$ wordt uit het geheugen gelezen, bij $R/\overline{W} = 0$ wordt naar het geheugen geschreven.



Figuur 4.20: *Principeschema microprocessor-interface.*

Om de microprocessor met het geheugen te laten werken, is een schakeling nodig voor het vertalen van de signalering. Het zal duidelijk zijn dat de signalen RD en WR nooit tegelijk logisch 1 zijn, de processor kan immers niet tegelijk lezen en schrijven. Wanneer beide signalen logisch 0 zijn, wordt de geheugenchip niet geactiveerd en is de waarde van signaal $R/\overline{W}$ niet van belang. Voor deze schakeling is een waarheidstabel op te stellen, zie tabel 4.2.

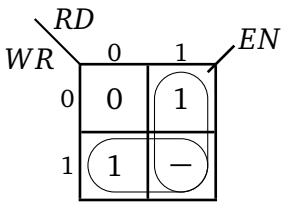
Tabel 4.2: *Waarheidstabel vertaalschakeling microprocessor-interface.*

RD	WR	EN	$R/\overline{W}$	opmerking
0	0	0	–	geen actie
0	1	1	0	schrijven naar geheugen
1	0	1	1	lezen uit geheugen
1	1	–	–	komt niet voor

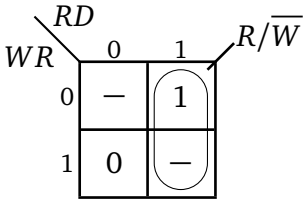
De functies voor EN en $R/\overline{W}$ zijn:

$$\begin{aligned} EN &= \sum m(1,2) + d(3) \\ R/\overline{W} &= \sum m(2) + d(0,3) \end{aligned} \tag{4.29}$$

We vullen de functies in een Karnaughdiagram in, zie figuur 4.21.



(a) *Karnaughdiagram voor EN.*



(b) *Karnaughdiagram voor $R/\overline{W}$.*

Figuur 4.21: *Karnaughdiagrammen voor de microprocessor-interface.*

Na uitwerking volgen de functies:

$$\begin{aligned} EN &= RD + RW \\ R/\overline{W} &= RD \end{aligned} \tag{4.30}$$

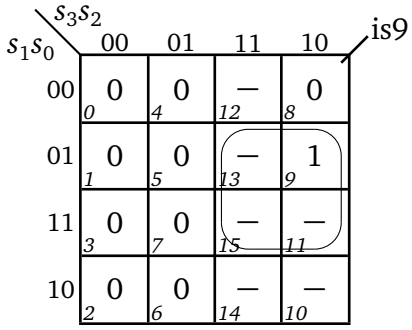
Opmerking: uit het oogpunt van veiligheid is het slim om de don't care in de functie voor EN als een 0 in te vullen. De functie is dan $EN = RD \oplus WR$. Hiermee wordt voorkomen dat eventuele foutieve aansturing vanuit de microprocessor (er zou een bug in kunnen zitten) tot een lees- of schrijfactie leidt.

Voorbeeld

Een BCD-cijfer wordt weergegeven met vier bits. Tien combinaties worden gebruikt, zes combinaties niet. We willen nu een functie bepalen die aangeeft dat het BCD-cijfer de waarde 9 heeft. Dit is onder andere nodig voor de transportfunctie van een BCD-teller. De functie *moet* een logische 1 geven als combinatie 1001 (minterm 9) wordt aangeboden. Aangezien de combinaties 1010 t/m 1111 niet voorkomen, worden deze ingevuld als don't care. De functie in de mintermvorm is:

$$is9 = \sum m(9) + d(10,11,12,13,14,15) \tag{4.31}$$

Het ingevulde en uitgewerkte Karnaughdiagram is weergegeven in figuur 4.22. Hierbij zijn ook de mintermnummers te zien.



Figuur 4.22: *Karnaughdiagram voor het bepalen van de stand 9.*

Minterm 9 is te combineren met minterm 11 en minterm 13. Dat levert in beide gevallen een productterm op van drie variabelen. Nu zijn minterm 11 en minterm 13 ook te combineren met minterm 15. We kunnen de logische 1 en drie don't cares dus samen nemen in één groep van vier mintermen. Dat levert een productterm op van twee variabelen. De drie don't cares in de omranding worden na uitwerking logisch 1. De gevonden functie is dan:

$$is9 = s_3 \cdot s_0 \tag{4.32}$$

Regels voor het uitlezen van Karnaughdiagrammen

Algemeen gezien zoeken we naar de meest eenvoudige uitdrukking van een functie. We kiezen dan voor een minimum aan groepen met zoveel mogelijk enen per groep (we gaan uit van de SOP-vorm van de functie). Begin echter met enen die omringd zijn door nullen. Deze enen horen bij producttermen die waarschijnlijk tot de uiteindelijke functie behoren. Hieronder staat een aantal spelregels voor het uitlezen van Karnaughdiagrammen:

- Zo weinig mogelijke groepen enen maken.

- Zo groot mogelijke groepen enen maken.
- Altijd groepen enen bij elkaar nemen in machten van 2 (1, 2, 4, 8, etc).
- Groepen mogen alleen rechthoeken en vierkanten vormen.
- Groepen mogen elkaar overlappen (maar het is geen noodzaak).
- De randen zijn met elkaar verbonden.
- Don't cares kunnen meegenomen worden in de omrandingen als de schakelfunctie hierdoor eenvoudiger wordt.

maar:

- Begin met enen die omringd worden door veel nullen ("zielige ééntjes").
- Diagonaal geplaatste enen kunnen niet samengenomen worden (EXOR!).
- Maak geen groepen met alleen don't cares.

*4.3.3 Quine-McCluskey

Karnaughdiagrammen kunnen praktisch worden gebruikt tot vijf variabelen. Bij meer variabelen wordt het gebruik ervan lastig. Voor problemen met een groter aantal variabelen kan een tabellarische methode ontworpen door Quine en McCluskey [71, 72] gebruikt worden. Deze methode is niet gebaseerd op het visueel herkennen van patronen maar werkt direct op de producttermen van een functie. Mintermen vallen hier ook onder, een minterm is immers een productterm. De methode is geschikt voor pen en papier én voor gebruik in een computerprogramma.

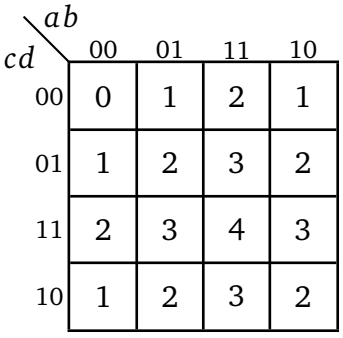
We hebben al eerder gezien dat twee mintermen te combineren zijn als ze slechts in één variabele verschillen. We kunnen dit in een tabel aangeven. Hiertoe rangschikken we de mintermen naar het aantal enen dat in de binaire representatie van de minterm voorkomt. Voor vier variabelen is dit te zien in tabel 4.3. Minterm 0 heeft geen enen. Deze minterm wordt in Groep 0 geplaatst. De mintermen 1, 2, 4 en 8 hebben alle één een en worden in Groep 1 geplaatst. Groep 2 bestaat uit de mintermen 3, 5, 6, 9, 10 en 12. Groep 3 bestaat uit de mintermen 7, 11, 13 en 14. De laatste groep wordt gevormd door minterm 15. Deze heeft vier enen. In figuur 4.23 is een Karnaughdiagram getekend met in de hokjes het aantal enen in de mintermen.

Vanuit de tabel is te zien dat de minterm van Groep 0 gecombineerd kan worden met alle mintermen van Groep 1. Zo levert combineren van 0000 (m_0) en 0001 (m_1) als resultaat 000—. Het halfhoge streepje (don't care) geeft hierbij aan dat deze variabele bij het samennemen verdwijnt. Het is trouwens wel noodzakelijk om een — te schrijven. Het kan niet weggelaten worden want dan staat er 000 en dan weten we niet welke variabele we kunnen weglaten. In de literatuur wordt ook wel een 2 gebruikt in plaats van een —.

Het is niet mogelijk om minterm 0 te combineren met mintermen uit Groep 2 omdat deze groep alleen mintermen heeft die twee enen hebben. Hetzelfde geldt voor de overige groepen. We moeten dus alleen met de *aangrenzende hogere* groep combineren. Verder

Tabel 4.3: *Mintermen gerangschikt naar het aantal enen in de binaire representatie.*

	minterm	a	b	c	d
Groep 0	0	0	0	0	0
Groep 1	1	0	0	0	1
	2	0	0	1	0
	4	0	1	0	0
	8	1	0	0	0
Groep 2	3	0	0	1	1
	5	0	1	0	1
	6	0	1	1	0
	9	1	0	0	1
	10	1	0	1	0
	12	1	1	0	0
Groep 3	7	0	1	1	1
	11	1	0	1	1
	13	1	1	0	1
	14	1	1	1	0
Groep 4	15	1	1	1	1



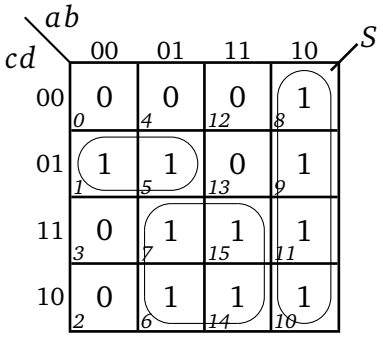
Figuur 4.23: *Het aantal enen in de mintermen.*

moet worden opgemerkt dat het niet altijd mogelijk is om mintermen uit aangrenzende groepen te combineren. Zo kan minterm 1 uit Groep 1 niet gecombineerd worden met minterm 6 uit Groep 2.

We leggen de methode uit aan de hand een voorbeeld. We gaan de volgende functie minimaliseren:

$$S = \sum m(1,5,6,7,8,9,10,11,14,15) \tag{4.33}$$

Links in figuur 4.24 is het Karnaughdiagram getekend. Het dient als ondersteuning van het proces. (En het is ook bijzonder handig bij het uitwerken van de tabellen). Rechts zijn de mintermen te zien, gerangschikt naar het het aantal enen. We zullen tijdens het voorbeeld ook een aantal begrippen introduceren.



1: 1 8
2: 5 6 9 10
3: 7 11 14
4: 15

Figuur 4.24: *Uitgewerkt Karnaughdiagram en mintermen gerankschikt naar het aantal enen.*

Tabel 4.4: *Uitwerking van de tabellen.*

Kolom I.

	a	b	c	d	m.t.	
Groep 1	0	0	0	1	1	✓
	1	0	0	0	8	✓
Groep 2	0	1	0	1	5	✓
	0	1	1	0	6	✓
	1	0	0	1	9	✓
	1	0	1	0	10	✓
Groep 3	0	1	1	1	7	✓
	1	0	1	1	11	✓
	1	1	1	0	14	✓
Groep 4	1	1	1	1	15	✓

Kolom II.

a	b	c	d	m.t.	
0	–	0	1	1,5	*A*
–	0	0	1	1,9	*B*
1	0	0	–	8,9	✓
1	0	–	0	8,10	✓
0	1	–	1	5,7	*C*
0	1	1	–	6,7	✓
–	1	1	0	6,14	✓
1	0	–	1	9,11	✓
1	0	1	–	10,11	✓
1	–	1	0	10,14	✓
–	1	1	1	7,15	✓
1	–	1	1	11,15	✓
1	1	1	–	14,15	✓

Kolom III.

a	b	c	d	m.t.	
1	0	–	–	8,9,10,11	*D*
1	0	–	–	8,10,9,11	
–	1	1	–	6,7,14,15	*E*
–	1	1	–	6,14,7,15	
1	–	1	–	10,11,14,15	*F*
1	–	1	–	10,14,11,15	

We stellen eerst een tabel op met alle mintermen die tot de functie behoren. We rangschikken de mintermen naar het aantal enen. Dit is te zien in Kolom I van tabel 4.4.

Daarna combineren we mintermen uit een groep en de aangrenzende hogere groep als beide mintermen in slechts één variabele verschillen. Die variabele wordt dan verwijderd. Zo kunnen de mintermen m_1 en m_5 gecombineerd worden tot de productterm 0—01. We noemen deze producterm een *implicant*. Een implicant is een productterm van één of meer variabelen die tot de functie behoren. Een minterm is dus ook een implicant. In kolom I vinken we de gecombineerde mintermen af en schrijven de implicanten in een tweede tabel, weer gerangschikt naar het aantal enen. Dit is te zien in Kolom II. De verwijderde variabele wordt geschreven met een halfhoog streepje, oftewel een don't care. Achter elke regel schrijven we ook de mintermen die bij de implicant horen.

We herhalen de vorige stap voor de tweede tabel. We onderzoeken weer of twee implicanten kunnen worden samengenomen tot een nieuwe implicant met minder variabelen. De implicanten moeten slechts in één variabele verschillen maar nu moet er ook op gelet worden dat eventuele don't cares op dezelfde posities staan. Deze variabele is immers verwijderd en doet niet mee. We vinken de implicanten af en schrijven de nieuwe combinaties in een derde tabel op, zie kolom III.

Het kan zijn dat een implicant niet meer gecombineerd kan worden (uiteraard geldt dat uiteindelijk voor alle implicanten). Zo'n implicant wordt een *priemimplicant* genoemd. Een priemimplicant komt overeen met een zo'n groot mogelijke omranding in een Karnaughdiagram. Achter een priemimplicant wordt een letter geschreven. Dit is te zien in Kolom II voor de implicanten (1,5), (1,9) en (5,7). Deze letters zijn in een latere stap nodig.

We herhalen het proces totdat er geen combinaties meer mogelijk zijn en er alleen nog maar priemimplicanten over zijn. Merk op dat sommige implicanten dubbel voorkomen. In Kolom III is dat bijvoorbeeld het geval met de implicanten (8,9,10,11) en (8,10,9,11).

Uiteraard kan één van de twee geschrapt worden.

We hebben nu de volgende priemimplicanten gevonden:

$$\begin{aligned} 0-01 &= A & 10--&= D \\ -001 &= B & -11-&= E \\ 01-1 &= C & 1-1-&= F \end{aligned}$$

Dit zijn *alle* priemimplicanten en dit leidt in de regel niet tot een minimale dekking van alle enen van de functie. We zoeken nu nog naar de kleinste verzameling van priemimplicanten die de functie dekt. We stellen hiervoor een dekkingstabel op, te zien in tabel 4.5. In de kolommen schrijven we de mintermen op. In de rijen geven we per priemimplicant aan of een minterm behoort tot de priemimplicant. Dit is te zien aan de x-en in de tabel.

Tabel 4.5: *Beginsituatie dekkingstabel.*

	m_1	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{14}	m_{15}
A	x	x								
B	x					x				
C		x		x						
D					x	x	x	x		
E			x	x					x	x
F							x	x	x	x

De eerste stap is om te zoeken naar kolommen waar slechts één x staat. Dit is het geval bij de mintermen m_6 van E en m_8 van D. Deze priemimplicanten zijn noodzakelijk voor de functie en worden *essentiële priemimplicanten* genoemd. Dit is in tabel 4.6 aangegeven met een (x). Verder is te zien dat de mintermen van F volledig worden gedekt door de mintermen van D en E. Priemimplicant F is niet nodig voor de functie en wordt geschrapt. We moeten nu nog kiezen uit (een combinatie van) A, B en C. De mintermen van B worden gedekt door A en D, die van C worden gedekt door A en E. De mintermen van A worden niet gedekt door D en E. We kunnen nu B en C schrappen, want A dekt de overgebleven mintermen af.

Tabel 4.6: *Uitgewerkte dekkingstabel.*

	m_1	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{14}	m_{15}
A	x	x								
~~B~~	~~x~~					~~x~~				
~~C~~		~~x~~		~~x~~						
D					(x)	x	x	x		
E			(x)	x					x	x
~~F~~							~~x~~	~~x~~	~~x~~	~~x~~

De minimale dekking voor de functie bestaat uit de priemimplicanten A, D en E. De functie is dan:

$$\begin{aligned} S &= A + D + E \\ &= \overline{a}\cdot\overline{c}\cdot d + a\cdot\overline{b} + b\cdot c \end{aligned} \tag{4.34}$$

Het is niet altijd direct duidelijk wat de minimale dekking van de functie is. Gelukkig bestaat er een systematische methode. Laten we de dekking door de mintermen nog eens op een rijtje zetten:

- minterm m_1 kan gedekt worden door A of B.
- minterm m_5 kan gedekt worden door A of C.
- minterm m_6 kan alleen gedekt worden door E.

  ⋮
- minterm m_{14} gedekt worden door E of F.
- minterm m_{15} gedekt worden door E of F.

De dekking voor de functie is compleet als elke minterm voorkomt in de functie. Dat houdt in dat we voor de dekking kunnen kiezen uit A of B voor m_1 én uit A of C voor m_5 én E voor m_6 etc. We kunnen deze voorwaarden in een Petrick-functie [73] schrijven:

$$P_S = (A+B)\cdot(A+C)\cdot E\cdot(C+E)\cdot D\cdot(B+D)\cdot(D+F)\cdot(E+F) \tag{4.35}$$

Als voor een bepaalde combinatie van A t/m F volgt dat $P_S = 1$, dan is er een dekking gevonden. We weten alleen nog niet of dit minimaal is. Daarom minimaliseren we de functie volgens de regels van de schakelalgebra. We maken gebruik van de absorptiewet $a\cdot(a+b)=a$ zodat een aantal somtermen geschrapt kan worden. Daarna vermenigvuldigen we de somtermen uit:

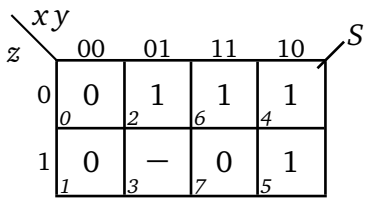
$$\begin{aligned} P_S &= (A+B)\cdot(A+C)\cdot E\cdot(C+E)\cdot D\cdot(B+D)\cdot(D+F)\cdot(E+F)\\ &= (A+B)\cdot(A+C)\cdot D\cdot E\\ &= (A+B\cdot C)\cdot D\cdot E\\ &= A\cdot D\cdot E + B\cdot C\cdot D\cdot E \end{aligned} \tag{4.36}$$

De functie is niet verder te minimaliseren. De functie P_S is 1 als één van beide producttermen 1 is. Beide producttermen bevatten de priemimplicanten die de functie volledig afdekken. Uiteraard zoeken we de productterm die de minimale dekking oplevert. De minimale dekking wordt dus gerealiseerd door de priemimplicanten A, D en E.

Don't Cares

De methode kan worden aangepast voor het gebruik van don't cares. Don't cares worden in eerste instantie meegenomen als enen in de functie. De gebruikelijke procedure voor het vinden van priemimplicanten wordt gevolgd. Bij het opstellen en uitwerken van de dekkingstabel worden de don't cares weggelaten, want ze behoren niet noodzakelijk tot de functie.

Als voorbeeld minimaliseren we de functie rechts in figuur 4.25. Links is het Karnaugh-diagram te zien. Minterm m_3 is als don't care ingevuld. We zoeken eerst naar de priemimplicanten, te zien in Kolom I en II van tabel 4.7. Bij het opstellen van de dekkingstabel laten we minterm m_3 weg. Minterm m_5 wordt alleen gedekt door C, dus C is een essentiële priemimplicant. Dat betekent automatisch dat voor minterm m_4 ook een dekking is



$$S = \sum m(2,4,5,6) + d(3) \tag{4.37}$$

Figuur 4.25: *De functie en het Karnaughdiagram als ondersteuning.*

gevonden. Priemimplicant B dekt zowel m_2 als m_6, dus A is niet nodig. Verder is te zien dat priemimplicant D niet nodig is omdat de bijbehorende mintermen gedekt worden door B en C. De minimale dekking wordt dus gerealiseerd door B en C. Merk trouwens op dat als m_3 een logische 1 is, A een essentiële priemimplicant is. Minterm m_3 kan niet anders worden gecombineerd dan met m_2.

Tabel 4.7: *Uitwerking van de implicantentabellen.*

Kolom I.

	x	y	z	m.t.	
Groep 1	0	1	0	2	✓
	1	0	0	4	✓
Groep 2	0	1	1	3	✓
	1	0	1	5	✓
	1	1	0	6	✓

Kolom II.

x	y	z	m.t.	
0	1	—	2,3	A
—	1	0	2,6	B
1	0	—	4,5	C
1	—	0	4,6	D

Dekkingstabel.

	m_2	m_4	m_5	m_6
~~A~~	x			
B	x			x
C		x	(x)	
~~D~~		x		x

De Petrick-functie voor deze functie is:

$$\begin{aligned} P_S &= (A+B)\cdot(C+D)\cdot C\cdot(B+D) \\ &= (A+B)\cdot C\cdot(B+D) \\ &= (A+B)\cdot(B+D)\cdot C \\ &= (B+A\cdot D)\cdot C \\ &= B\cdot C + A\cdot C\cdot D \end{aligned} \tag{4.38}$$

De minimale functie wordt gevormd door de priemimplicanten B en C.

4.3.4 Andere vereenvoudigingsmethoden

Karnaughdiagrammen kunnen worden gebruikt tot vijf variabelen en zijn arbeidsintensief. Het Quine-McCluskey-algoritme is geschikt om in een computerprogramma te verwerken. Helaas is deze methode onbruikbaar bij grote aantallen variabelen, de rekentijd loopt exponentieel op. Bij *topologisch minimaliseren* wordt niet uitgegaan van mintermen maar van (al bestaande) implicanten. Zie [74] voor meer informatie. De bovenstaande methoden leveren een gegarandeerde minimale dekking voor een functie. Die wordt alleen gevonden door alle mogelijke combinaties te onderzoeken. Andere methoden, zoals Espresso [75], pogen om met minimale rekentijd een bijna-minimale oplossing te vinden.

Verder merken we op dat veel schakelingen meerdere uitgangen hebben. Zowel Karnaugdiagrammen als Quine-McCluskey kunnen aangepast worden bij *multiple output minimization* waarbij gemeenschappelijke termen gebruikt worden voor het realiseren van de schakeling. We gaan hier verder niet op in. Zie onder andere [76, 77].

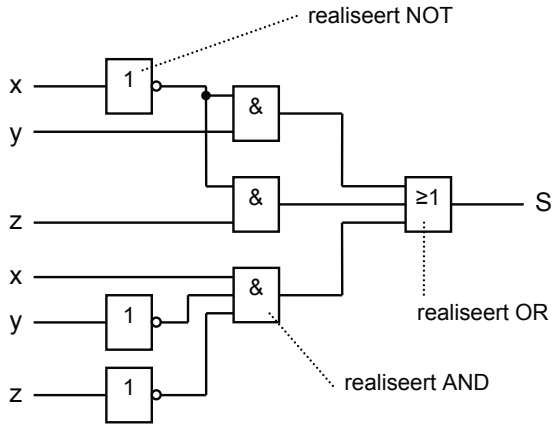
4.4 Realisatie met AND, OR en NOT

Realisatie vanuit de SOP- of POS-vorm is eenvoudig. De schakelfuncties zijn direct afbeeldbaar op AND-, OR- en NOT-poorten. Het levert altijd een schakeling op met drie lagen (of niveaus): één laag met NOT-poorten, één laag met AND-poorten (SOP-vorm) of OR-poorten (POS-vorm) en één laag met OR-poorten (SOP-vorm) of AND-poorten (POS-vorm).

Als voorbeeld nemen we de volgende functie in de SOP-vorm:

$$S = \overline{x} \cdot y + \overline{x} \cdot z + x \cdot \overline{y} \cdot \overline{z} \tag{4.39}$$

De realisatie is te zien in figuur 4.26.



Figuur 4.26: *Schema voor de functie in NOT-AND-OR-vorm.*

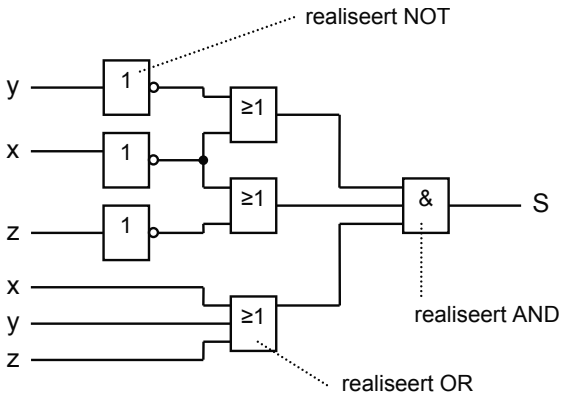
Realisatie in de POS-vorm gaat op vergelijkbare wijze. We gaan uit van POS-vorm van de functie:

$$S = (\overline{x} + \overline{y}) \cdot (\overline{x} + \overline{z}) \cdot (x + y + z) \tag{4.40}$$

De realisatie is te zien in figuur 4.27.

4.5 Realisatie met NAND en NOR

De keuze om schakelingen te realiseren met AND-, OR- en NOT-poorten komt voort uit de toegepaste operatoren uit de schakelalgebra. Uit technologische overwegingen kan gekozen worden voor andere typen poorten, bijvoorbeeld omdat deze eenvoudiger op een ic te realiseren zijn. Er moet dan een extra stap gemaakt worden om van AND, OR en NOT naar de andere operatoren te komen.



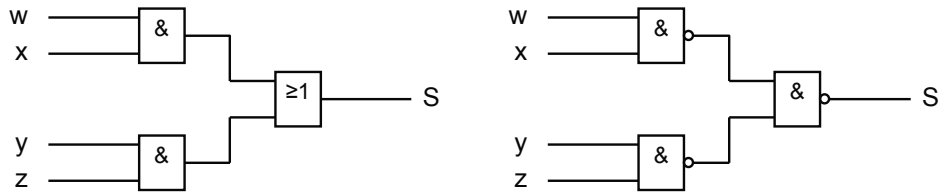
Figuur 4.27: *Schema voor de functie in NOT-OR-AND-vorm.*

Zo zijn NAND- en NOR-poorten met minder transistoren te realiseren dan AND- en OR-poorten. In de meest gebruikte technologie, CMOS, zijn voor een NAND-poort met twee ingangen vier transistoren nodig terwijl voor een AND-poort zes transistoren nodig zijn [78]. Hetzelfde geldt voor NOR- en OR-poorten. Het is dus nuttig NAND- en NOR-poorten te gebruiken bij het implementeren van functies op een ic.

Voor het ontwerpen van een schakeling met NAND-poorten gaan we uit van de SOP-vorm van een functie. We passen eenmaal de stellingen van De Morgan toe om de functie om te werken:

$$\begin{aligned} S &= w \cdot x + y \cdot z \\ &= \overline{\overline{w \cdot x + y \cdot z}} \\ &= \overline{\overline{w \cdot x} \cdot \overline{y \cdot z}} \end{aligned} \tag{4.41}$$

In figuur 4.28 zijn de AND-OR- en NAND-NAND-schakelingen te zien.

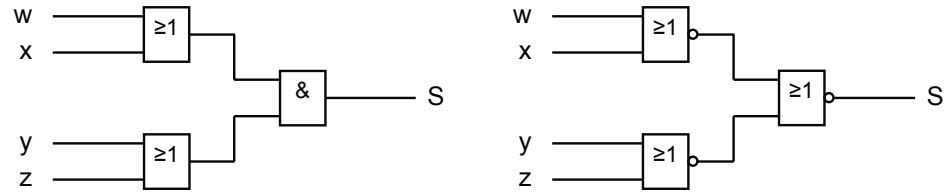


Figuur 4.28: *Schema's met AND-OR en NAND-NAND.*

Voor het ontwerpen van een schakeling met NOR-poorten gaan we uit van de POS-vorm van een functie. We passen eenmaal De Morgan toe om de functie om te werken:

$$\begin{aligned} S &= (w + x) \cdot (y + z) \\ &= \overline{\overline{(w + x)} + \overline{(y + z)}} \\ &= \overline{\overline{w + x} + \overline{y + z}} \end{aligned} \tag{4.42}$$

In figuur 4.29 zijn de OR-AND- en NOR-NOR-schakelingen te zien.



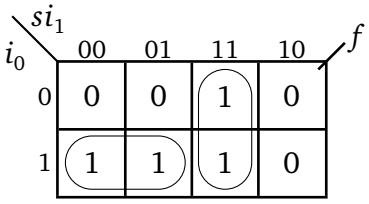
Figuur 4.29: *Schema's met OR-AND en NOR-NOR.*

Verder moet nog gezegd worden dat eventuele NOT-poorten met NAND- en NOR-poorten gerealiseerd kunnen worden. In de CMOS-technologie worden echter gewoon NOT-poorten gebruikt omdat die met twee transistoren te maken zijn [79].

4.6 Realisatie met multiplexers

Een multiplexer is een digitale component die data op een van de data-ingangen doorgeeft aan de data-uitgang onder besturing van een of meerdere besturingsingangen. Een multiplexer selecteert dus een van de data-ingangen, vandaar dat ook wel de naam *selector* wordt gebruikt. Deze component wordt gebruikt wanneer verschillende bronnen data kunnen leveren aan één doel. De multiplexer komt in onder andere microprocessors en routers voor.

De 2x1-multiplexer heeft twee data-ingangen (i_0 en i_1), één stuuringang (s) en één data-uitgang (f). De werking laat zich als volgt beschrijven: de uitgang volgt i_0 als $s = 0$ en volgt i_1 als $s = 1$. We stellen een waarheidstabel op en vullen het Karnaughdiagram in. Dit is te zien in figuur 4.30.



s	i_1	i_0	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

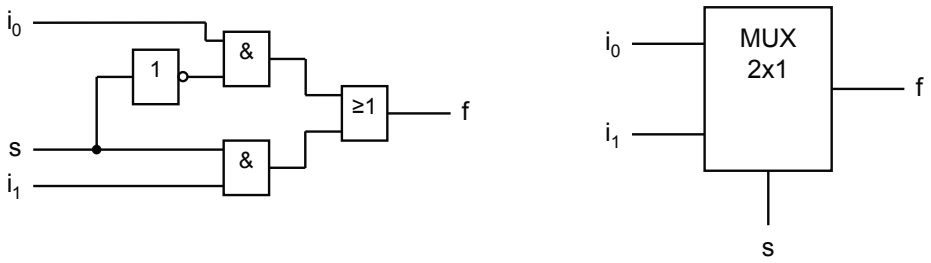
Figuur 4.30: *Karnaughdiagram en waarheidstabel voor 2x1-multiplexer.*

Uitwerken van het Karnaughdiagram levert de functie:

$$f = \bar{s} \cdot i_0 + s \cdot i_1 \tag{4.43}$$

De realisatie met AND, OR en NOT is te zien in figuur 4.31. Tevens is een symbool te zien omdat we deze multiplexer willen gaan gebruiken in grotere schakelingen.

Als we grotere multiplexers willen ontwerpen dan blijkt de aanpak via een waarheidstabel en Karnaughdiagram niet praktisch. Een 4x1-multiplexers heeft vier data-ingangen en twee stuuringangen, een 8x1-multiplexer heeft acht data-ingangen en drie stuuringangen.

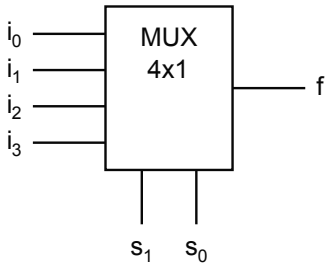


Figuur 4.31: *Schema en symbool van een 2x1-multiplexer.*

Die laatste levert een waarheidstabel op van 2048 regels, niet echt hanteerbaar. Om van het Karnaughdiagram nog maar te zwijgen.

Handiger is om een *functietabel* op te stellen. Hierin wordt niet de uitgang in nullen en enen beschreven, maar komt de functie te staan. In dit geval is de functie erg eenvoudig. Zie figuur 4.32.

s_1	s_0	f
0	0	i_0
0	1	i_1
1	0	i_2
1	1	i_3

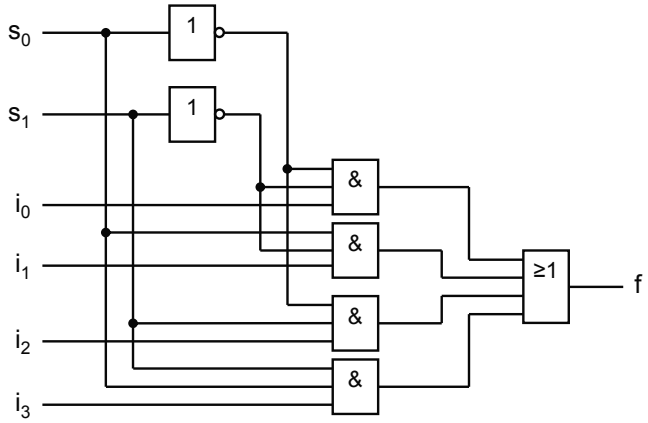


Figuur 4.32: *Functietabel en symbool van een 4x1-multiplexer.*

We schrijven de functie direct uit de functietabel op:

$$f = \overline{s_1} \cdot \overline{s_0} \cdot i_0 + \overline{s_1} \cdot s_0 \cdot i_1 + s_1 \cdot \overline{s_0} \cdot i_2 + s_1 \cdot s_0 \cdot i_3 \tag{4.44}$$

Het schema is direct te realiseren in de SOP-vorm. Zie figuur 4.33. We hebben naast twee NOT-poorten nog vier AND-poorten met drie ingangen en een OR-poort met vier ingangen nodig. Dat levert ook nog eens achttien ingangen op. De schakeling kost nogal wat poorten.



Figuur 4.33: *Schema van een 4x1-multiplexer.*

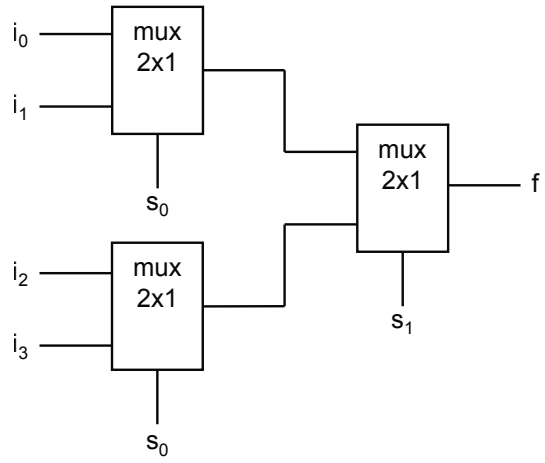
Het is ook mogelijk om een 4x1-multiplexer op te bouwen uit 2x1-multiplexers. We herschrijven de functie (4.44) door $\overline{s_1}$ respectievelijk s_1 buiten haakjes te halen. Dit is te zien in functie (4.45). Wat opvalt is dat binnen de haakjes tweemaal de functie staat van een 2x1-multiplexer waarbij s_0 als stuursignaal dient.

$$f = \overline{s_1} \cdot \underbrace{(\overline{s_0} \cdot i_0 + s_0 \cdot i_1)}_{\text{2x1 mux}} + s_1 \cdot \underbrace{(\overline{s_0} \cdot i_2 + s_0 \cdot i_3)}_{\text{2x1 mux}} \tag{4.45}$$

Vanuit s_1 gezien is dit ook de functie van een 2x1-multiplexer. Dit is goed te zien in functie (4.46) waarbij de beschrijvingen van de andere twee multiplexers zijn vervangen door puntjes.

$$f = \underbrace{\overline{s_1} \cdot (\ldots) \; + \; s_1 \cdot (\ldots)}_{\text{2x1 mux}} \tag{4.46}$$

Het schema is te zien in figuur 4.34. Te zien is dat de uitgangen van de multiplexers aan de linkerkant verbonden zijn met de data-ingangen van de multiplexer rechts. De multiplexer rechts, waar s_1 als stuursignaal op is aangesloten, selecteert een van de twee data-uitgangen van de multiplexers aan de linkerkant. De multiplexer linksboven selecteert aan de hand van het stuursignaal s_0 ingang i_0 of i_1. De multiplexer linksonder selecteert aan de hand van het stuursignaal s_0 ingang i_2 of i_3.

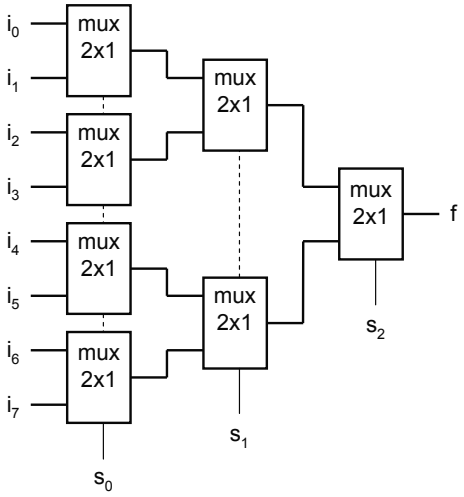


Figuur 4.34: *Schema van een 4x1-multiplexer opgebouws uit 2x1 multiplexers.*

Deze manier van "uit elkaar halen" van een functie wordt *Shannon-decompositie* [67] genoemd. Het is hierdoor mogelijk om elke multiplexer op te bouwen uit 2x1-multiplexers. In figuur 4.35 is de opbouw en de functietabel van een 8x1-multiplexer te zien. De functie is te zien in (4.47).

$$\begin{aligned} s = {} & \overline{s_2} \cdot \overline{s_1} \cdot \overline{s_0} \cdot i_0 + \overline{s_2} \cdot \overline{s_1} \cdot s_0 \cdot i_1 + \overline{s_2} \cdot s_1 \cdot \overline{s_0} \cdot i_2 + \overline{s_2} \cdot s_1 \cdot s_0 \cdot i_3 + \\ & s_2 \cdot \overline{s_1} \cdot \overline{s_0} \cdot i_4 + s_2 \cdot \overline{s_1} \cdot s_0 \cdot i_5 + s_2 \cdot s_1 \cdot \overline{s_0} \cdot i_6 + s_2 \cdot s_1 \cdot s_0 \cdot i_7 \end{aligned} \tag{4.47}$$

Als we de functie goed bekijken dan zien we dat de functie geschreven is als een *som van mintermen*. De variabelen s_0, s_1 en s_2 vormen de mintermen en variabelen i_0 t/m i_7 zijn de functiewaarden. We kunnen nu de multiplexer gebruiken voor het realiseren

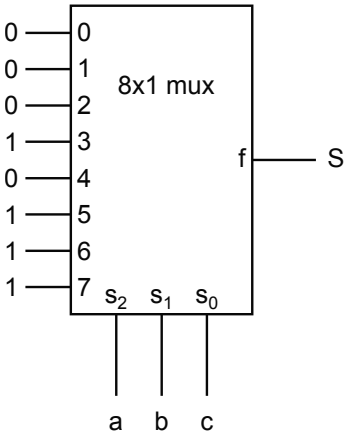


s_2	s_1	s_0	f
0	0	0	i_0
0	0	1	i_1
0	1	0	i_2
0	1	1	i_3
1	0	0	i_4
1	0	1	i_5
1	1	0	i_6
1	1	1	i_7

Figuur 4.35: *Blokschema en functietabel van een 8x1-multiplexer.*

van een willekeurige schakelfunctie. Hiertoe verbinden we de stuuringangen van de multiplexer met de ingangssignalen van de te realiseren schakeling. De functiewaarden van de schakeling worden verbonden met de data-ingangen van de multiplexer.

Als voorbeeld realiseren we de functie $S_{a,b,c} = \sum m(3,5,6,7)$. We verbinden variabele a met stuuringang s_2, b met s_1 en c met s_0. De functiewaarden bij de mintermen 3, 5, 6 en 7 zijn logisch 1, dus de bijbehorende data-ingangen worden met een logische 1 verbonden. De overige data-ingangen worden met een logische 0 verbonden. De realisatie en de waarheidstabel van de functie is te zien in figuur 4.36.



a	b	c	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figuur 4.36: *Realisatie van een logische functie met een multiplexer: aansluitschema en waarheidstabel.*

Deze manier van realiseren van een functie was populair in de jaren '80. Er hoeft immers maar één component gebruikt te worden. Gebruik op een ic is niet zo voor de hand liggend. De functie is met behulp van minimalisatie waarschijnlijk veel compacter (lees: minder transistoren) te bouwen dan wanneer een volledige multiplexer wordt gebruikt.

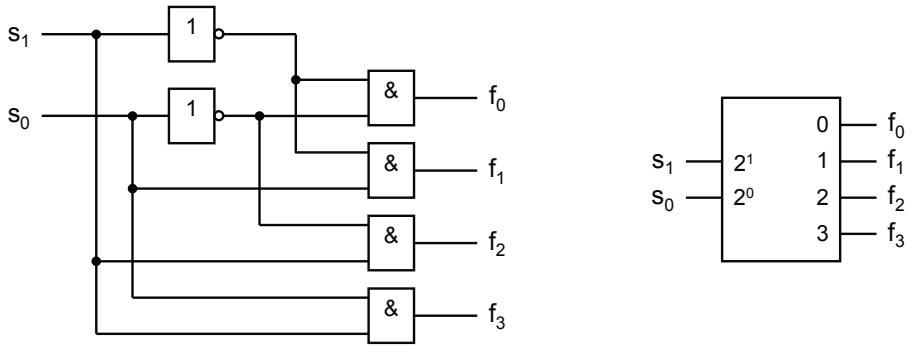
We kunnen echter de data-ingangen *configureerbaar* maken, dat wil zeggen, de gebruiker (ontwerper) kan zelf de waarden van de data-ingangen vastleggen. De data-ingangen worden voorzien van geheugencellen die te programmeren zijn. Deze manier van reali-

seren van een schakeling wordt toegepast in een *Field Programmable Gate Array* (FPGA). Zie [80, 81] voor een uitvoerige introductie over FPGA's.

4.7 Realisatie met decoders en ROM's

Schakelingen kunnen ook op een meer praktische manier ontworpen worden door uit te gaan van de mintermen. We minimaliseren de functie dan niet. Hiervoor maken we gebruik van een *decoder*.

Een decoder is een schakeling die voor elke unieke combinatie van ingangswaarden slechts één uitgang logisch 1 maakt. De andere uitgangen zijn dan logisch 0. Een decoder met n ingangen heeft dus 2^n uitgangen. In figuur 4.37 is het schema en het bloksymbool van een 2-line-to-4-line decoder te zien. Elke uitgang is slechts bij één ingangscombinatie logisch 1.



Figuur 4.37: *Schema en bloksymbool van een 2-line-to-4-line decoder.*

De schakeling realiseert de functies:

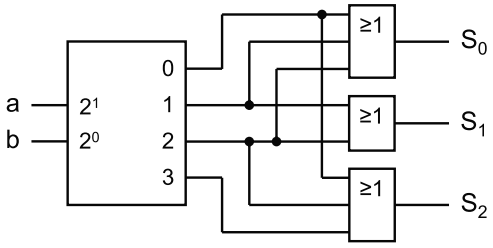
$$\begin{aligned} f_0 &= \overline{s_1} \cdot \overline{s_0} \\ f_1 &= \overline{s_1} \cdot s_0 \\ f_2 &= s_1 \cdot \overline{s_0} \\ f_3 &= s_1 \cdot s_0 \end{aligned} \tag{4.48}$$

Als we de ingangen van een functie aanbieden aan de ingangen van een decoder dan representeert elke uitgang precies één minterm van de functie. De decoder decodeert de mintermen uit. We kunnen nu elke functie realiseren door de benodigde mintermen samen te nemen in een logische som.

In figuur 4.38 worden drie functies gerealiseerd. Er zijn slechts één decoder en drie OR-poorten nodig.

De schakeling realiseert de functies:

$$\begin{aligned} S_0 &= m_0 + m_1 + m_2 = \overline{a \cdot b} \\ S_1 &= m_1 + m_2 = a \oplus b \\ S_2 &= m_0 + m_2 + m_3 = a + \overline{b} \end{aligned} \tag{4.49}$$



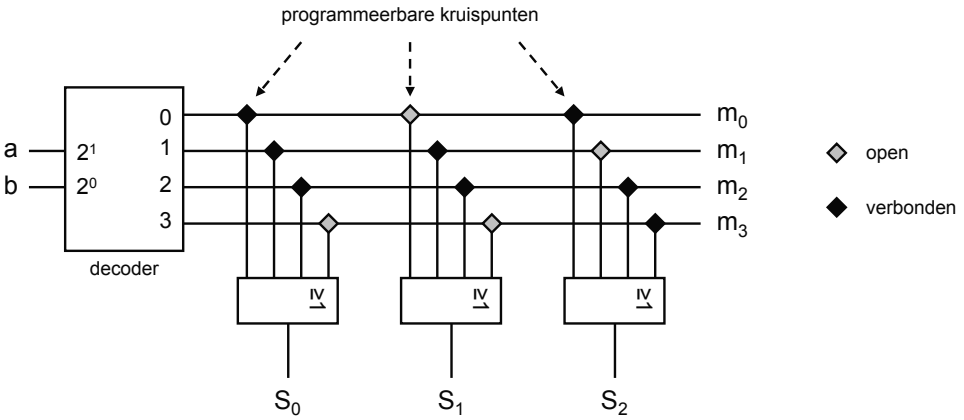
a	b	S_0	S_1	S_2
0	0	1	0	1
0	1	1	1	0
1	0	1	1	1
1	1	0	0	1

Figuur 4.38: *Schema en waarheidstabel van de functies gerealiseerd door decoder en OR-poorten.*

In de jaren ’80 van de vorige eeuw nam de integratiedichtheid van transistoren toe. Het werd economisch rendabel om AND-poorten en OR-poorten massaal op een chip te plaatsen. Fabrikanten begonnen in te zien dat het realiseren van ic’s met een kant-en-klare schakeling de ontwerper veel tijd kon besparen. Dit heeft geleid tot de ontwikkeling van de ROM (*Read Only Memory*). Zoals de naam al aangeeft, ligt de inhoud vast (read only) en kan niet gewijzigd worden. Memory geeft aan dat deze schakeling gebruikt werd als geheugen in een systeem, bijvoorbeeld voor de opstartcode in een computer.

Latere ontwikkelingen hebben geleid tot de Mask-ROM en PROM. In de Mask-ROM ligt de structuur van de AND-poorten vast (mintermen) maar die van de OR-poorten was door de gebruiker in te stellen. Dat moest wel in de fabriek gebeuren. In een PROM (*programmable read-only memory*) worden alle mintermen met AND-poorten uitgecodeerd en zijn de verbindingen naar de OR-poorten door de gebruiker zelf programmeerbaar door middel van een *PROM burner*. Dit levert een behoorlijke tijdwinst en kostenbesparing op bij het realiseren van een schakeling.

In figuur 4.39 is het schema te zien van een PROM die de functies uit (4.49) realiseert. Een zwart ruitje geeft een verbinding aan, een grijs ruitje geeft aan dat er geen verbinding is.



Figuur 4.39: *Principeschema van een (programmeerbare) ROM.*

Overigens mag wel duidelijk zijn dat bij ROM’s met veel ingangen de hoeveelheid logica behoorlijk oploopt. Bij acht ingangen zijn er al 256 mintermen. De decoder bestaat dan uit 256 AND-poorten met elk 8 ingangen. De OR-poorten hebben 256 ingangen nodig om elke minterm te kunnen meenemen in een functie. In de praktijk worden daarom *wired-AND*- en *wired-OR*-technieken toegepast. We gaan hier verder niet op in.

In een PAL (*Programmable Array Logic*) zijn de ingangen van de AND-poorten programmeerbaar en ligt de structuur van de OR-poorten vast. Dit beperkt de functionaliteit.

In een PLA (*Programable Logic Array)* zijn niet alleen de ingangen van de OR-poorten programmeerbaar maar ook de ingangen van de AND-poorten. Het idee van het volledig uitcoderen van de mintermen met een decoder wordt dan verlaten. Het voordeel is dat alleen de producttermen die nodig zijn voor de functies moeten worden gerealiseerd. Gewoonlijk is dat aantal veel lager dan het aantal mintermen van de functies. Nadeel is dat er maar een beperkt aantal AND-poorten beschikbaar is voor het realiseren van producttermen. Ook het aantal ingangen van de OR-poorten is beperkt. Dit wordt gedaan om de hoeveelheid logica (lees: chipoppervlakte) te beperken met als gevolg dat niet elke logische functie kan worden gerealiseerd.

Verder moet worden opgemerkt dat de PAL en PLA gedateerd zijn.

4.8 Ontwerp van een schakeling

De functionaliteit van een schakeling volgt meestal uit een geschreven specificatie. De procedure om tot een schakeling te komen begint met het opstellen van een waarheidstabel voor het probleem. Daarna wordt de functie geminimaliseerd zodat een niet-reduceerbare minimale vorm verkregen wordt (SOP- of POS-vorm). Vervolgens wordt de schakeling gerealiseerd met poorten. Eventueel kan de functie eerst omgewerkt worden naar een vorm waardoor de schakeling met bijvoorbeeld NAND-poorten gerealiseerd kan worden.

In deze paragraaf laten we twee ontwerpen zien. Het eerste ontwerp betreft een schakeling met drie ingangen en één uitgang. Daarbij moet de schakeling ook gerealiseerd worden met NAND-poorten. De tweede schakeling heeft vier ingangen en twee uitgangen. De realisatie is met standaard poorten (NOT, AND en OR).

Voorbeeld

Ontwerp een schakeling met drie ingangen en één uitgang. De uitgang is logisch 1 als de meerderheid van de ingangen logisch 1 is, anders is de uitgang 0. Realiseer de schakeling met NOT-, AND- en OR-poorten. Realiseer de schakeling ook met alleen NAND-poorten.

Uitwerking

Uit de specificatie blijkt dat als twee van de drie ingangen óf drie ingangen logisch 1 zijn, de uitgang logisch 1 moet zijn. In alle andere gevallen moet de uitgang logisch 0 zijn. In tabel 4.8 is de waarheidstabel gegeven. Er zijn vier ingangscombinaties waarbij twee of meer ingangen logisch 1 zijn. Vervolgens wordt de functie geminimaliseerd met behulp van een Karnaughdiagram. Zie figuur 4.40.

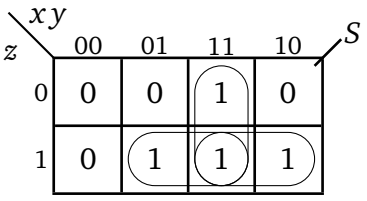
Uit het Karnaughdiagram volgt dat de functie voor S is:

$$S = x \cdot y + x \cdot z + y \cdot z \tag{4.50}$$

De schakeling is direct te realiseren uit deze functie. Om een implementatie met NAND-poorten te realiseren, moet de functie omgewerkt worden door eenmaal de stellingen

Tabel 4.8: *Waarheidstabel voor de schakeling.*

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

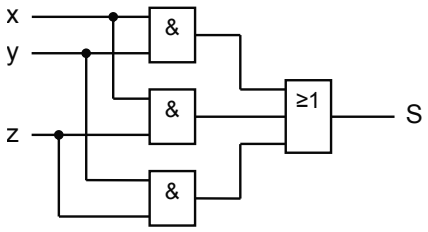


Figuur 4.40: *Karnaughdiagram voor de schakeling.*

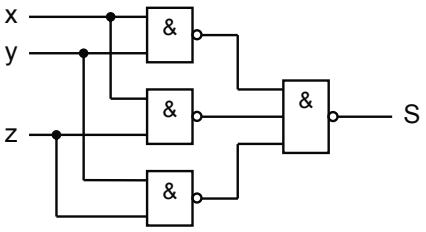
van De Morgan toe te passen.

$$\begin{aligned} S &= x \cdot y + x \cdot z + y \cdot z \\ &= \overline{\overline{x \cdot y} \cdot \overline{x \cdot z} \cdot \overline{y \cdot z}} \end{aligned} \tag{4.51}$$

In figuur 4.41(a) is het schema te zien met standaard poorten. Figuur 4.41(b) laat het schema zien met NAND-poorten.



(a) *Schema met NOT/AND/OR*



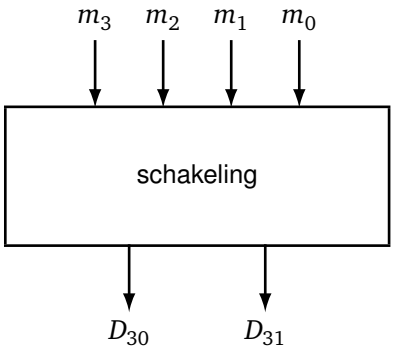
(b) *Schema met NAND/NAND.*

Figuur 4.41: *Realisatie schakeling.*

Voorbeeld

Als onderdeel van een kalendersysteem moet een schakeling worden ontwikkeld die bepaalt of een opgegeven maand 30 of 31 dagen heeft. De maand wordt opgegeven als een 4-bit niet-negatief getal $M = m_3m_2m_1m_0$ waarbij de maanden worden genummerd vanaf 1_{10} (januari) t/m 12_{10} (december). De maanden augustus, december, januari, juli, mei, maart en oktober hebben 31 dagen. De overige maanden (met uitzondering van februari want die heeft 28 óf 29 dagen) hebben 30 dagen. De schakeling heeft twee uitgangen: D_{30} en D_{31}. D_{30} is logisch 1 als de opgegeven maand 30 dagen bevat, anders logisch 0. D_{31} is logisch 1 als de opgegeven maand 31 dagen bevat, anders logisch 0. Zie figuur 4.42.

a) Geef de waarheidstabel voor de functies D_{30} en D_{31}. Maak gebruik van eventuele don't cares.



Figuur 4.42: *Schakeling voor het bepalen van het aantal dagen in een maand.*

b) Teken de Karnaughdiagrammen voor de in a) gevonden functies en geef de vereenvoudigde functies. Laat duidelijk de omrandingen zien.

c) Teken een poortschakeling voor de in b) gevonden functies met alleen NOT, AND en OR. Van AND en OR mogen ook poorten met meer dan twee ingangen gebruikt worden.

Uitwerking

Er zijn twaalf maanden die gecodeerd worden volgens de in het kortschrift voor datums gebruikelijk getallen, dus januari $= 1_{10} = 0001_2$, februari $= 2_{10} = 0010_2$, ..., december $= 12_{10} = 1100_2$. Met vier bits zijn 16 combinaties te maken, te weten van 0000_2 t/m 1111_2. Niet alle combinaties stellen dus een maand voor. Een aantal uitgangswaarden van de functies kan (en moet) dus als don't care gespecificeerd worden. Let erop dat februari zowel geen 30 als 31 dagen heeft. Voor de overige maanden geldt dat de twee functies min of meer elkaars inverse zijn: heeft de maand geen 30 dagen, dan heeft het wel 31 dagen.

(Opgave a) Eerst moet de waarheidstabel worden opgezet. De bitcombinaties $M = m_3m_2m_1m_0$ kunnen gezien worden als getallen die liggen tussen 0 en 15. Van alle mogelijkheden wordt een waarheidstabel opgezet, die is te vinden in tabel 4.9.

(Opgave b) Nadat de waarheidstabel is opgezet, moeten de functies uitgewerkt worden m.b.v. Karnaughdiagrammen. Dit is weergegeven in figuur 4.43.

De gevonden functies voor D_{30} en D_{31} zijn:

$$\begin{aligned} D_{30} &= m_3 \cdot m_0 + \overline{m_3} \cdot m_2 \cdot \overline{m_0} \\ D_{31} &= \overline{m_3} \cdot m_0 + m_3 \cdot \overline{m_0} \end{aligned} \tag{4.52}$$

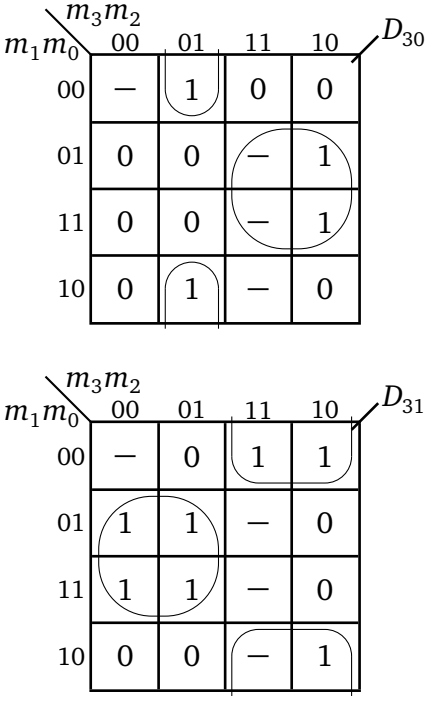
(Opgave c) Het schema voor de functies is weergegeven in figuur 4.44.

4.9 Ingangs- en uitgangsschakelingen

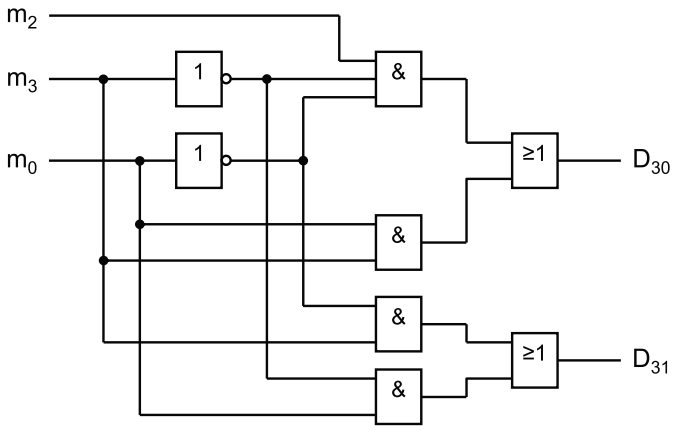
Als ontwerper hebben we te maken met de elektrische eigenschappen van een digitale schakeling. Het is van belang om te weten hoe een schakeling elektrisch reageert op

Tabel 4.9: *Waarheidstabel voor D_{30} en D_{31}.*

maand	m_3	m_2	m_1	m_0	D_{30}	D_{31}
–	0	0	0	0	–	–
jan	0	0	0	1	0	1
feb	0	0	1	0	0	0
maa	0	0	1	1	0	1
apr	0	1	0	0	1	0
mei	0	1	0	1	0	1
jun	0	1	1	0	1	0
jul	0	1	1	1	0	1
aug	1	0	0	0	0	1
sep	1	0	0	1	1	0
okt	1	0	1	0	0	1
nov	1	0	1	1	1	0
dec	1	1	0	0	0	1
–	1	1	0	1	–	–
–	1	1	1	0	–	–
–	1	1	1	1	–	–



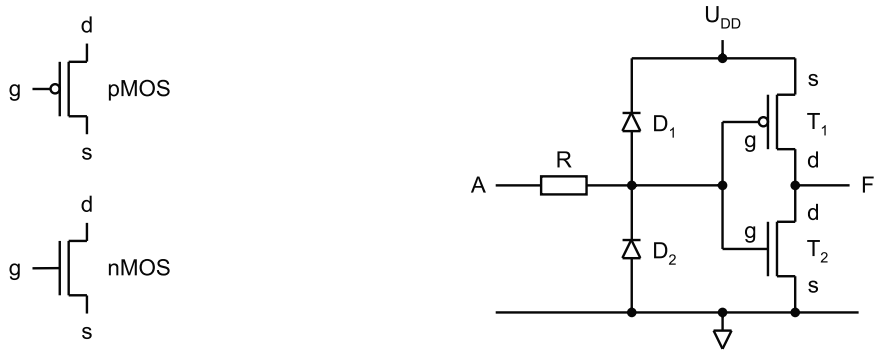
Figuur 4.43: *Karnaughdiagrammen.*



Figuur 4.44: *Schema voor de functies D_{30} en D_{31}.*

zijn omgeving. De digitale ontwerper moet dus ook rekening houden met de analoge techniek.

Er zijn diverse *technologieën* (manieren om transitoren te maken) in omloop, zoals de bipolaire transistor en FET's (Field Effect Transistor). De bipolaire technologie, waaronder TTL, I^2L en ECL, is grotendeels verdwenen uit de markt. We bespreken daarom alleen de MOSFET (Metal Oxide Semiconductor FET, afgekort MOS) omdat dit type transistor dominant is in de huidige chiptechnologie. Het volstaat om de MOS-transistoren te zien als ideale schakelaars. Dat wil zeggen dat de MOS-transistoren stroom geleiden of sperren. Van de vier typen MOS-transistoren zijn alleen de *normally off* (enhancement) varianten



(a) *nMOS- en pMOS-transtoren.*

(b) *CMOS-inverter met ingangsbeveiliging.*

Figuur 4.45: *MOS-transistoren en CMOS-inverter.*

van belang. We zullen alleen een fenomenologische benadering[2] geven. Zie voor een volledig overzicht onder andere [82].

4.9.1 Ingangen en push-pull uitgangen – de CMOS-inverter

In figuur 4.45(a) zijn de symbolen van de nMOS- en pMOS-transistor te zien. De transistoren hebben drie aansluitingen: de *gate* (g), de *drain* (d) en de *source* (s). Tussen de drain en de source is een stukje halfgeleidermateriaal geplaatst (het *kanaal* genoemd) waarvan de geleiding kan worden beïnvloed door een elektrisch veld. Dat veld wordt opgewekt door een spanning aan te brengen tussen de gate en de source.

De nMOS-transistor geleidt als er een positieve spanning tussen gate en source wordt aangeboden. Deze spanning moet boven een zekere drempelspanning (Engels: threshold) liggen. De nMOS geleidt dus als $U_{GS} > U_{TH,nMOS}$. Zo niet, dan spert de transistor (normally off). Voor de pMOS-transistor geldt dat een negatieve spanning tussen gate en source zorgt voor geleiding. Voor deze transistor geldt dat $U_{GS} < U_{TH,pMOS}$ (merk op dat beide spanningen negatief zijn).

ABOUT ZERO VOLTS...

Er zijn nogal wat namen voor de referentie in omloop: 0 V, ground, aarde, referentiespanning enz. Er is dan nogal wat onduidelijkheid. Wij gebruiken in dit boek de term “referentie”.

- 0 V betekent echt 0 V, dus ook ten opzichte van de referentie.
- Aarde betekent echt grond. Dit is ook wat we in huis hebben tegen blikseminslag en voor randaarde.
- Ground wordt gezien als 0 V, maar kan soms als earth ground worden gezien, dus nogal vaag soms.
- Referentie is hetzelfde als ground of 0 V.
- Referentiespanning is hetzelfde als referentie of 0 V.

[2] Dat wil zeggen dat we niet de exacte werking bespreken.

In figuur 4.45(b) is het schema van een CMOS-inverter te zien. De C staat voor *complementary* (Nederlands: complementair, aanvullend). Het bestaat uit een ingangsbeveiliging en de feitelijke inverter. De ingangsimpedantie van de gate is zeer hoog ($R_{in} \approx 10^{12}\ \Omega$) en moet beschermd worden tegen statische spanningen. Daar zorgen diodes D_1 en D_2 voor. Weerstand R zorgt voor demping van hoogfrequentie stoorsignalen van statische ontladingen. De weerstand zorgt ook voor stroombegrenzing als er een spanning hoger de voedingsspanning of lager dan de referentie wordt aangeboden. Als een lage spanning op ingang A wordt gezet, geleidt pMOS-transistor T_1 en spert nMOS-transistot T_2. Dat betekent dat uitgang F met U_{DD} wordt verbonden (push), de weerstand van het kanaal van T_1 is immers klein. Bij een hoge ingangsspanning op ingang A spert T_1 en geleidt T_2. De uitgang wordt dan verbonden met de referentie (pull), de kanaalweerstand van T_2 is klein. In zijn geheel draait de CMOS-inverter de signaalniveaus om. Merk op dat de drains van beide transistoren verbonden zijn met F. Alleen op deze manier kan een negatieve U_{GS} bij de pMOS-transistor worden gerealiseerd.

Tijdens het omschakelen van de ingang geleiden beide transistoren in beperkte mate. Dat betekent dat er een geleidend pad is tussen de voedingsspanning en de referentie waardoor er een stroom gaat lopen. Tijdens de dynamische toestand verbruikt de CMOS-schakeling dus energie. Dit energieverbruik is recht evenredig met de frequentie van het ingangssignaal. Een hogere frequentie betekent dus meer energieverbruik. Op een chip sturen uitgangen alleen CMOS-ingangen aan. De gatestroom is zeer klein en de gate gedraagt zich als een capacitieve belasting. Het opladen en ontladen kost energie. Het vermogen tijdens omschakelen van de CMOS-inverter kan worden bepaald met:

$$P_T = C_L \cdot f \cdot U_{DD}^2 \tag{4.53}$$

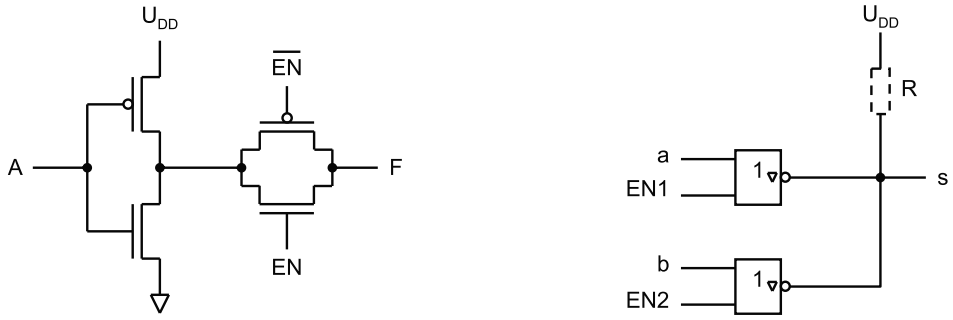
De T in P_T staat voor *transient*. Verder is C_L de capacitieve belasting aan de uitgang, f de schakelfrequentie en U_{DD} de voedingsspanning. Merk op dat de dissipatie kwadratisch oploopt met de voedingsspanning. Het verlagen van de voedingsspanning is dan ook de wens van de chipfabrikanten. De huidige generatie chips werkt op een spanning van 1,2 V tot 3,3 V. Een goede beschrijving van de CMOS-inverter wordt gegeven in [83]. Verder moet worden opgemerkt dat in de huidige generaties CMOS-ic's de statische (lek-)stroom niet meer te verwaarlozen is.

4.9.2 Tri-state uitgangen

Soms is het wenselijk om de logische werking van een poort tijdelijk uit te schakelen. We willen dan dat de poort zich gedraagt alsof de uitgang is losgekoppeld van de signaallijn. Een voorbeeld hiervan is het aansturen van een gemeenschappelijke *buslijn* door meerdere poorten. De genoemde poort heeft dan eigenlijk drie mogelijke uitgangswaarden: logisch 0, logische 1 en hoge impedantie. In Engeltalige literatuur komt de benaming *High-Z* voor. De hoge impedantie zorgt ervoor dat de poort geen invloed heeft op het logisch signaalniveau van de buslijn. Door toevoeging van twee transistoren kan dit gerealiseerd worden.

In figuur 4.46(a) is het schema te zien van een NOT-poort met een tri-state uitgang. Links is de NOT-schakeling te zien. Rechts zijn de transistoren te zien die de uitgangswaarde van de NOT-poort kan doorgeven of blokkeren. De doorgifte wordt gerealiseerd met

pass transistoren. Als stuursignaal EN logisch 0 is, zijn de transistoren gesperd en is de weerstand tussen drain en source zeer hoog. Het signaalniveau van F is dan niet gedefinieerd. We spreken dan ook wel het "zweven" van de uitgang. Een logische 1 op EN zorgt ervoor dat de transistoren geleiden en dat het signaalniveau van de NOT-poort doorgegeven wordt.



(a) *NOT-poort met tri-state-uitgang.* **(b)** *Tri-state NOT-poorten aan een buslijn.*

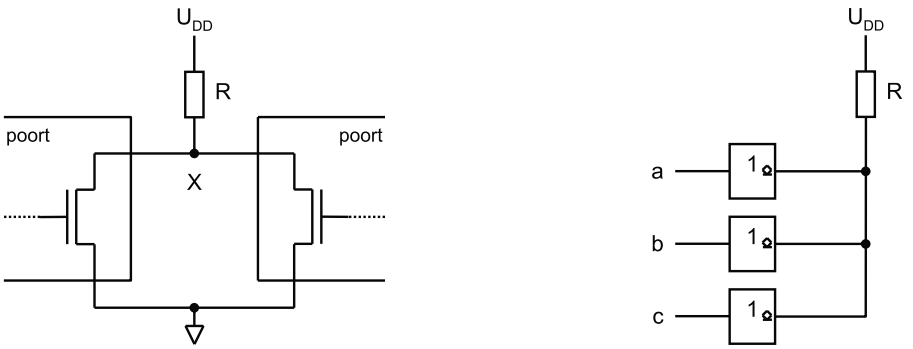
Figuur 4.46: *Het gebruik van tri-state NOT-poorten.*

Meerdere NOT-poorten met tri-state uitgangen zijn aan elkaar te verbinden. Dit is te zien in figuur 4.46(b). In de poortsymbolen is een driehoekje te zien dat aangeeft dat het een tri-state uitgang betreft. Het EN-signaal zorgt ervoor dat de uitgang ingeschakeld wordt. We spreken dan ook wel van een *output enable*. In het ontwerp moet aandacht worden geschonken aan de activering van de uitgangen. De signalen $EN1$ en $EN2$ mogen niet tegelijk logisch 1 zijn want dan kan er kortsluiting ontstaan als beide poorten verschillende spanningen aanbieden. Aan de buslijn wordt soms een pull-up weerstand geplaatst om voor een gedefinieerd logisch niveau te zorgen als alle poorten in tri-state staan.

4.9.3 Open drain uitgangen

Bij open drain uitgangen is de pMOS-transistor uit de uitgangstrap verwijderd. Er is alleen een nMOS-transistor aangebracht. Dat betekent dat de uitgang alleen met de referentie kan worden verbonden. Als de nMOS-transistor spert, zweeft de uitgang. Er is nu een pull-up weerstand nodig om een logisch signaalniveau te definiëren. Dit is te zien in figuur 4.47(a). Dat betekent dat het lage spanningsniveau dominant is over het hoge spanningsniveau. Elke uitgang kan signaal X "omlaag trekken" maar voor een hoog spanningsniveau moeten alle nMOS-transistoren sperren. Deze constructie wordt *wired-OR* en/of *wired-AND* genoemd. De verwarring ontstaat door hoe we naar de werking van de schakeling kijken. Bij wired-OR gaan we uit van een laag signaalniveau en de eigenschap dat elke uitgang de spanning omlaag kan trekken ("of die poort of die poort genereert een laag"), bij wired-AND gaan we uit van een hoog signaalniveau en de eigenschap dat geen enkele uitgang de spanning omlaag mag trekken ("en die poort en die poort genereert een hoog").

Open drain uitgangen worden gebruikt bij het realiseren van buslijnen. Dit is te zien in figuur 4.47(b). In het poortsymbool is een ruitje met een streepje te zien. Dat betekent



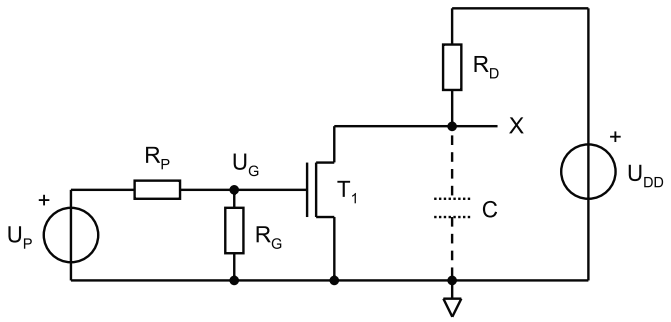
(a) *Poorten met open drain uitgangen.* **(b)** *Open drain buffers aan een buslijn.*

Figuur 4.47: *Het gebruik van open drain uitgangen.*

dat de poort een open drain uitgang heeft en dat er pull-up weerstand nodig is om een gedefinieerd spanningsniveau op de buslijn te zetten als geen van de uitgangen actief is.

Open drain uitgangen hebben een voordeel t.o.v. andere uitgangstrappen. Het is namelijk niet mogelijk om kortsluiting te veroorzaken. Meerdere uitgangen kunnen tegelijk actief zijn (transistor geleidt) en trekken zo het spanningsniveau van de buslijn omlaag. Verder is het voor een poort mogelijk om het signaalniveau te bemonsteren. Hier zijn *bidirectionele* buslijnen mee te maken, een buslijn waarmee data twee kanten op kan worden gestuurd. Er is dus datatransport mogelijk tussen meerdere poorten. Een goed voorbeeld hiervan is het I^2C-protocol [84].

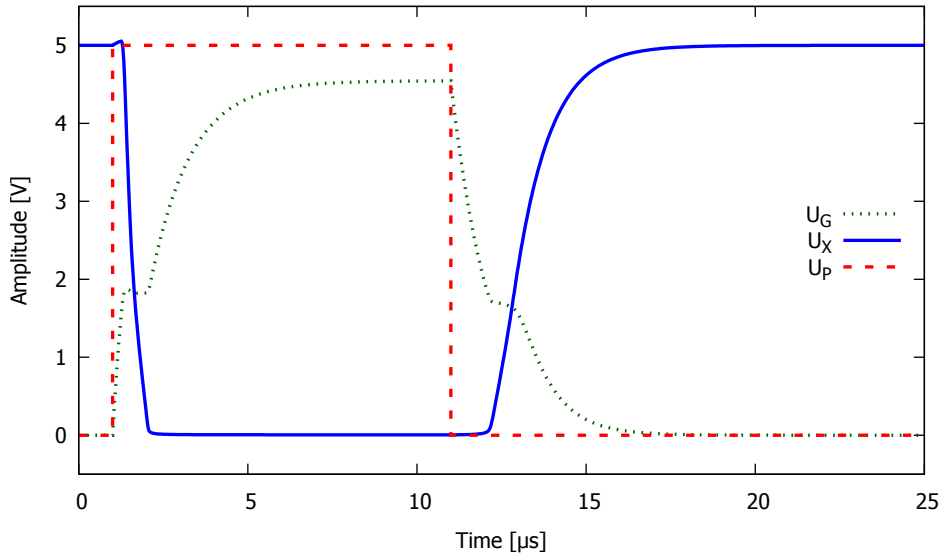
Een groot nadeel van open drain buslijnen is dat het herstel van het hoge signaalniveau langzamer gaat naar mate de capaciteit t.o.v. de referentie toeneemt. Deze capaciteit wordt gevormd door de capaciteiten tussen de drain en de source van de transistoren, van de bedrading en van ingangen van andere poorten. We demonstreren dat aan de hand van de schakeling in figuur 4.48. De schakeling bestaat uit een nMOS-transistor, twee spanningsbronnen, een drietal weerstanden en een capaciteit. De spanning van U_{DD} is 5 V. De waarde van weerstand R_D is 2,2 kΩ en de capaciteit C bedraagt 400 pF. De gate van T_1 wordt aangestuurd door spanningsbron U_P en weerstanden R_P en R_G. De waarde van R_P is 10 kΩ en van R_G 100 kΩ. Weerstand R_G zorgt voor een geleidend pad naar de referentie als spanningsbron U_P verwijderd is. De twee weerstanden vormen een spanningsdeler zodat $U_G \approx 0{,}9 \cdot U_P$. Voor de transistor is gekozen voor het type 2N7002 [85]. De gehele schakeling is gesimuleerd met LTspice XVII [86].



Figuur 4.48: *Aansturing open drain uitgang.*

De spanningsvorm van bron U_P is een puls van 5 V met een periodeduur van 10 µs. Deze puls is in figuur 4.49 getekend met een streepjeslijn. De starttijd is 1 µs na het begin. De spanning op de gate is getekend met de stippellijn. De gate gedraagt zich als een capaciteit die opgeladen wordt. Rond 1,8 V is een kleine knik in de gatespanning te zien. De drempelspanning van de gate van de transistor is namelijk ongeveer 1,8 V en de transistor gaat hier over van sperren naar geleiden (tussen drain en source) waardoor de gate-capaciteit verandert. De uitgangsspanning U_X, getekend met de ononderbroken lijn, gaat snel omlaag. Dat komt omdat de weerstand tussen drain en source klein is. De condensator C ontlaadt daardoor snel. Na ongeveer 1 µs heeft de uitgang een spanning van enkele millivolt bereikt.

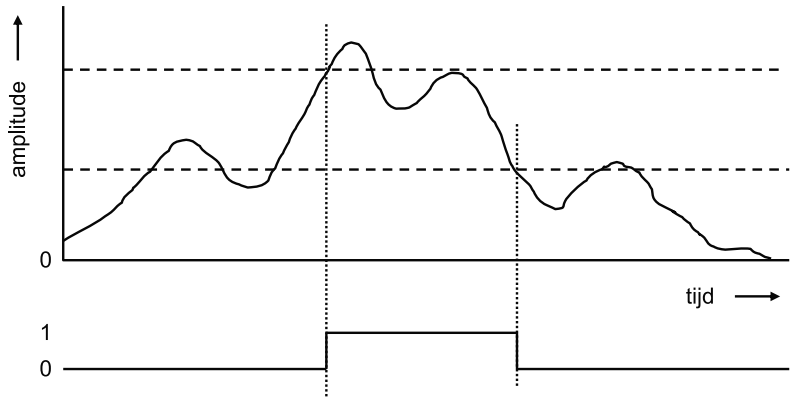
Op tijdstip 11 µs gaat de spanning van U_P omlaag naar de referentie. De gate wordt ontladen via de parallelschakeling van de weerstanden R_P en R_G. Totdat de gatespanning U_G onder de drempelspanning komt, geleidt de transistor nog steeds. De uitgang blijft nog laag. Als de gatespanning onder de drempelspanning komt, spert de transistor en wordt capaciteit C opgeladen via weerstand R_D. Het opladen zorgt voor het trage herstel van spanning op punt X.



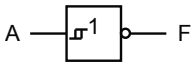
Figuur 4.49: *Spanningsverloop van de bron, de gate en de drain.*

4.9.4 Schmitt-trigger ingangen

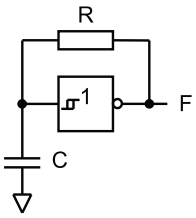
Ingangssignalen veranderen soms niet duidelijk tussen laag en hoog. Een signaal kan langzaam stijgen of dalen waardoor het voor langere tijd in het niet-gedefinieerde spanningsgebied terecht komt. Een andere oorzaak waardoor de signalen zich in het niet-gedefinieerde gebied kunnen bevinden is *overspraak* tussen signaallijnen. Het betreft hier het inwerken van een elektromagnetisch veld op de signaallijn (antennewerking). Het ingangssignaal kan hersteld worden met behulp van een *Schmitt-trigger* ingang. Bij Schmitt-trigger ingangen ligt het omslagpunt van laag naar hoog op een ander spanningsniveau dan van laag naar hoog. Dit wordt geïllustreerd in figuur 4.50(a).



(a) *Herstel van het ingangssignaal.*



(b) *Symbool NOT-poort.*



(c) *Oscillator.*

Figuur 4.50: *Werking en toepassing van een Schmitt-trigger.*

In figuur 4.50(b) is het symbool van een Schmitt-trigger NOT-poort te zien. Het *hysteresisteken*[3] geeft de Schmitt-trigger werking aan. Een aardige toepassing van een NOT-poort met Schmitt-trigger is de oscillatorschakeling in figuur 4.50(c). Bij opstarten is de condensator leeg. De spanning op de ingang is laag. De uitgangsspanning is hoog. Via de weerstand wordt de condensator langzaam opgeladen. Als de spanning over de condensator boven de drempelspanning van laag naar hoog komt, klapt de uitgang om naar laag. De condensator wordt nu ontladen totdat de spanning onder de drempelspanning van hoog naar laag komt. Al doende oscilleert de condensatorspanning tussen de twee drempelspanningen. De uitgang oscilleert mee. De frequentie is niet “hard” maar varieert met de voedingsspanning, de temperatuur en het fabricageproces.

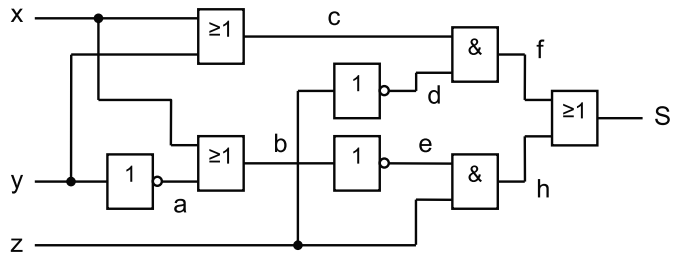
4.10 Timing van combinatoriek

Een ander aspect van een digitaal systeem betreft de timing van een schakeling. We leggen de timing van combinatoriek uit aan de hand van een voorbeeld.

In figuur 4.51 is de te onderzoeken schakeling gegeven. De schakeling heeft als ingangen x, y en z en heeft één uitgang S. Te zien is dat er diverse paden zijn van de ingangen naar de uitgang. Om een gedetailleerd timingmodel op te stellen zijn de signaalnamen a t/m h toegevoegd. Deze zijn niet strikt noodzakelijk maar helpen bij het opstellen van het timingmodel. Van de drie type poorten in de schakeling zijn de minimale en maximale vertragingstijden gegeven in tabel 4.10. De tijden zijn in nanoseconden (10^{-9} s) gegeven. Er is geen alternatief dan alle mogelijk paden langs te lopen en van elk pad de minimale en maximale vertragingstijd te bepalen. Dan kunnen we met zekerheid stellen dat we de minimale en maximale tijden hebben gevonden.

Om de paden te vinden beginnen we bij een ingang en volgen de informatiestroom door de schakeling. Als voorbeeld beginnen we bij ingang x. Dit signaal splitst zich al direct en gaat naar twee OR-poorten toe. We volgen het pad naar de bovenste OR-poort. We

[3] Hysterese of hysteresis (Grieks: ’het achterblijven’) is het verschijnsel dat het verband tussen oorzaak en gevolg niet alleen afhangt van de grootte van de oorzaak, maar ook van de richting waarin de oorzaak verandert.



Figuur 4.51: *Een schakeling.*

Tabel 4.10: *Vertragingstijden*

Poort	$t_{P(min)}$	$t_{P(max)}$
AND	2,4	6,9
OR	3,1	7,2
NOT	1,5	4,3

komen dan uit bij signaal c. We vervolgen onze weg via een AND-poort naar signaal f en komen via de laatste OR-poort uit bij uitgang S. Het afgelegde pad is dus $x \rightarrow c \rightarrow f \rightarrow S$.

We stellen eerst een lijst op met alle mogelijke paden van een ingang naar de uitgang S. In totaal zijn er zes paden mogelijk. Hieronder zijn ze weergegeven.

$$x \longrightarrow c \longrightarrow f \longrightarrow S$$
$$x \longrightarrow b \longrightarrow e \longrightarrow h \longrightarrow S$$
$$y \longrightarrow c \longrightarrow f \longrightarrow S$$
$$y \longrightarrow a \longrightarrow b \longrightarrow e \longrightarrow h \longrightarrow S$$
$$z \longrightarrow d \longrightarrow f \longrightarrow S$$
$$z \longrightarrow h \longrightarrow S$$

Na enig onderzoek vinden we het kortste en langste pad in tijd. Het kortste pad in tijd is $z \longrightarrow h \longrightarrow S$. Er wordt een AND-poort en een OR-poort gepasseerd. De vertragingstijd bedraagt:

$$t_{P(min)}(\text{schakeling}) = t_{P(min)}(\text{AND}) + t_{P(min)}(\text{OR}) = 2{,}4 + 3{,}1 = 5{,}5 \text{ ns} \tag{4.54}$$

Het langste pad in tijd is $y \longrightarrow a \longrightarrow b \longrightarrow e \longrightarrow h \longrightarrow S$. Er worden twee NOT-poorten, twee OR-poorten en een AND-poort gepasseerd. De vertragingstijd bedraagt:

$$\begin{aligned} t_{P(max)}(\text{schakeling}) &= 2 \cdot t_{P(max)}(\text{NOT}) + 2 \cdot t_{P(max)}(\text{OR}) + t_{P(max)}(\text{AND}) \\ &= 2 \cdot 4{,}3 + 2 \cdot 7{,}2 + 6{,}9 \\ &= 29{,}9 \text{ ns} \end{aligned} \tag{4.55}$$

De worst-case minimale vertragingstijd is 5,5 ns en de worst-case maximale tijd is 29,9 ns. We noemen dit de globale vertragingstijden. Veel timingvraagstukken kunnen worden opgelost met de globale timingparameters.

4.11 Timing hazards

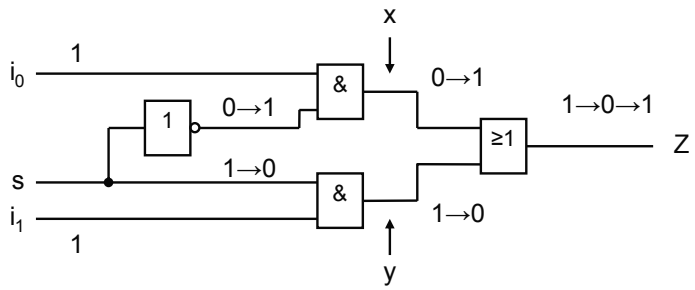
Elke poort heeft een bepaalde vertragingstijd. Het kost enige tijd voordat een verandering van een of meerdere ingangssignalen een verandering van een uitgangssignaal geeft. Als gevolg van de verschillende vertragingstijden van de poorten en de opbouw van een schakeling kunnen uitgangen kortstondig van waarde veranderen terwijl dat niet gewenst is. Dit verschijnsel wordt *hazard* genoemd. Er zijn drie typen hazards: (1) een

statische 0-hazard waarbij de uitgang tijdelijk logisch 1 wordt terwijl het 0 had moeten blijven, (2) een statische 1-hazard waarbij de uitgang tijdelijk logisch 0 wordt terwijl het 1 had moeten blijven en (3) een dynamische hazard waarbij de uitgang meer dan twee keer verandert terwijl die uitgang alleen van 0 naar 1 of van 1 naar 0 had moeten veranderen. In figuur 4.52 zijn de verschillende hazards te zien.

(a) *Statische 0-hazard.* **(b)** *Statische 1-hazard.* **(c)** *Dynamische hazards.*

Figuur 4.52: *Hazardtypen.*

We illustreren het begrip hazard aan de hand van een veelgebruikt voorbeeld: een 2x1 multiplexer. In figuur 4.53 is de multiplexer te zien. In het begin zijn alle ingangssignalen logisch 1. De waarde op signaal x (gevormd door productterm $\bar{s} \cdot i_0$) is dan logisch 0, signaal y (gevormd door productterm $s \cdot i_1$) is logisch 1 en uitgang Z is ook logisch 1. Op een bepaald moment schakelt ingang s om van 1 naar 0. Uitgang y zal na enige tijd logisch 0 worden en uitgang van de NOT-poort verandert na enige tijd van 0 naar 1. Het gevolg hiervan is dat uitgang x logisch 1 wordt maar ook dat kost enige tijd. Er is dus een moment dat zowel x als y even logisch 0 zijn waardoor uitgang Z ook een korte tijd logisch 0 is. We noemen dit een statische 1-hazard. Dit gebeurt als de functie "schakelt" tussen twee producttermen.



Figuur 4.53: *Multiplexer met statische 1-hazard.*

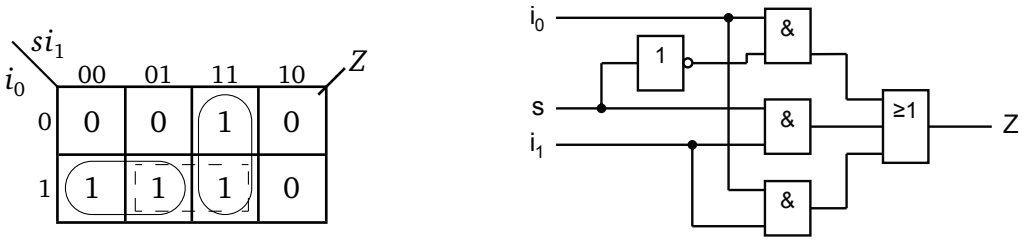
De statische 1-hazard is te vinden door het Karnaughdiagram van de functie te onderzoeken. De bovengenoemde situatie komt voor als de functie schakelt tussen minterm m_3 en m_7. Dat is de situatie waarbij de ingangen i_0 en i_1 beide logisch 1 zijn en s schakelt tussen 0 en 1. De oplossing voor het verwijderen van de hazard is om beide mintermen samen te nemen in een productterm die de andere producttermen overlapt. Dit is te zien in figuur 4.54(a). De hazardvrije schakeling in te zien in figuur 4.54(b). De extra AND-poort genereert de productterm $i_1 \cdot i_0$. Merk op dat deze productterm volgens de schakelalgebra overbodig is. We hebben in feite de consensuswet (3.19a) toepast.

De schakeling in figuur 4.53 implementeert de functie in som-van-productenvorm:

$$Z = \bar{s} \cdot i_0 + s \cdot i_1 \tag{4.56}$$

Als de schakeling wordt gerealiseerd in de product-van-sommenvorm, namelijk:

$$Z = (s + i_0) \cdot (\bar{s} + i_1) \tag{4.57}$$



(a) *Verwijderen statische 1-hazard.*

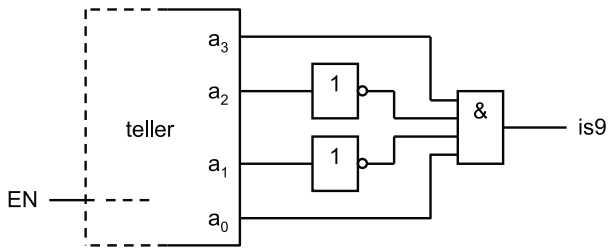
(b) *Hazardvrije multiplexer.*

Figuur 4.54: *Het verwijderen van een statische 1-hazard en een hazardvrij multiplexer.*

dan kan de uitgang tijdelijk logisch 1 worden terwijl die 0 had moeten blijven. Dit wordt een statische 0-hazard genoemd. Statische 0-hazards kunnen worden verwijderd door somtermen toe te voegen die de andere somtermen overlappen.

Het derde type, *dynamische hazards*, ontstaan door twee oorzaken. De eerste oorzaak is dat in grote schakelingen meerdere signaalpaden naar de uitgang leiden. Dit probleem kan wel worden opgelost maar kost veel logica. De tweede oorzaak is dat meerdere (ingangs-)signalen tegelijk veranderen. Dat gebeurt onder andere in schakelingen met geheugen waarbij de geheugenelementen op hetzelfde moment van waarde veranderen (bijvoorbeeld als gevolg van een centraal kloksignaal). Deze hazards worden ook wel *functiehazards* genoemd.

In figuur 4.55 is een schakeling te zien die detecteert of de teller de telstand 9 heeft bereikt. De telstand is beschikbaar via de uitgangen a_3 t/m a_0. De telstand verandert door een puls op de *EN*-ingang. Door de interne opbouw van de teller vinden de veranderingen niet tegelijkertijd plaats. Zodoende heeft uitgang *is*9 last van dynamische hazards. In feite is hier niets tegen te doen. Het beste is gewoon wachten totdat alle uitgangen een definitieve waarde hebben gekregen. Het oplossen van hazards door toevoegen van logica kost meer poorten dan strikt noodzakelijk volgens de schakelalgebra. Dit betekent meer oppervlakte op een ic, meer dissipatie en het is ook lastig om te testen of de extra logica om hazards te voorkomen correct werkt. Zonder deze extra logica functioneert de schakeling immers ook.



Figuur 4.55: *De schakeling heeft last van dynamische hazards.*

Hazards leveren problemen op bij *asynchrone sequentiële logica*. Dit is logica waarbij een of meerdere uitgang worden teruggekoppeld naar ingangen. Er zitten dus terugkoppellussen in de schakeling waardoor geheugenelementen ontstaan die direct reageren op signaalveranderingen. Grote schakelingen worden daarom in de regel door middel van *synchrone sequentiële logica* gerealiseerd. Hierin worden alle acties binnen het sys-

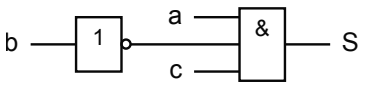
teem uitgevoerd op een gemeenschappelijk signaal, het kloksignaal. Zodoende leveren hazards geen problemen op. In hoofdstuk 6 bespreken we enkele van deze schakelingen.

Voor meer informatie over hazards, zie [87]. Een goede uitleg over hazards en het oplossen ervan wordt gegeven in [88].

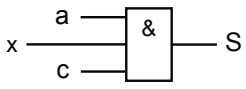
*4.12 Permissible functions

Bij grote schakelingen kan het zo zijn dat een interne logische functie bij bepaalde ingangswaarden geen invloed uitoefent op de uitgangswaarden van de schakeling. In dergelijke situaties is de uitgangswaarde van de interne functie niet van belang. Bepaalde functiewaarden kunnen dus als don't care gespecificeerd worden. Elke functie die voldoet aan deze niet-volledig gespecificeerde functie wordt een *permissible function* genoemd [89, 90]. De niet-volledig gespecificeerde functie wordt dan beschreven met een verzameling van permissible functions. Elke functie uit deze verzameling kan gebruikt worden om de interne logische functie te realiseren.

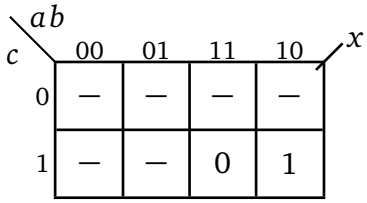
We demonstreren het opstellen van de verzameling van permissible functions aan de hand van een eenvoudig voorbeeld met een AND-poort [91]. In figuur 4.56(a) is een schakeling te zien die de functie $S = a \cdot \overline{b} \cdot c$ realiseert. De inverse van b kan gerealiseerd worden met een NOT-poort. Maar er zijn ook andere functies mogelijk. In figuur 4.56(b) is de NOT-poort vervangen door signaal x. De vraag is nu welke functies er allemaal kunnen worden aangeboden op signaal x. De functie S is logisch 0 als ingang a en/of c logisch 0 zijn. De uitgangswaarde van de NOT-poort is dan niet van belang. Alleen als a en c beide 1 zijn, is de waarde van x van belang. Dit is te zien in het Karnaughdiagram in figuur 4.56(c).



(a) *De te realiseren functie.*



(b) *Equivalente functie.*



(c) *Karnaughdiagram.*

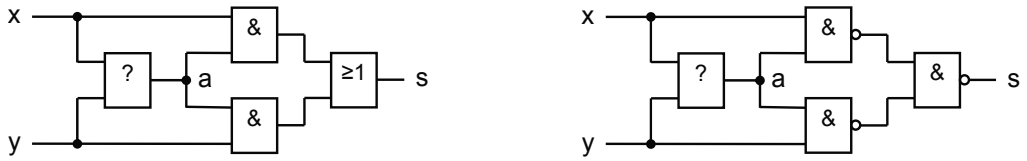
Figuur 4.56: *Permissible functions voor x.*

Om functie S te realiseren kan x onder andere zijn:

$$\begin{aligned} &\overline{b} \\ &\overline{a} + \overline{b} \\ &\overline{a} + \overline{b} + \overline{c} \end{aligned} \tag{4.58}$$

Er zijn zes don't cares dus in totaal zijn er $2^6 = 64$ verschillende functies mogelijk. Als blijkt dat een van de functies al beschikbaar is een schakeling, dan kan de NOT-poort uitgespaard worden.

We bespreken nog een voorbeeld. We willen graag een EXOR-functie realiseren volgens het schema in figuur 4.57(a). Figuur 4.57(b) is een versie met NAND-poorten. In beide schema's is een poort met onbekende functie en uitgang a opgenomen. De vraag is nu welke functie de poort moet realiseren zodat de EXOR-functie gerealiseerd wordt.



(a) *Schema met AND/OR* **(b)** *Schema met NAND/NAND.*

Figuur 4.57: *Realisatie EXOR-poort.*

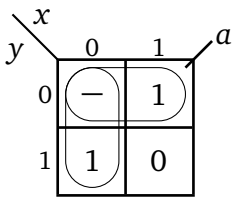
De functie van s uitgedrukt in x, y en a is:

$$s = x \cdot a + y \cdot a \tag{4.59}$$

Voor de combinatie $xy = 00$ moet de functie een logische 0 opleveren. Aangezien beide producttermen een logische 0 opleveren maakt niet zoveel uit wat de waarde van a is. We specificeren dit als een don't care. Bij $xy = 01$ is de functie logisch 1, dus a moet 1 zijn zodat $y \cdot a$ een logische 1 oplevert. Hetzelfde geldt voor $xy = 10$, nu wordt $x \cdot a$ logisch 1. Bij de combinatie $xy = 11$ moet s logisch 0 worden, a moet dan logisch 0 zijn. We kunnen de waarden voor a in een waarheidstabel plaatsen. Dit is te zien in tabel 4.11. Hierin zijn ook de waarden voor s opgenomen, de functie van de EXOR.

Tabel 4.11: *Waarheidstabel voor a.*

x	y	a	s
0	0	–	0
0	1	1	1
1	0	1	1
1	1	0	0



Figuur 4.58: *Karnaughdiagram voor functie a.*

In figuur 4.58 is het Karnaughdiagram gegeven. De meest eenvoudige functie is:

$$a = \overline{x} + \overline{y} \tag{4.60}$$

Met behulp van de stellingen van De Morgan kunnen we de functie omzetten naar:

$$a = \overline{x} + \overline{y} = \overline{x \cdot y} \tag{4.61}$$

De poort met het vraagteken is te realiseren met een NAND-poort, zodat de schakeling voor s te realiseren is met vier NAND-poorten. Zie figuur 4.57(b).

*4.13 Combinatorische schakelingen met VHDL

VHDL is een taal om digitale schakelingen te beschrijven. We geven hier slechts een korte introductie en gaan niet dieper op de mogelijkheden in anders dan het beschrijven van logische functies op poortniveau.

In listing 4.1 is de beschrijving te zien van de schakeling in figuur 4.44. De *entity* beschrijft de ingangen en uitgangen van de schakeling. De *architecture* beschrijft het gedrag. De schakeling heeft vier ingangen `m3`, `m2`, `m1` en `m0`. De uitgangen zijn `d30` en `d31`. VHDL maakt voor signaalnamen geen onderscheid tussen hoofd- en kleine letters. Alle signalen zijn van het type `std_logic` en kunnen meer signaalniveaus aannemen dan 0 en 1. Denk hierbij aan tri-state en open drain aansluitingen. Om dit type te gebruiken moet een *library* geladen worden. Dit is te zien in de eerste twee regels. Aanbevolen wordt om `std_logic` voor logische signalen te gebruiken.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity calendar is
5      port (m3, m2, m1, m0 : in std_logic;
6            d30, d31       : out std_logic;
7           );
8  end entity calendar;
9
10 architecture gates of calendar is
11 begin
12     d30 <= (m3 and m0) or (not m3 and m2 and m0);
13     d31 <= (not m3 and m0) or (m3 and not m0);
14 end architecture gates;
```

Listing 4.1: *VHDL-beschrijving van een schakeling met logische functies.*

In de architecture zijn twee twee *concurrent* signaaltoekenningen te zien. Concurrent wil zeggen gelijktijdig. Hardware werkt per definitie gelijktijdig. De volgorde van de toekenningen maakt dan ook niet uit. We kunnen de statements net zo goed verwisselen. De werking blijft hetzelfde, ook in tijdgedrag. Dat is lastig voor softwareprogrammeurs die gewend zijn dat statements sequentiëel (na elkaar) worden uitgevoerd. De beschrijving is rechttoe-rechtaan. NOT heeft een hogere prioriteit dan AND en OR. AND en OR hebben dezelfde prioriteit. De haakjes dwingen bewerkingsvolgorde af.

Verder valt op dat de taal “wollig” is. Er zijn veel taalconstructies nodig om tot een correcte beschrijving te komen.

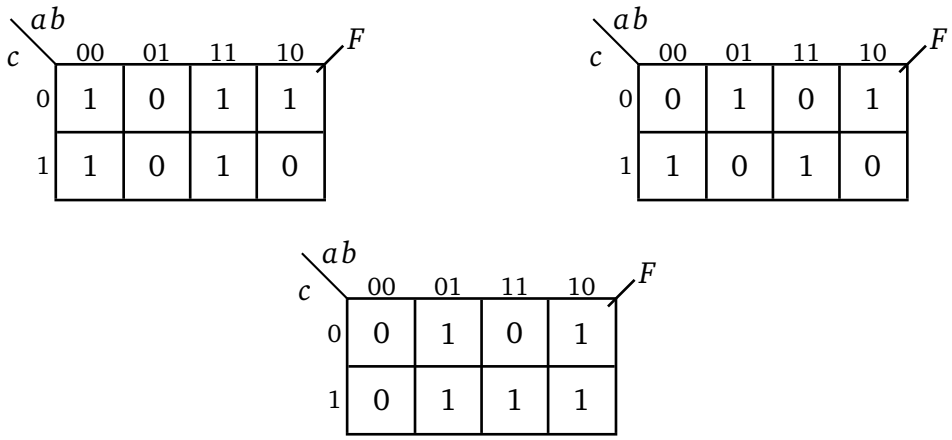
4.14 Opgaven

4.1. Minimaliseer de functies gegeven in de Karnaughdiagrammen in figuur P4.1:

4.2. Gegeven de functies:

$$\begin{aligned} f_{x,y,z} &= \sum m(0,1,2,6) \\ f_{a,b,c} &= \sum m(1,2,5,6,7) \\ f_{s_2,s_1,s_0} &= \sum m(0,3,6) + d(1,2) \end{aligned} \tag{P4.1}$$

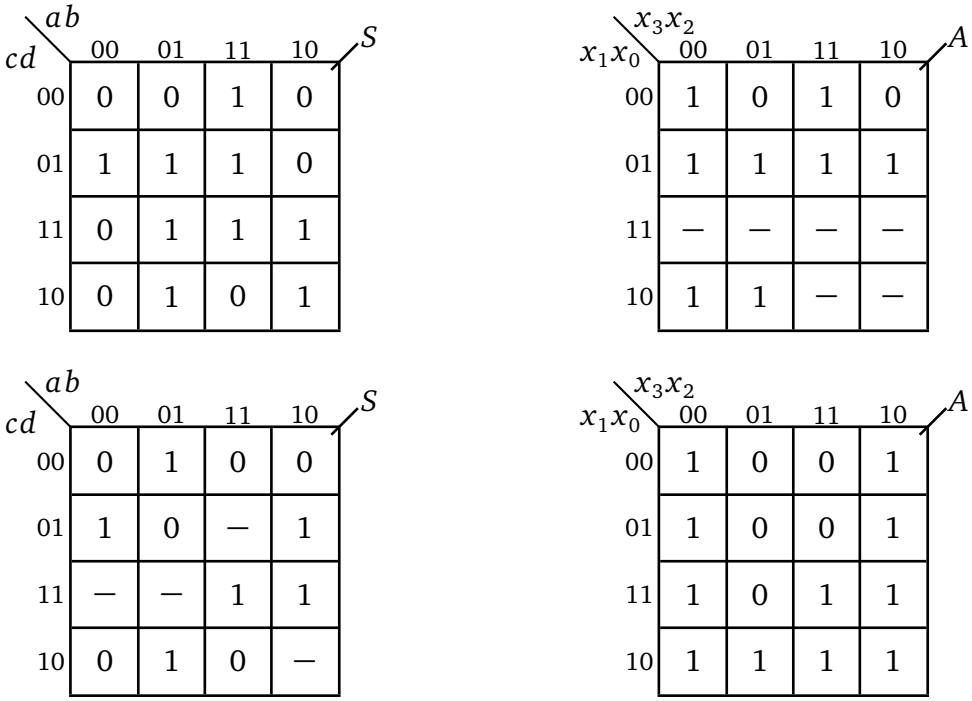
Stel de Karnaughdiagrammen op en geef de geminimaliseerde functies. Teken de bijbehorende schakelingen met NOT, AND en OR.



Figuur P4.1: *Karnaughdiagrammen met drie variabelen.*

4.3. Ontwerp een *majority gate*. Dit is een schakeling met drie ingangen en één uitgang. De uitgang is 1 als de meerderheid van de ingangen 1 is, anders is de uitgang 0.

4.4. Minimaliseer de functies gegeven in de Karnaughdiagrammen in figuur P4.2:



Figuur P4.2: *Karnaughdiagrammen met vier variabelen.*

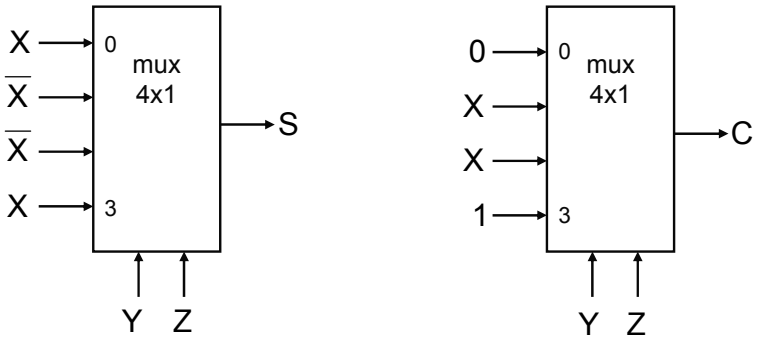
4.5. Gegeven de functies:

$$\begin{aligned} f_{w,x,y,z} &= \sum m(1,2,3,5,7,10,11,12,15) \\ f_{a,b,c,d} &= \sum m(6,7,8,12,14,15) \\ f_{s_3,s_2,s_1,s_0} &= \sum m(4,5,9,10,12) + d(7,8,11,15) \\ f_{x_3,x_2,x_1,x_0} &= \prod M(1,2,3,5,7,10) + D(11,12,15) \end{aligned} \tag{P4.2}$$

Minimaliseer met behulp van Karnaughdiagrammen naar een SOP-vorm.

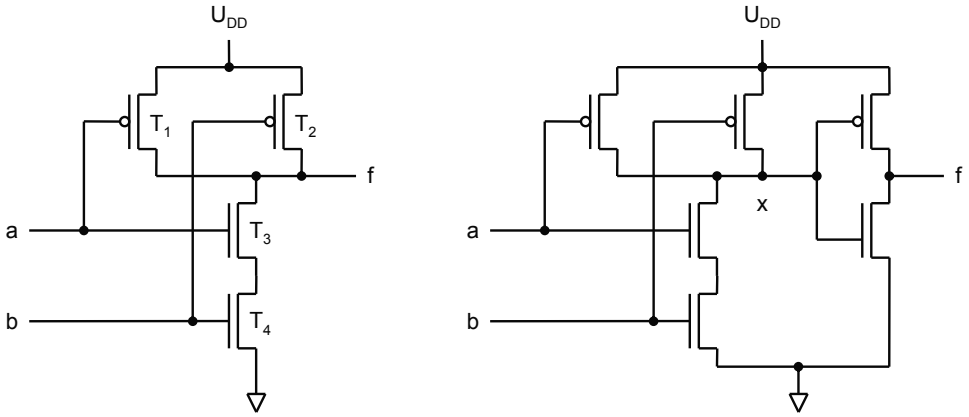
* **4.6.** Gegeven de functies in opgave 4.5. Minimaliseer met behulp van de Quine-McCluskey-methode naar een SOP-vorm.

4.7. Gegeven twee 4x1 multiplexers in figuur P4.3. Deze worden aangesloten volgens onderstaand schema. Bepaal de waarheidstabellen van S en C.



Figuur P4.3: *Logische functies met 4x1 multiplexers.*

4.8. In figuur P4.4 zijn twee schakelingen te zien met MOS-transistoren. Bepaal van elk van de schakelingen de waarheidstabel en de logische functie.



(a) *Schakeling met transistoren.* **(b)** *Schakeling met transistoren.*

Figuur P4.4: *Twee schakelingen met MOS-transistoren.*

5

Rekenschakelingen

Computers zijn bedoeld om data te verwerken. Een van de aspecten van het verwerken van data is het uitvoeren van berekeningen. Dat kunnen computers zeer snel, de huidige generatie voert een berekening in een nanoseconde uit. Daarnaast voeren ze de berekeningen foutloos uit. Het mag duidelijk zijn dat computers op dit vlak beter presteren dan mensen. In computers zitten digitale schakelingen die gebruikt worden om de berekeningen uit te voeren. Deze schakelingen maken gebruik van het binaire talstelsel. We gaan in dit hoofdstuk hier dieper op in.

We behandelen de vier elementaire rekenkundige bewerkingen: optellen, aftrekken, vermenigvuldigen en delen. Bij het optellen en delen geven we eerst voorbeelden in het decimale talstelsel om te laten zien hoe de bewerkingen verlopen. Daarna behandelen we de optelling en deling in het binaire talstelsel. De overige twee bewerkingen worden alleen in het binaire talstelsel behandeld. We gaan uit van de zogenoemde pen-en-papier-methode. Daarna ontwikkelen we schakelingen die op deze methoden berusten. Dat is echter niet altijd de meeste effectieve methode in termen van snelheid of grootte van de schakeling. We bekijken daarom ook een alternatief.

In eerste instantie gebruiken we alleen niet-negatieve gehele getallen. Dat zijn gehele getallen groter dan of gelijk aan 0. De Engelse term hiervoor is *unsigned*, getallen zonder teken. Later introduceren we het gebruik van gehele getallen met teken. De Engelse term hiervoor is *signed*. Bekend is de teken-en-grootte representatie waarbij het negatief zijn van een getal wordt wordt aangegeven met een minteken. Voor het gebruik in digitale schakelingen is dit niet de beste keuze. We introduceren dan ook de two's complement representatie die veel gebruikt wordt in digitale systemen en computers.

Het afbeelden van binaire getallen als decimale getallen is een lastige aangelegenheid. Het gebruik van de BCD-code kan de problemen enigszins oplossen. We bespreken daarom de BCD-opteller voor het optellen twee BCD-gecodeerde cijfers. We bespreken kort de ten's complement representatie. Dat is de decimale variant van de binaire two's complement representatie. De ten's complement representatie wordt gebruik bij het rekenen met negatieve BCD-getallen. Als laatste bespreken we de fixed point en floating

point representaties waarmee getallen met een komma kunnen worden weergegeven.

5.1 Optellen in het decimale talstelsel

Optellen in het decimale talstelsel gaat heel eenvoudig. We maken gebruik van de positie van de afzonderlijke cijfers van de op te tellen getallen en hanteren een aantal regels voor het optellen van cijfers. We bekijken eerst het optellen van twee decimale cijfers, daarna gaan we grotere getallen optellen. Voor de goede orde: in de operatie $S = A + B$ heet S de som, A het opteltal en B de opteller. We zullen de laatste twee termen niet zoveel gebruiken.

In figuur 5.1 is een aantal voorbeelden gegeven van het optellen van twee decimale cijfers. De optellingen $0+0$ en $4+5$ leveren als antwoord respectievelijk 0 en 9 op. Dit cijfer wordt het *somcijfer* genoemd. We noteren het cijfer onder de streep. Bij de drie overige optellingen is het resultaat te groot om met één cijfer weer te geven. Zo levert de optelling $6+7$ als uitkomst 13 op. We noemen de 3 weer het somcijfer en de 1 wordt het *overloopcijfer* (Engels: carry) genoemd. Het overloopcijfer geeft dus aan dat het resultaat van de optelling niet in een kolom past.

$$\begin{array}{r} 0 \\ 0 \\ \hline 0 \end{array} + \qquad \begin{array}{r} 4 \\ 5 \\ \hline 9 \end{array} + \qquad \begin{array}{r} 5 \\ 5 \\ \hline 10 \end{array} + \qquad \begin{array}{r} 6 \\ 7 \\ \hline 13 \end{array} + \qquad \begin{array}{r} 9 \\ 9 \\ \hline 18 \end{array} +$$

Figuur 5.1: *Enkele voorbeelden van het optellen van twee decimale cijfers.*

We kunnen gebruik maken van de kolomsgewijze optelling om grotere getallen op te tellen. Als voorbeeld tellen we de twee getallen 8649 en 5739 bij elkaar op, zie figuur 5.2. We schrijven de getallen onder elkaar zodat cijfers met dezelfde positie (of gewicht) recht onder elkaar staan. Daarna tellen we per kolom op. We beginnen met de eenheden, in dit geval de twee negens in de meest rechtse kolom. We voeren de optelling $9+9$ uit met als resultaat 18. We noteren het somcijfer 8 in de kolom onder de streep. Het overloopcijfer 1 stelt een tiental voor en wordt boven de aangrenzende linker kolom geplaatst. Dit is de kolom met de tientallen. In deze kolom wordt de optelling $1+4+3$ uitgevoerd en dat levert als resultaat 8 op. Het resultaat past, er is geen overloop (beter: het overloopcijfer is 0) zodat bij de aangrenzende linker kolom slechts twee cijfers moeten worden opgeteld. Daarna gaan we verder met de honderdtallen en duizendtallen. Het optellen van de duizendtallen resulteert ook in een overloop. Er zijn echter geen tienduizendtallen zodat het overloopcijfer direct als meest significante cijfer genoteerd wordt.

$$\begin{array}{r} 8649 \\ 5739 \\ \hline \end{array} + \qquad \begin{array}{|c|c|c|c|c|c|} \hline 1 & 1 & & 1 & & \\ \hline & 8 & 6 & 4 & 9 & \\ \hline & 5 & 7 & 3 & 9 & + \\ \hline 1 & \not{1}4 & \not{1}3 & 8 & \not{1}8 & \\ \hline \end{array}$$

Figuur 5.2: *Voorbeeld van het optellen van twee decimale getallen.*

We zien dat het optellen van twee getallen gebaseerd is op het herhalen van dezelfde procedure: optellen van de cijfers in de kolom, noteren van het somcijfer in de kolom

en noteren van het overloopcijfer boven de aangrenzende linker kolom. De lege plekken mogen opgevuld worden met nullen zodat de kolommen een identiek formaat hebben. Het somcijfer ligt in het bereik van 0 t/m 9 en het overloopcijfer is 0 of 1.

5.2 Optellen in het binaire talstelsel

Optellen in het binaire talstelsel werkt op dezelfde wijze als in het decimale talstelsel, maar nu worden alleen de cijfers 0 en 1 gebruikt. De algemeen gangbare naam voor een binair cijfer is *bit*, een samentrekking van de Engelse woorden *binary digit*. Daarnaast zullen we het meer gebruikelijke woord *carry* gebruiken in plaats van overloop.

Laten we beginnen met twee bits op te tellen. We kunnen makkelijk alle mogelijkheden uitschrijven. Er zijn er maar vier. Deze zijn te zien in figuur 5.3. De optelling $0+0$ levert als sombit een 0 op. De optellingen $0+1$ en $1+0$ leveren als sombit een 1 op. Bij deze drie optellingen past het resultaat in één bit, er is geen carry. De optelling $1+1$ geeft als resultaat het binaire getal 10. Hierin is de 0 weer de sombit en de 1 de carrybit. De carry heeft hier dezelfde functie als bij een decimale optelling. Het geeft aan dat het resultaat van de optelling niet in een kolom past.

$$\begin{array}{r} 0 \\ 0 \\ \hline 0 \end{array} + \qquad \begin{array}{r} 0 \\ 1 \\ \hline 1 \end{array} + \qquad \begin{array}{r} 1 \\ 0 \\ \hline 1 \end{array} + \qquad \begin{array}{r} 1 \\ 1 \\ \hline 10 \end{array} +$$

Figuur 5.3: *Alle mogelijke optellingen van twee bits.*

Het optellen van twee binaire getallen verloopt hetzelfde als in het decimale talstelsel. Als voorbeeld tellen we de getallen 01101100 en 01011010 bij elkaar op, zie figuur 5.4. We zetten de bits met dezelfde positie (of gewicht) onder elkaar. Daarna tellen we per kolom op. Merk op dat we nu spreken van eenheden, tweetallen, viertallen, achttallen, zestientallen etc. omdat het grondtal 2 is. We beginnen met de minst significante bits, de twee nullen aan de rechterkant. In dezelfde kolom is de (inkomende) carrybit 0. In feite is deze bit niet nodig, maar het levert een identiek formaat van de kolommen op.

	128	64	32	16	8	4	2	1	
	1	1	1	1	0	0	0	0	
01101100	0	1	1	0	1	1	0	0	
01011010 +	0	1	0	1	1	0	1	0	+
	~~0~~ 1	~~1~~ 1	~~1~~ 0	~~1~~ 0	~~1~~ 0	~~0~~ 1	~~0~~ 1	~~0~~ 0	

Figuur 5.4: *Voorbeeld van het optellen van twee binaire getallen.*

Uit de optelling $0+0+0$ volgt dat de sombit en carrybit beide 0 zijn. De sombit wordt in de kolom genoteerd en de carrybit wordt boven de aangrenzende linker kolom geplaatst, op de positie van de tweetallen. Uit de tweede kolom (vanaf de rechterkant gezien) volgt de optelling $0+0+1$ met als sombit een 1 en als carrybit een 0. Het resultaat past zodat een 1 (sombit) in de kolom wordt geplaatst en een 0 (carrybit) boven de aangrenzende

linker kolom, op de positie van de viertallen. Daarna volgt de optelling $0+1+0$. Deze optelling heeft hetzelfde resultaat als de eerdere optelling van $0+0+1$. De optelling $0+1+1$ bij de viertallen levert als sombit een 0 en als carrybit een 1. De sombit wordt in de kolom geplaatst en de 1 van de carrybit wordt boven de aangrenzende linker kolom geplaatst. De optellingen $1+0+1$ en $1+1+0$ leveren hetzelfde resultaat op als de optelling $0+1+1$. De optelling $1+1+1$ levert als sombit een 1 en als carrybit een 1. Dit is tevens het grootste resultaat dat mogelijk is met drie bits.

We zien dat het optellen van twee getallen gebaseerd is op het herhalen van dezelfde procedure: optellen van de bits in de kolom, noteren van de sombit in dezelfde kolom en noteren van de carrybit boven de aangrenzende linker kolom. De lege plekken mogen opgevuld worden met nullen zodat de kolommen een identiek formaat hebben.

Per kolom worden drie bits opgeteld. Er zijn dus acht mogelijke optellingen. Deze zijn te zien in figuur 5.5. Uit de optellingen volgen een sombit en een carrybit.

$$\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\hline
00 & 01 & 01 & 10 & 01 & 10 & 10 & 11
\end{array}$$

Figuur 5.5: *Alle mogelijke optellingen met drie bits.*

In het onderstaande voorbeeld (figuur 5.6) worden twee optellingen met acht bits uitgevoerd. De optelling aan de linkerkant levert een antwoord van acht bits op, even veel bits als beide op te tellen getallen. De optelling in het midden levert een antwoord van negen bits op, één bit meer dan het oorspronkelijke aantal bits. Er is dan sprake van een *uitgaande carry*. We bedoelen hiermee dat de uitgaande carry 1 is. In de literatuur wordt ook wel gesproken van een *unsigned overflow*. Algemeen gesproken leidt het optellen van twee n-bits getallen tot een antwoord met $n+1$ bits. In een microprocessor die met vaste eenheden gewerkt zoals 8, 16 en 32 bits, wordt de uitgaande carrybit opgeslagen in de *carry flag*. Een computerprogramma kan deze flag testen om te bepalen of het antwoord past.

$$\begin{array}{lll}
11111000 & 1)\ 11111100 & \text{carry's} \\
00110101 & 11110010 & \text{getal A} \\
01011100 \quad + & 11001110 \quad + & \text{getal B} \\
\hline
10010001 & 1)\ 11000000 & \text{resultaat}
\end{array}$$

Figuur 5.6: *Twee optellingen waarvan er bij één overflow optreedt.*

5.3 Ontwerp van een opteller voor twee binaire getallen

We willen nu een schakeling ontwerpen voor het optellen van twee binaire getallen. Hierbij gaan we uit van de besproken methode voor het optellen van getallen. De methode bestaat uit het herhaald uitvoeren van een optelling van twee of drie bits. Voor deze

optellingen ontwerpen we schakelingen die we kunnen gebruiken in het opbouwen van een opteller voor twee binaire getallen.

Nu werkt een digitale schakeling met logische nullen en enen. We moeten dus een vertaling maken van rekenkundige nullen en enen naar logische nullen en enen. Uiteraard nemen we de voor de hand liggende omzetting:

$$\text{numeriek } 0 \longleftrightarrow \text{logische } 0$$

$$\text{numeriek } 1 \longleftrightarrow \text{logische } 1$$

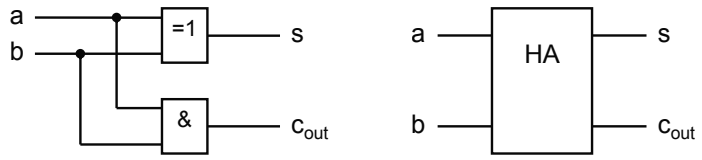
We kunnen in de waarheidstabellen en functies nu de logische nullen en enen gebruiken.

5.3.1 Ontwerp van een opteller voor twee bits

We beginnen met het ontwerpen van een opteller voor twee bits. Deze optellingen zijn te zien in figuur 5.3. De schakeling heeft twee ingangen van de op te tellen bits die we a en b noemen. De uitgangen zijn s voor de sombit (Engels: sum) en c_{out} voor de carrybit.

De waarheidstabel volgt direct uit de eerder gegeven optellingen van twee bits, zie figuur 5.7. De functies zijn eenvoudig van aard. De carry is te realiseren met een AND-poort en de som is te realiseren met een EXOR-poort. In het midden van de figuur is het schema te zien. Geheel rechts is het bloksymbool te zien. De opteller voor twee bits wordt een *half adder* genoemd omdat deze schakeling slechts twee bits bij elkaar optelt en niet drie zoals we in de volgende paragraaf zullen zien.

a	b	c_{out}	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

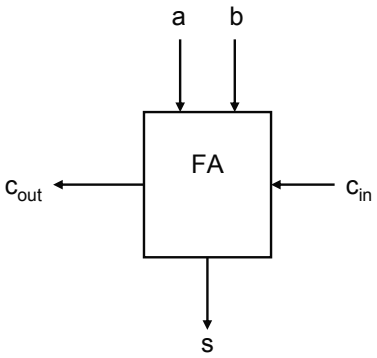


Figuur 5.7: *Waarheidstabel, schema en bloksymbool van een half adder.*

5.3.2 Ontwerp van een opteller voor drie bits

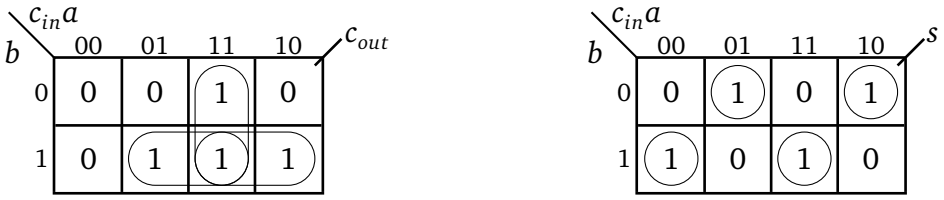
We hebben gezien dat per kolom drie bits moeten worden opgeteld, twee bits van de beide getallen en de inkomende carry van de rechts aangrenzende kolom. Het antwoord bestaat uit een sombit en een uitgaande carrybit. We gaan uit van de acht mogelijke optellingen zoals zijn gegeven in figuur 5.5. We kunnen deze optellingen omzetten in een waarheidstabel. De te realiseren schakeling wordt een *full adder* genoemd omdat deze schakeling drie bits bij elkaar optelt en niet twee zoals we in de vorige paragraaf hebben gezien. In figuur 5.8 zijn het bloksymbool en de waarheidstabel van de full adder gegeven. Hierin zijn a en b de bits van de twee op te tellen getallen en c_{in} de *inkomende carrybit*. De uitgang s is de sombit en c_{out} is de *uitgaande carrybit*.

Met behulp van Karnaughdiagrammen minimaliseren we de functies. Dit is te zien in figuur 5.9.



c_{in}	a	b	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figuur 5.8: *Bloksymbool en waarheidstabel van een 1-bit full adder.*



Figuur 5.9: *Karnaughdiagram voor c_{out} en s.*

Na uitwerking van de Karnaughdiagrammen volgen de functies in SOP-vorm:

$$\begin{aligned} c_{out} &= a \cdot b + a \cdot c_{in} + b \cdot c_{in} \\ s &= \overline{a} \cdot \overline{b} \cdot c_{in} + \overline{a} \cdot b \cdot \overline{c_{in}} + a \cdot \overline{b} \cdot \overline{c_{in}} + a \cdot b \cdot c_{in} \end{aligned} \tag{5.1}$$

Het blijkt dat de functie voor s niet te vereenvoudigen is, maar we kunnen de functie wel anders opschrijven. Daarvoor maken we gebruik van de waarheidstabel. Uit de eerste vier functiewaarden blijkt dat s een EXOR-functie is van a en b als $c_{in} = 0$. De laatste vier functiewaarden is een EXNOR-functie van a en b als $c_{in} = 1$. We kunnen de functie dus ook schrijven als:

$$s = \overline{c_{in}} \cdot (a \oplus b) + c_{in} \cdot (\overline{a \oplus b}) \tag{5.2}$$

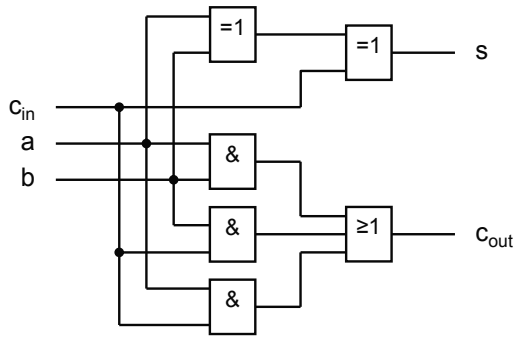
Hierin herkennen we weer een EXOR-functie:

$$\begin{aligned} s &= \overline{c_{in}} \cdot (a \oplus b) + c_{in} \cdot (\overline{a \oplus b}) \\ &= c_{in} \oplus (a \oplus b) \\ &= c_{in} \oplus a \oplus b \end{aligned} \tag{5.3}$$

De functie voor s is dus te schrijven als een EXOR van a, b en c_{in}. In figuur 5.10 is het schema van de full adder te zien. Hierin is de functie voor s gerealiseerd met twee EXOR-poorten met twee ingangen. De functie voor c_{out} volgt direct uit functie (5.1).

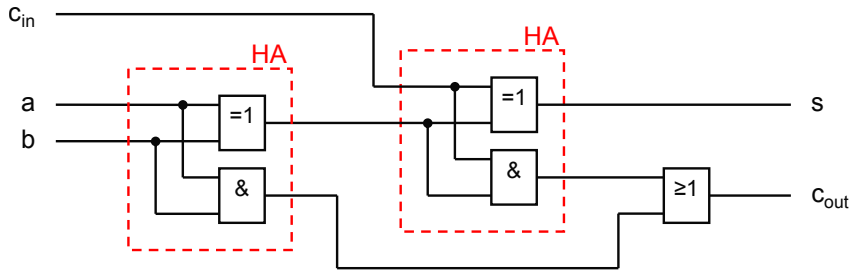
De realisatie kost zes poorten en dertien ingangen. De vertraging bedraagt twee poortvertragingen. De functie van c_{out} kan ook nog anders worden geschreven:

$$\begin{aligned} c_{out} &= a \cdot b + a \cdot c_{in} + b \cdot c_{in} \\ &= a \cdot b + c_{in} \cdot (a \oplus b) \end{aligned} \tag{5.4}$$



Figuur 5.10: *Schakeling voor een 1-bit full adder.*

Realisatie vanuit deze functies is te zien in figuur 5.11. Het blijkt dat de full adder op te bouwen is uit twee half adders en een OR-poort. Deze realisatie kost vijf poorten en tien ingangen. De vertraging kost (maximaal) drie poortvertragingen.



Figuur 5.11: *Alternatieve schakeling voor een 1-bit full adder.*

De laatste realisatie is gunstig in termen van chipoppervlak en dissipatie maar ongunstig in termen van vertraging.

5.3.3 Ontwerp van een 4-bits opteller voor binaire getallen

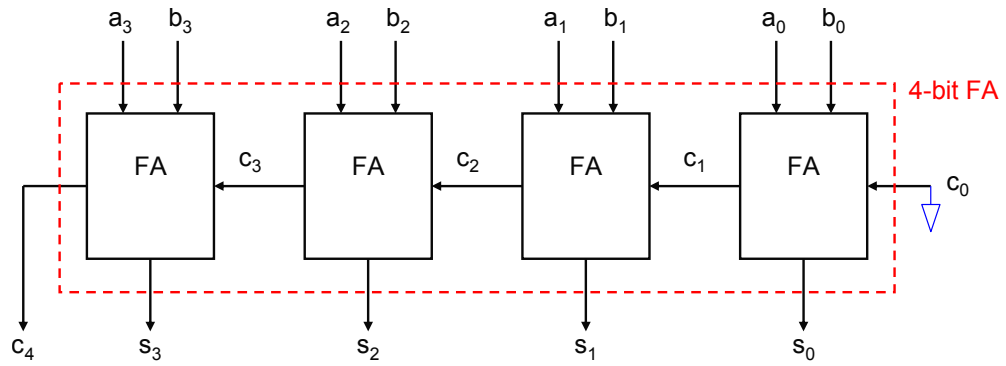
Een 4-bits opteller[1] wordt opgebouwd uit een serieschakeling van 1-bit full adders. We schrijven de getallen A en B hiertoe in hun afzonderlijke bits: getal $A = a_3a_2a_1a_0$ en getal $B = b_3b_2b_1b_0$. De subscriptnummers komen overeen met de posities van de afzonderlijke bits en geven ook de exponent van het gewicht aan ($a_3 \rightarrow 2^3$, etc.).

Het blokschema is te zien in figuur 5.12. De naamgeving van c_{in} en c_{out} verandert: de inkomende carrybit krijgt hetzelfde nummer als de a- en b-bits, de uitgaande carrybit krijgt één hoger. Zo geldt voor de full adder voor a_1 en $b_1 \longrightarrow c_{in} = c_1$, $c_{out} = c_2$.

De ingaande carrybit c_0 wordt verbonden met een logische 0. Dat is hier weergegeven met een verbinding met de referentie. Opgemerkt moet worden dat de carrybit c_4 ook gezien kan worden als sombit s_4 afhankelijk van de interpretatie van het resultaat. We kunnen s_4 namelijk schrijven als de optelling van de carrybit en twee nullen: $s_4 = c_4 + 0 + 0$.

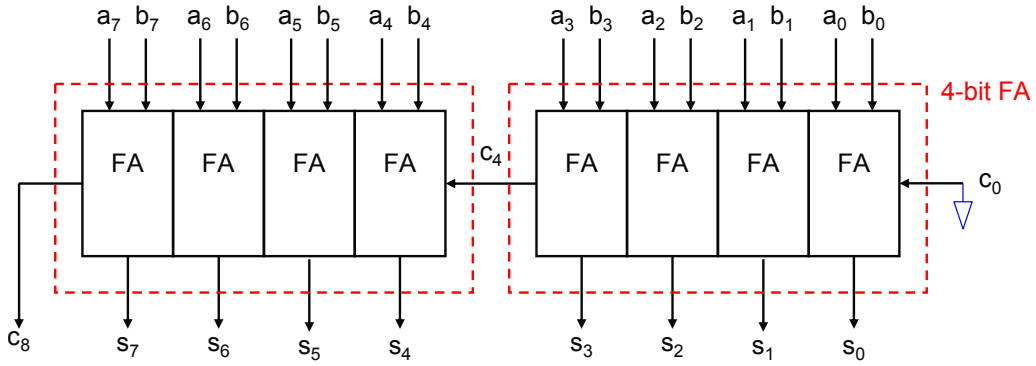
Ook voor de eenheden wordt een full adder gebruikt. Op deze manier zijn modulaire optellers te realiseren in eenheden van vier bits. Om bijvoorbeeld een 8-bits opteller

[1] We kunnen natuurlijk ook een 8-bits of 32-bits opteller bespreken. De keuze voor 4 bits is simpel: het schema past dan mooi op de pagina en de schakeling kan dan gebruikt worden bij de BCD-opteller.



Figuur 5.12: *Schakeling voor een 4-bits full adder.*

te realiseren moet de uitgaande carry c_4 van de minst significante vier bits verbonden worden met de inkomende carry c_0 van de meest significante vier bits. De inkomende carry c_0 van de minst significante vier bits moet verbonden worden met een logische 0. Een schakelschema is te zien in figuur 5.13.



Figuur 5.13: *Schakeling voor een 8-bits full adder. De interne carry's zijn niet te zien.*

Het voordeel van deze realisatie is dat er maar één logische schakeling hoeft worden te ontworpen en het systeem is eenvoudig uitbreidbaar. Het nadeel van deze realisatie is dat het lang duurt om de juiste waarde voor de uitgaande carrybit c_4 te krijgen. Na het instellen van de getallen A en B en carrybit c_0, kost het enige tijd voordat c_4 beschikbaar is. Dit wordt veroorzaakt door het transport van de uitgaande carry's naar de aangrenzende linker full adder. Deze realisatie wordt een *ripple carry adder* genoemd.[2].

Deze vertraging heeft geleid tot een scala aan andere implementaties: carry lookahead, carry-select, carry-skip, carry-completion. Deze implementaties zijn allemaal bedoeld om het berekenen van de carry's (en de sombits) te versnellen. In paragraaf 5.23 wordt de carry lookahead opteller besproken.

5.3.4 Vermeerderen van een getal met 1

Een speciaal geval van optellen is het vermeerderen van een getal met 1. Dit komt onder andere voor bij *tellers*. De bewerking tellen komt in veel applicaties voor. Meestal betreft

[2] Het Engelse werkwoord To ripple betekent kabbelen. De carry's kabbelen als het ware door de full adder secties heen.

het toepassingen waarbij wordt bijgehouden hoeveel keer een bepaalde gebeurtenis optreedt.

Voor het vermeerderen van een getal met 1 kan natuurlijk de eerder ontwikkelde optelschakeling worden gebruikt. We bieden dan op de B-ingangen het getal 0001 aan. We zien dat bit $b_0 = 1$ en dat de overige bits van B 0 zijn. De inkomende carry c_0 is ook 0. In figuur 5.14(a) is een optelling met deze opzet te zien. We vermeerderen het getal 1011_2 met 1 met als resultaat 1100_2.

$$\begin{array}{r} 0110 \\ 1011 \\ 0001 \quad + \\ \hline 1100 \end{array} \qquad\qquad \begin{array}{r} 0111 \\ 1011 \\ 0000 \quad + \\ \hline 1100 \end{array}$$

(a) *Vermeerderen met $c_0 = 0$ en met $b_0 = 1$.* **(b)** *Verwisselen van c_0 en b_0.*

Figuur 5.14: *Twee varianten om een getal te vermeerderen met 1.*

Figuur 5.14(b) toont een andere opzet. Hier zijn de inkomende carry c_0 en bit b_0 verwisseld. Dat mag op grond van de gelijkheid $c_0 + a_0 + b_0 = b_0 + a_0 + c_0$. We zien nu dat $B = 0000$ is en $c_0 = 1$. Dat betekent dat alle B-ingangen 0 zijn. We kunnen de full adder schakeling aanzienlijk vereenvoudigen. In tabel 5.1(a) is de waarheidstabel van de full adder te zien. We schrappen alle rijen waarin $b = 1$, alleen de regels met $b = 0$ blijven over. Aangezien b altijd 0 is, schrappen we de kolom b. Wat overblijft is een waarheidstabel met alleen de ingangen a en c_{in}. Dit is te zien in figuur 5.1(b). We herkennen hier de waarheidstabel van een half adder in.

Tabel 5.1: *Waarheidstabellen voor full adder en half adder.*

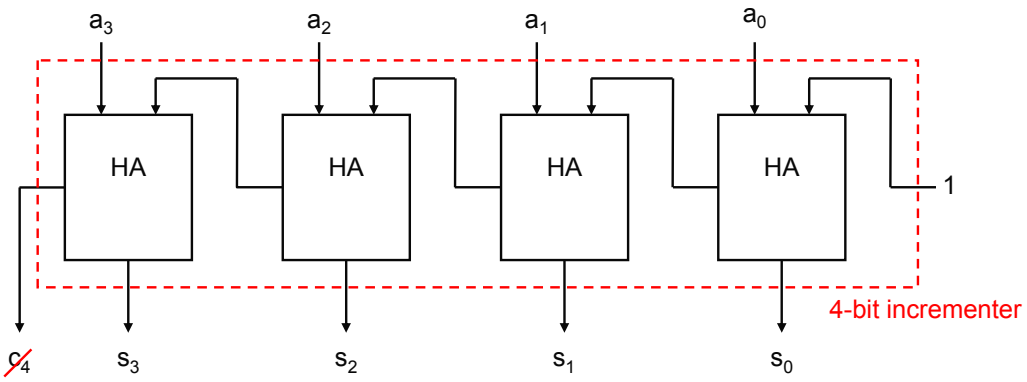
(a) *Waarheidstabel voor full adder.*

c_{in}	a	b	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(b) *Waarheidstabel voor half adder.*

c_{in}	a	c_{out}	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

In figuur 5.15 is het blokschema van de 4-bits *incrementer* te zien. Deze is opgebouwd uit een serieschakeling van half adders. De uitgaande carry c_4 wordt doorgaans niet gebruikt, maar kan gebruikt worden als carry naar de volgende half adder of als sombit s_4. Merk op dat de inkomende carry op 1 gezet moet worden voor de juiste werking. Als de inkomende carry op 0 gezet wordt, telt de incrementer nul bij A op. We kunnen de inkomende carry dus als stuursignaal gebruiken om de waarde van A met één te vermeerderen óf om de waarde van A ongewijzigd door te geven.



Figuur 5.15: *Schakeling voor een 4-bits incrementer.*

*5.4 Aftrekken in het binaire talstelsel

Aftrekken in het binaire stelsel is vergelijkbaar met aftrekken in het decimale stelsel. In de operatie $D = A - B$ wordt D het verschil (Eng: difference), A het aftrektal en B de aftrekker genoemd (dit geldt ook voor de individuele bits). In eerste instantie gaan we ervan uit dat het aftrektal, de aftrekker en het verschil groter zijn dan of gelijk zijn aan 0, dus $A \geq 0$, $B \geq 0$ en $D \geq 0$.

We bekijken eerst het aftrekken van twee bits. Er zijn maar vier mogelijkheden waarvan er drie direct uitgevoerd kunnen worden:

$0 - 0 = 0$
$1 - 0 = 1$
$1 - 1 = 0$
$0 - 1 = ?$

De laatste mogelijkheid, $0 - 1$, kan niet direct. De oplossing is om te *lenen* van de links gelegen kolom. Dit wordt duidelijk als we een aftrekking van twee 4-bits getallen bekijken. Zie figuur 5.16.

		8	4	2	1	
				$\not{1}^{0}$	$\not{0}^{10}$	
1110		1	1			
1001	−	1	0	0	1	−
		0	1	0	1	

Figuur 5.16: *Voorbeeld van het aftrekken van twee binaire getallen.*

In de kolom van de eenheden is de aftrekking $0 - 1$ niet direct te realiseren. Er moet geleend worden van de direct naastgelegen linkerkolom, de tweetallen. We lenen één tweetal oftewel twee eenheden. Nu kan de aftrekking van de eenheden wel uitgevoerd worden als $10 - 1 = 1$. De 10 stelt hier het geleende tweetal voor, uiteraard in binaire notatie. Omdat er geleend is bij de tweetallen moet dit aftrektal met 1 verlaagd worden wat resulteert in de aftrekking $0 - 0$. De aftrekkingen bij de viertallen en achttallen kunnen direct uitgevoerd worden.

In figuur 5.17 is nog een aftrekking te zien. Ook hier moet geleend worden bij de tweetallen om de aftrekking bij de eenheden uit te voeren. Eigenlijk kan er niet geleend worden bij de tweetallen want het aftrektal bij de tweetallen is 0. Voor de aftrekking bij de tweetallen moet daarom geleend worden bij de viertallen. Het aftrektal bij de tweetallen wordt dan 10 maar omdat er één geleend is aan de eenheden moet het aftrektal met één verlaagd worden. Nu kan de aftrekking voor de tweetallen uitgevoerd worden, $1-1$ levert 0 op. Er is geleend bij de viertallen maar ook dat kan eigenlijk niet, het aftrektal is namelijk 0. De viertallen moeten lenen bij de achttallen. Er is één geleend aan de tweetallen dus het aftrektal wordt 1. De aftrekking van de viertallen kan nu uitgevoerd worden en levert 0 op. De achttallen hebben er één geleend aan de viertallen, het aftrektal wordt met één verlaagd naar 0. De aftrekking $0-0$ levert als antwoord 0 op.

	$\not{1}^{0}$	$\not{0}^{1}$	$\not{0}^{1}$	$\not{0}^{10}$	
1000	1	0	0	0	
0111 −	0	1	1	1	−
	0	0	0	1	

Figuur 5.17: *Voorbeeld van het aftrekken van twee binaire getallen.*

Het concept van lenen kan inzichtelijk worden gemaakt door boven een kolom een 1 te plaatsen als er één geleend wordt aan de rechter buurkolom. Deze 1 moet dan nog van het resultaat van een aftrekking $a-b$ worden afgetrokken. In figuur 5.18 is een aftrekking van twee 8-bits getallen te zien.

	128	64	32	16	8	4	2	1	
	1	1	1	0	0	0	1		
10001110	1	0	0	0	1	1	1	0	
01111001 −	0	1	1	1	1	0	0	1	−
	0	0	0	1	0	1	0	1	

Figuur 5.18: *Voorbeeld van het aftrekken van twee 8-bits binaire getallen.*

De aftrekking bij de eenheden kan niet in één keer, er moet worden geleend bij de tweetallen. Nu kan de aftrekking wel uitgevoerd worden met als resultaat $10+0-1=1$. De 10 komt van het lenen van de tweetallen. Merk op dat we het aftrektal hebben laten staan. Om aan te geven dat er één geleend is, plaatsen we boven de kolom van de tweetallen een 1.

Vervolgens voeren we de aftrekking van de tweetallen uit. Het aftrektal is 1 en de aftrekker is 0. Het resultaat van $1-0$ is 1. Maar er is geleend aan de eenheden. Er moet dus nog 1 worden afgetrokken van het aftrektal. We krijgen nu de aftrekking $1-0-1$ waarbij de laatste 1 aangeeft dat er aan de eenheden geleend is. Het resultaat is 0.

De viertallen hebben niet geleend aan de tweetallen, we plaatsen een 0 boven de viertallen. De aftrekking $1-0-0$ levert 1 op. Er wordt niet geleend van de achttallen, dus boven de kolom van de achttallen wordt een 0 geplaatst. De aftrekking bij de achttallen gaat ook direct: $1-1-0=0$. Ook hier wordt niet geleend, dus boven de zestientallen wordt een 0 geplaatst.

Bij de zestientallen moet de aftrekking $0-1-0$ worden uitgevoerd. Dat kan niet direct, er moet geleend worden bij de tweeëndertigtallen. We voeren de aftrekking $10+0-1-0$ (de 10 komt van het lenen) uit met als resultaat 1. We plaatsen de 1 onder de streep en een 1 in de kolom van de tweeëndertigtallen, er is immers geleend aan de zestientallen.

Bij de tweeëndertigtallen moet nu de aftrekking $0-1-1$ uitgevoerd worden. Dit gaat ook niet direct en moet er geleend worden bij de vierenzestigtallen. De aftrekking wordt nu $10+0-1-1$. Het resultaat is 0 en wordt geplaatst onder de streep. Boven de kolom van de vierenzestigtallen wordt een 1 geplaatst. Bij de vierenzestigtallen wordt de aftrekking $1-0-1=0$ uitgevoerd. Dit geldt (toevallig) ook voor de van de honderdachtentwintigtallen[3].

In de voorgaande voorbeelden is het aftrektal groter dan de aftrekker. Het resultaat is altijd positief. Laten we eens kijken wat er gebeurt als de aftrekker groter is dan het aftrektal. Zie figuur 5.19. Bij de eenheden, tweetallen en viertallen gaat de aftrekking direct. Bij de achttallen gaat dat niet. Er moet één geleend worden van de (niet bestaande) zestientallen. Het resultaat van de 4-bits aftrekking is +9 en is uiteraard niet correct. Dat is te zien aan de leen in kolom van de zestientallen[4]. Deze leen signaleert dat er een *underflow* heeft plaatsgevonden.

			1)	0	0	0			
3				0	0	1	1		
10	−			1	0	1	0	−	
?			1)	1	0	0	1		

Figuur 5.19: *Een aftrekking waarbij de aftrekker groter is dan het aftrektal.*

*5.5 Ontwerp van een aftrekker voor twee binaire getallen

Op eenzelfde wijze als het ontwerpen van een optelschakeling, kan ook een aftrekschakeling gemaakt worden. We beginnen met het ontwerp van een *full subtractor* voor drie bits (het ontwerp van de half subtractor slaan we over). Later breiden we die uit naar een aftrekschakeling voor meerdere bits.

De functie die gerealiseerd wordt is $a-b-b_{in}$. Hierin is a het aftrektal, b de aftrekker en b_{in} de inkomende leen (Engels: borrow). In figuur 5.20 is links het bloksymbool te zien. Uitgang d geeft het verschil (Engels: difference) aan, uitgang b_{out} geeft aan of er geleend moet worden om de aftrekking uit te voeren.

We onderscheiden twee situaties. Als de aftrekking $a-b-b_{in}$ direct kan worden uitgevoerd, is de uitgaande borrow 0. Er hoeft immers niet geleend te worden. Het verschil d kan direct worden bepaald. In de tweede situatie kan de aftrekking niet direct worden uitgevoerd en moet er geleend worden. De uitgaande borrow is dan 1 en het verschil

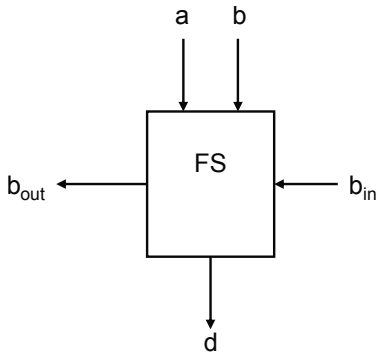
[3] Stel je voor dat je hier 3x woordwaarde voor zou krijgen!

[4] Deze leen geeft aan dat er een tekort is van 2^4. Nu is $+9-2^4=-7$. En $+3-10=-7$!.

kan worden uitgerekend met $10 + a - b - b_{in}$ (de 10 komt van het lenen). Samengevat:

$$\begin{array}{lll} a-b-b_{in} \geq 0 & \rightarrow & d = a-b-b_{in} \qquad b_{out} = 0 \\ a-b-b_{in} < 0 & \rightarrow & d = 10+a-b-b_{in} \qquad b_{out} = 1 \end{array} \tag{5.5}$$

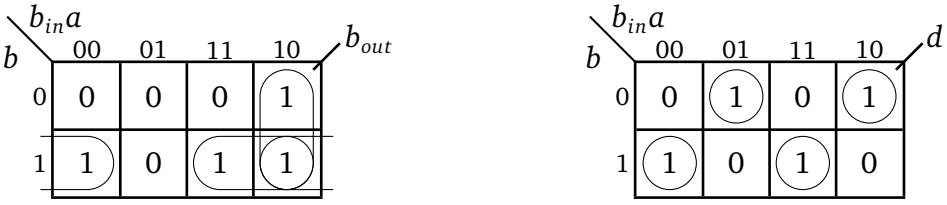
We kunnen nu de waarheidstabel opstellen, zie hiervoor figuur 5.20. Let erop dat de inkomende borrow in de eerste kolom staat.



b_{in}	a	b	b_{out}	d
0	0	0	0	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

Figuur 5.20: *Bloksymbool en waarheidstabel van een 1-bit full subtractor.*

Met behulp van Karnaughdiagrammen minimaliseren we de functies. Dit is te zien in figuur 5.21.



Figuur 5.21: *Karnaughdiagrammen voor b_{out} en d.*

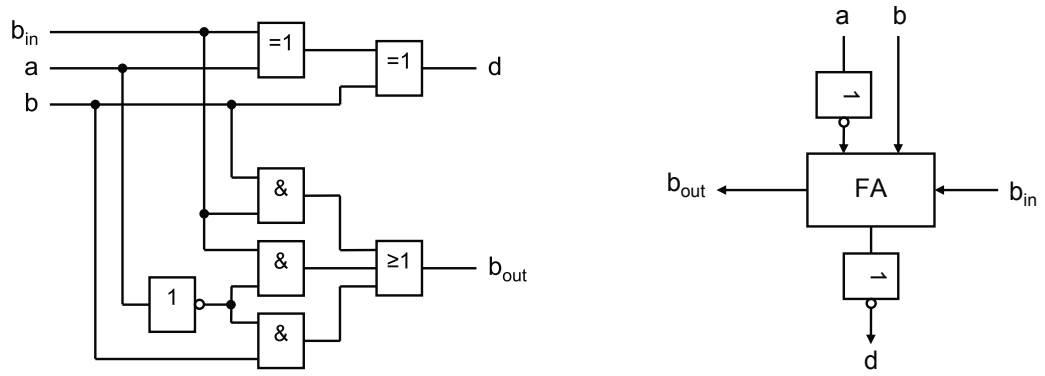
Na uitwerking van de Karnaughdiagrammen volgen de functies:

$$\begin{aligned} b_{out} &= \overline{a} \cdot b + \overline{a} \cdot b_{in} + b \cdot b_{in} \\ d &= b_{in} \oplus a \oplus b \end{aligned} \tag{5.6}$$

Deze twee functies hebben veel overeenkomsten met de functies van de 1-bit full adder. De functie van de verschilbit is identiek aan die van de sombit. In de functie van de uitgaande borrowbit moet het aftrektal worden geïnverteerd.

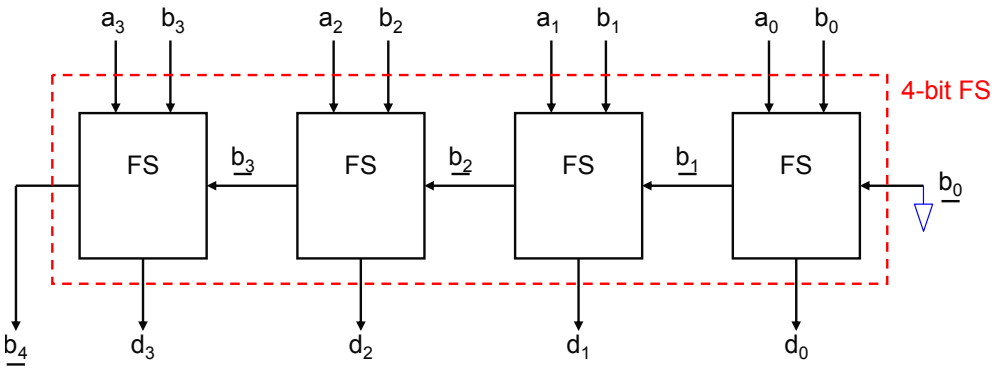
In figuur 5.22(a) is het schema van een 1-bit full subtractor te zien. Een full subtractor kan ook gerealiseerd worden met een full adder. Dit is te zien in figuur 5.22(b). De inverse van a wordt nu ook aangeboden aan dat deel van de schakeling dat d realiseert. Deze uitgang van de full adder moet geïnverteerd worden om de juiste waarden voor d te krijgen. We hebben gebruik gemaakt van de volgende gelijkheid:

$$\begin{aligned} d &= b_{in} \oplus a \oplus b \\ &= \overline{b_{in} \oplus \overline{a} \oplus b} \end{aligned} \tag{5.7}$$



(a) *Schema full subtractor.* **(b)** *Op basis van een full adder.*

Figuur 5.22: *Realisatie van een Full Subtractor.*



Figuur 5.23: *Schakeling voor een 4-bits full subtractor.*

De 1-bit full subtractors zijn, net als bij de opteller, in serie te schakelen tot grotere aftrekschakelingen. In figuur 5.23 is een schakeling te zien van een 4-bits full subtractor. Om verwarring tussen de B-bits en de borrows te vermijden, zijn de borrows onderstreept. De uitgaande borrow $\underline{b_4}$ kan gebruikt worden om een tekort te detecteren. Als $\underline{b_4} = 0$ is dan betekent dat het aftrektal groter is dan of gelijk is aan de aftrekker en dat het resultaat correct is. Als $\underline{b_4} = 1$ is dan betekent dat de aftrektal kleiner is dan de aftrekker en dat het resultaat dus niet correct is. Zie ook figuur 5.19

Een veel voorkomende operatie is het verlagen van een getal met 1. We bespreken deze *decrementer* niet. We merken op dat de decrementer op dezelfde wijze kan worden ontworpen als de incrementer.

In veel digitale systemen moet er zowel opgeteld als afgetrokken kunnen worden. Dat vereist een optel- en een aftrekschakeling (en de nodige logica om de informatie te routeren). In de praktijk wordt echter een optelschakeling gebruikt en wordt de wiskundige gelijkheid gebruikt:

$$A - B = A + (-B) \tag{5.8}$$

Dit vereist echter wel het gebruik van negatieve getallen. Negatieve binaire getallen worden verderop behandeld.

5.6 Vermenigvuldiger voor twee binaire getallen

Een derde belangrijke rekenkundige bewerking is vermenigvuldigen van twee getallen. We voeren hier de operatie $P = A \times B$ uit waarbij P het product, A het vermenigvuldigtal en B de vermenigvuldiger wordt genoemd[5].

Vermenigvuldigen van twee getallen bestaat uit het herhaald uitvoeren van deelvermenigvuldigingen van twee bits. In het binaire talstelsel in het erg makkelijk, er zijn maar twee tafels nodig: de tafel van 0 en de tafel van 1. Eventueel kan daarbij nog de tafel van 10 gebruikt worden, vergelijkbaar met de tafel van 10 in het decimale talstelsel. In figuur 5.24 zijn de drie tafels te zien.

×	0	1	10
0	0	0	0
1	0	1	10
10	0	10	100

Figuur 5.24: *De drie tafels in het binaire talstelsel.*

In figuur 5.25 is de vermenigvuldiging van 1101 en 1011 te zien. We schrijven de getallen recht onder elkaar zodat de cijfers met hetzelfde gewicht een kolom vormen, net zoals bij optellen. We beginnen met het vermenigvuldigen van de eenheden. Deze zijn beide 1. Uit de tafel van 1 lezen we af dat $1 \times 1 = 1$. We noteren deze 1 onder de streep in de kolom van de eenheden. Daarna volgt vermenigvuldigen met de 0 (het tweetal) van 1101. Dit levert een 0 op en die schrijven we in de kolom van de tweetallen. Daarna volgen nog deelvermenigvuldigingen met twee enen. Resumerend levert de vermenigvuldiging van 1 met 1101 natuurlijk 1101 op. We hadden dus ook direct 1101 onder de streep kunnen schrijven. We kunnen het proces van bit voor bit vermenigvuldigen versnellen door steeds een van de bits van 1011 met het getal 1101 te vermenigvuldigen.

$$\begin{array}{rl} 1101 & \\ \underline{1011} & \times \\ 1101 & \\ 11010 & \\ 000000 & \\ \underline{1101000} & + \\ 10001111 & \end{array}$$

Figuur 5.25: *Vermenigvuldiging van 1101_2 en 1011_2.*

Daarna volgt de 1 links van de meeste rechtse 1. Deze 1 stelt een tweetal voor. Hierdoor moeten we “inspringen” naar links: we schrijven rechts eerst een 0 en dan volgt de deelvermenigvuldiging. Het deelresultaat is 11010. Daarna volgt de 0 van 1011. Deze 0 stelt een viertal voor zodat we met twee nullen inspringen. Daarna schrijven we 0000.

[5] In de literatuur wordt de naamgeving ook wel verwisseld: A wordt de vermenigvuldiger genoemd en B het vermenigvuldigtal. De oorsprong ligt in het uitvoeren van de vermenigvuldiging als een serie optellingen: B moet A keer bij elkaar worden opgeteld, of A moet B keer bij elkaar worden opgeteld. Wij gebruiken de laatste definitie.

Normaliter schrijven we deze deelvermenigvuldiging niet op want het deelresultaat is toch 0. We doen het hier wel omdat we er later een schakeling voor willen ontwerpen. De meest linker 1 van 1011 stelt een achttal voor. We springen in met drie nullen en daarna schrijven we 1101 zodat het deelresultaat 1101000 volgt. Als alle deelvermenigvuldigingen zijn uitgevoerd, moeten de deelresultaten worden opgeteld. Daaruit volgt het eindresultaat 10001111.

Het nadeel van deze oplossing is dat er een multi-input opteller nodig is. Dat is lastig te ontwerpen. Daarbij moet voor elke bitbreedte een nieuwe opteller worden ontworpen. Slimmer is om tijdens het vermenigvuldigen tussentijds op te tellen. Dit is weergegeven in figuur 5.26. We kunnen nu gebruik maken van de eerder ontwikkelde opteller voor twee binaire getallen.

$$
\begin{array}{rl}
1101 & \\
\underline{1011} & \times \\
1101 & \\
\underline{11010} & + \\
100111 & \\
\underline{000000} & + \\
100111 & \\
\underline{1101000} & + \\
10001111 &
\end{array}
$$

Figuur 5.26: *Vermenigvuldiging en tussentijds optellen van 1101_2 en 1011_2.*

Vermenigvuldigen in het binaire talstelsel bestaat dus uit deelvermenigvuldigingen van twee bits en een serie optellingen.

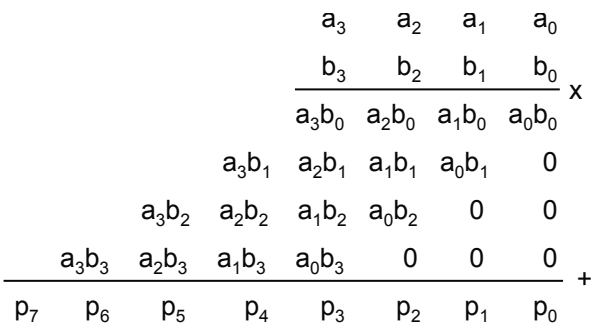
Het ontwerpen van een vermenigvuldiger voor twee bits is niet zo lastig. Er zijn maar twee tafels nodig, de tafels van 0 en van 1. We kunnen de tafels even uitschrijven, zie figuur 5.27. We zetten de numerieke nullen en enen weer om naar logische nullen en enen zoals we dat deden bij de opteller. Na invullen van de waarheidstabel zien we dat de tafels eenvoudig te realiseren zijn met behulp van een AND-poort.

×	0	1
0	0	0
1	0	1

a	b	$a \times b = 1$	$a \cdot b$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

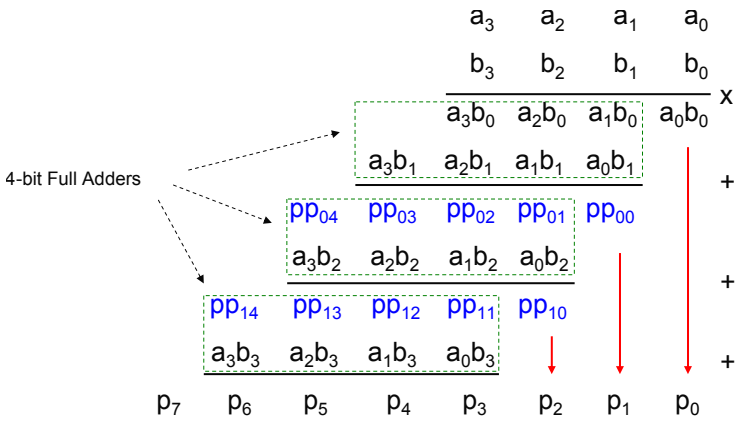
Figuur 5.27: *De twee tafels in het binaire talstelsel en de waarheidstabel.*

De vermenigvuldiging van twee 4-bits getallen $A = a_3a_2a_1a_0$ en $B = b_3b_2b_1b_0$ is te zien in figuur 5.28. De deelproducten a_0b_0 t/m a_3b_3 worden gevormd door in totaal zestien AND-poorten. Producten die in één kolom staan hebben hetzelfde gewicht en moeten worden opgeteld. Zo hebben a_3b_0, a_2b_1, a_1b_2 en a_0b_3 allemaal het gewicht 8. Het resultaat is een 8-bits getal $P = p_7p_6p_5p_4p_3p_2p_1p_0$.



Figuur 5.28: *Vermenigvuldiging uitgeschreven naar de afzonderlijke bits.*

In de bovenstaande uitvoering is echter een multi-input opteller nodig. Beter is om uit te gaan van de standaard opteller voor twee getallen. We gaan dus tussentijds optellen. We krijgen tussenresultaten die we *partial products* noemen. Er zijn in totaal drie 4-bits optellers nodig. Zie figuur 5.29.

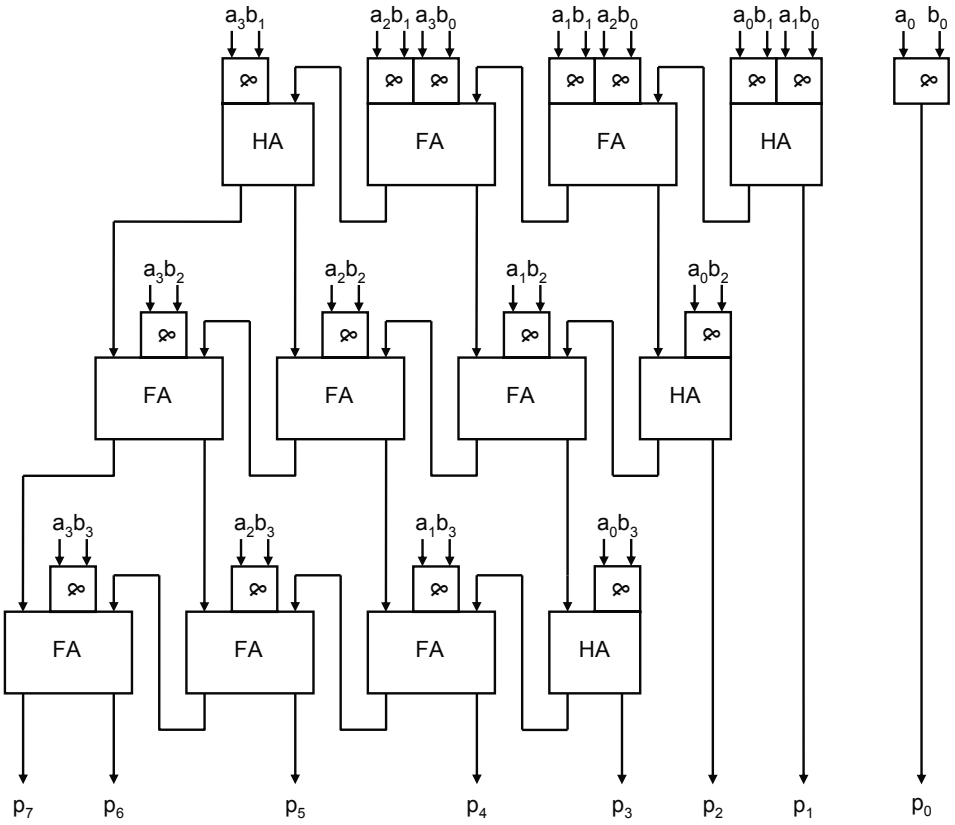


Figuur 5.29: *Vermenigvuldiging uitgeschreven naar de afzonderlijke bits, met tussentijdse optelling.*

Het blokschema is te vinden in figuur 5.30. Hierin staat FA voor Full Adder en HA voor Half Adder. De opbouw is met ripple carry adders uitgevoerd. Deze schakeling wordt een *parallel vermenigvuldiger* genoemd en is een pure combinatorische schakeling.

Merk op hoe elegant de structuur van de vermenigvuldiger is. Toch heeft ook deze oplossing z'n beperkingen. Het aantal AND-poorten stijgt kwadratisch met de bitbreedte en voor elke bit meer is er een extra opteller nodig die ook nog een bit breder moet zijn. Bedenk dat een 32-bits vermenigvuldiger 1024 AND-poorten en 31 optellers van 32 bits nodig heeft.

De grootste vertraging is van ingangen a_0b_1/a_1b_0 naar uitgangen $p7/p6$ en bedraagt acht optelsecties (en nog een AND-poort). Dit heeft weer te maken met het transport van de carry's. In de praktijk worden daarom ook wel andere varianten gebruikt zoals *carry save adder* vermenigvuldigers, *Wallace* en *Dadda trees* [92].



Figuur 5.30: *Blokschema van een 4x4-bits parallel vermenigvuldiger.*

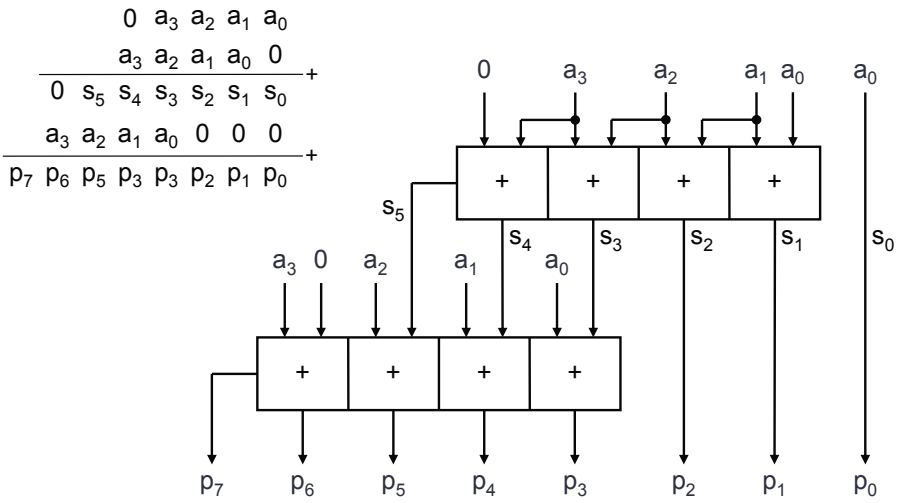
5.7 Vermenigvuldigen met een constante

Soms moet er vermenigvuldigd worden met een constante. We kunnen daar de eerder besproken parallel vermenigvuldiger voor gebruiken en een van de twee getallen als een serie logische nullen en enen aanbieden. Dit levert echter onnodig gebruik van veel hardware. Een vermenigvuldiging met een constante kan eenvoudig worden omgezet naar een serie optellingen. We nemen als constante het getal 11_{10}. We voeren eerst de vermenigvuldiging van 13_{10} met 11_{10} uit om het principe te tonen. Het getal 11_{10} is te schrijven als $8+2+1$ of, in machten van 2, als $2^3+2^1+2^0$. Dus de vermenigvuldiging is te schrijven als $13\cdot 8+13\cdot 2+13\cdot 1=13\cdot 2^3+13\cdot 2^1+13\cdot 2^0$. Nu is vermenigvuldigen met 8 (2^3) niets anders dat drie plaatsen naar links schuiven en aanvullen met nullen. Vermenigvuldigen met 2 (2^1) is één plaats naar links schuiven en aanvullen met een nul. De vermenigvuldiging van 1101_2 met 1011_2 is dus te schrijven als: $1101\times 1011=1101000+11010+1101$.

Vermenigvuldigen van een 4-bits getal $A=a_3a_2a_1a_0$ met 1011_2 is te schrijven als

$$a_3a_2a_1a_0\times 1011_2=a_3a_2a_1a_0000+a_3a_2a_1a_00+a_3a_2a_1a_0 \tag{5.9}$$

We kunnen dit omzetten in twee optellingen met het getal 1101_2 waarbij ook de geschoven varianten gebruikt worden. Zie figuur 5.31. Hier en daar worden wat nullen toegevoegd om de verschuivingen te realiseren of om de kolommen een identiek formaat te geven.



Figuur 5.31: *Blokschema van een vermenigvuldiger met de constante 11.*

5.8 Delen van twee binaire getallen

De deeloperatie voor gehele getallen laat zich wat minder makkelijk omschrijven. Er zijn twee situaties te onderscheiden. In het eerste geval gaat de deling precies op, wat inhoudt dat de rest na deling 0 is. We kunnen de deling dan schrijven als $Q = A \div B$, ook wel geschreven als $Q = A/B$. Hierin wordt Q het quotiënt, A het deeltal en B de deler genoemd. In het tweede geval is er na de deling een rest R die groter is dan 0. Dan is het beter om de deling als volgt te schrijven:

$$A = Q \cdot B + R \tag{5.10}$$

Voor R geldt: $0 \leq R \leq B-1$. Verder geldt dat het deeltal B niet 0 mag zijn. De uitkomst is dan onbepaald. We gaan stilzwijgend ervan uit dat aan deze voorwaarde wordt voldaan.

We beginnen met een deling in het decimale stelsel en geven daarbij uitleg over de gevolgde stappen. We gaan uit van de pen-en-papiermethode wat resulteert in een *staartdeling*.

Bij delen wordt de deler steeds een aantal keer van het deeltal afgetrokken waarbij dat aantal tussen 0 en 9 ligt. Dit aantal levert steeds één cijfer van het quotiënt op. Merk op dat het verkregen aftrektal wordt *opgelijnd* met het deeltal, zodanig dat de eerste stap 1, ...,9 keer gaat. Daarna wordt het aftrektal van de deler afgetrokken. Bij het resultaat wordt nu één cijfer van het deeltal aangehaald en wordt de procedure opnieuw uitgevoerd, net zolang tot er geen cijfers van het deeltal meer voorradig zijn.

In figuur 5.32 is de staartdeling van 12345 door 25 te zien. We proberen de 1 van de tienduizendtallen te delen door 25. We zien gelijk dat dat niet gaat. We trekken de 2 van de duizendtallen bij. De deling 12 door 25 gaat nog steeds niet. We trekken de 3 van de honderdtallen bij. Daarna voeren we de deling 123 door 25 uit. Het resultaat is 4, want $4 \times 25 = 100$. Deze 4 stelt het honderdtal van het quotiënt voor. We schrijven 100 onder de 123 en voeren een aftrekking uit, $123 - 100 = 23$. Daarna trekken we de 4 van de tientallen bij. We voeren de deling $234 \div 25$ uit, dat gaat 9 keer, want $9 \times 25 = 225$. We

```
                        12345 ÷ 25 = 493 ⟷ quotiënt
vier maal aftrekken     100 −
                         234
negen maal aftrekken     225 −
                          95
drie maal aftrekken       75 −
                          20 ⟷ rest
```

Figuur 5.32: *Deling van deeltal 12345 door deler 25 met als quotiënt 493 en rest 20.*

schrijven de 9 op. Dit stelt het tiental van het quotiënt voor. We trekken 225 van 234 af, schrijven 9 onder de streep en trekken de 5 van de eenheden bij. Daarna delen we 95 door 25, dat gaat 3 keer. We noteren de 3 bij het quotiënt en trekken 75 van de 95 af. We houden 20 over. Er zijn geen cijfers meer voorradig in het deeltal. Na deling is het quotiënt 493 en de rest 20.

Het mag duidelijk zijn dat deze deling niet al te lastig is. Er wordt meer van de rekenaar verwacht bij de deling 678352 ÷ 924. We gaan hier niet verder op in.

Delen in het binaire talstelsel is vergelijkbaar met delen in het decimale talstelsel. Het aftrektal kan nu slechts nul of één keer van het deeltal worden afgetrokken. Ook hier moet de deler worden opgelijnd met het deeltal voordat de eerste stap kan worden uitgevoerd. Zie figuur 5.33.

```
                        11010111 ÷ 1011 = 10011 ⟷ quotiënt
één maal aftrekken      1011 −
                        00100      één cijfer bijtrekken
nul maal aftrekken       0000 −
                         01001
nul maal aftrekken       0000 −
                         10011
één maal aftrekken        1011 −
                          10001
één maal aftrekken        1011 −
                          0110 ⟷ rest
```

Figuur 5.33: *Deling van deeltal 11010111 door deler 1011 met als quotiënt 10011 en rest 0110.*

Combinatorische delers leveren, net als vermenigvuldigers, een flinke schakeling op. In veel systemen worden daarom sequentiële delers gebruikt. We gaan hier niet verder op in. Zie onder andere [93, 94] voor meer informatie over combinatorische delers.

5.9 Introductie representaties van gehele getallen

Tot nu toe zijn alleen positieve getallen en nul behandeld. Getallen kunnen natuurlijk ook negatief zien, bijvoorbeeld om een tekort op een banksaldo aan te geven of de rich-

ting van een (negatieve) stroom. In het alledaags gebruik wordt de *signed magnitude* representatie (Nederlands: teken-en-grootte) gebruikt. Het ligt voor de hand om deze representatie ook in digitale systemen te gebruiken. Dat levert echter behoorlijk complexe schakelingen op. In digitale schakelingen en computers wordt daarom de two's complement representatie gebruikt. Deze representatie heeft als voordeel dat de digitale schakelingen eenvoudig van aard zijn en dat optellingen en aftrekkingen door dezelfde schakeling kunnen worden uitgevoerd. De two's complement representatie is gebaseerd op *modulair rekenen*.

We bespreken eerst de signed magnitude representatie, daarna bespreken we modulair rekenen en vervolgens de two's complement representatie.

5.10 Signed magnitude

In de *signed magnitude* representatie bestaat een getal uit een grootte en een teken dat aangeeft of het een positief of negatief getal is. Om aan te geven dat een getal negatief is, wordt het minteken (−) gebruikt. Een positief getal mag voorzien worden van een plus-teken (+) maar dat hoeft niet. Zo zijn de getallen +13 en −13 even groot (magnitude) maar *tegengesteld*. Er zijn twee weergaven van het getal nul, +0 en −0, beide met dezelfde waarde.

Het optellen van twee signed magnitude getallen kost enige moeite. Als de twee getallen hetzelfde teken hebben, moeten de grootten (magnitudes) worden opgeteld en krijgt het antwoord hetzelfde teken. Als de twee getallen verschillende tekens hebben, moet de kleinste grootte worden afgetrokken van de grootste grootte, en het antwoord krijgt het teken van de grootste grootte. We geven een aantal voorbeelden in het decimale talstelsel.

In figuur 5.34 is een aantal optellingen van positieve en negatieve getallen te zien. Uiterst links is de optelling van twee positieve getallen gegeven. Het antwoord is direct te berekenen met de procedure uit paragraaf 5.1. De optelling van twee negatieve getallen verloopt ook zonder enige moeite. De grootten worden opgeteld en het antwoord krijgt een minteken.

$$\begin{array}{r} +321 \\ +244 \\ \hline +565 \end{array} + \qquad \begin{array}{r} -456 \\ -123 \\ \hline -579 \end{array} + \qquad \begin{array}{r} +321 \\ -132 \\ \hline \downarrow \\ +321 \\ +132 \\ \hline +189 \end{array} - \qquad \begin{array}{r} +231 \\ -342 \\ \hline \downarrow \\ +342 \\ +231 \\ \hline +111 \\ \downarrow \\ -111 \end{array} +$$

Figuur 5.34: *Enkele optellingen met de signed magnitude representatie in het decimale talstelsel.*

Het optellen van een positief en een negatief getal levert wat meer werk op. In het derde voorbeeld is de grootte van het positieve getal groter dan de grootte van het negatieve getal. De optelling is dan direct om te zetten in een aftrekking. Het resultaat is positief en

krijgt (eventueel) een plusteken. In het laatste voorbeeld is de grootte van het negatieve getal groter dan die van het positieve getal. We zetten de optelling weer om in een aftrekking maar verwisselen ook de volgorde van de getallen. Het resultaat moet een minteken krijgen. Uiteraard zijn meer mogelijkheden te bedenken.

In oudere computers werd de binaire variant van signed magnitude gebruikt, zoals de IBM 704 [95]. Laten we eens kijken naar het algemene getal van n bits. Van een n-bit getal wordt de meest significante bit als tekenbit gebruikt zodat er nog $n-1$ bits overblijven voor de grootte. Een positief getal heeft een 0 als teken, een negatief getal gebruikt een 1 als teken. In figuur 5.35 zijn enkele representaties van signed magnitude getallen met een verschillend aantal bits in het binaire talstel gegeven.

+0	0000	$n=4$	−0	1000	$n=4$
+6	0110	$n=4$	−6	1110	$n=4$
+21	010101	$n=6$	−29	10011101	$n=8$
+127	01111111	$n=8$	−127	11111111	$n=8$
+511	0111111111	$n=10$	−511	1111111111	$n=10$

Figuur 5.35: *Enkele getallen in de signed magnitude representatie in het binaire talstelsel.*

Het bereik van een n-bits getal M in de signed magnitude representatie is:

$$-(2^{n-1}-1) \leq M \leq 2^{n-1}-1 \tag{5.11}$$

Het bereik bij de signed magnitude representatie is *symmetrisch*, er zijn evenveel positieve als negatieve getallen. Er zijn twee representaties van het getal 0, namelijk −0 en +0. Het bereik is trouwens makkelijk uit te breiden door meer bits te nemen. De tekenbit verandert daarbij van positie, het schuift op naar links.

In figuur 5.36 is een aantal voorbeelden te zien van optellingen in de signed magnitude representatie met binaire getallen. De voorbeelden zijn naar analogie van figuur 5.34. Merk op dat de tekenbits geen onderdeel zijn van de optellingen en aftrekkingen. In de figuur is dat weergegeven door een spatie tussen de tekenbits en de magnitudebits.

$$\begin{array}{r} 0\ 001 \\ 0\ 010 \\ \hline 0\ 011 \end{array} + \qquad \begin{array}{r} 1\ 010 \\ 1\ 011 \\ \hline 1\ 101 \end{array} + \qquad \begin{array}{r} 0\ 110 \\ 1\ 001 \\ \hline \downarrow \\ 0\ 110 \\ 0\ 001 \\ \hline 0\ 101 \end{array} + \qquad \begin{array}{r} 0\ 010 \\ 1\ 011 \\ \hline \downarrow \\ 0\ 011 \\ 0\ 010 \\ \hline 0\ 001 \\ \downarrow \\ 1\ 001 \end{array} +$$

$$\begin{array}{r} \\ \\ \\ \\ - \\ \hline \end{array} \qquad \begin{array}{r} \\ \\ \\ \\ - \\ \hline \end{array}$$

Figuur 5.36: *Enkele optellingen met de signed magnitude representatie in het binaire talstelsel.*

Gebruik van de signed magnitude representatie in optel- en aftrekschakelingen is niet handig. Afhankelijk van de tekens en grootten van de getallen moet bepaald worden of er moet worden opgeteld of afgetrokken en of de tekens moeten worden omgekeerd. Al met al een hoop ‘als’, ‘optellen’, ‘aftrekken’ en ‘vergelijken’. Dit levert complexe schakelingen op.

5.11 Modulair rekenen

In digitale systemen worden getallen opgeslagen met een eindig aantal bits. Een mooi voorbeeld hiervan is een eenvoudige microprocessor waar getallen in eenheden van bijvoorbeeld acht bits worden opgeslagen. Dat betekent dat er maar een eindig aantal getallen is weer te geven. Met acht bits zijn de getallen 0 t/m 2^8-1 (0 t/m 255) weer te geven. Er zijn $2^8 = 256$ combinaties (verschillende getallen) mogelijk. Een optelling van twee getallen van acht bits levert een resultaat van negen bits, maar het negende bit wordt niet opgeslagen. Alle berekeningen zijn modulo 2^8. Dat houdt in dat de *rest na deling door* 2^8 wordt opgeslagen. Zie ook paragraaf 2.10.

We zullen nu een paar voorbeelden geven van optellingen met vier bits. De algemene structuur van de optelling is te zien in figuur 5.37. Het resultaat bestaat uit de sombits s_3 t/m s_0, sombit s_4 wordt genegeerd[6].

$$\begin{array}{cc|cccc} & & a_3 & a_2 & a_1 & a_0 \\ & & b_3 & b_2 & b_1 & b_0 \\ \hline \cancel{s_4} & | & s_3 & s_2 & s_1 & s_0 \end{array} +$$

Figuur 5.37: *Een modulaire optelling van twee getallen van vier bits resulteert in een antwoord van vier bits.*

In figuur 5.38 zijn enkele optellingen met vier bits te zien. De eerste twee optellingen leiden tot een antwoord dat past in vier bits. De optelling van $1101_2 + 1001_2$ leidt tot het antwoord 10110_2 maar alleen de vier minst significante bits worden gebruikt. Het uiteindelijke antwoord is dus 0110_2. Geheel rechts is de optelling van $1111_2 + 1111_2$ te zien, met als uiteindelijk resultaat 1110_2.

$$\begin{array}{r} 0000 \\ 0000 \\ \hline 0 \mid 0000 \end{array} + \qquad \begin{array}{r} 1010 \\ 0101 \\ \hline 0 \mid 1111 \end{array} + \qquad \begin{array}{r} 1101 \\ 1001 \\ \hline 1 \mid 0110 \end{array} + \qquad \begin{array}{r} 1111 \\ 1111 \\ \hline 1 \mid 1110 \end{array} +$$

Figuur 5.38: *Enkele optellingen met vier bits.*

In figuur 5.39 zijn nog eens vier optellingen te zien. Hier is een van de op te tellen getallen 1111_2. De eerste optelling is $0001_2 + 1111_2$ en geeft als antwoord 10000_2. Het vierde sombit wordt genegeerd en het resultaat is 0000_2.

$$\begin{array}{r} 0001 \\ 1111 \\ \hline 1 \mid 0000 \end{array} + \qquad \begin{array}{r} 0010 \\ 1111 \\ \hline 1 \mid 0001 \end{array} + \qquad \begin{array}{r} 0011 \\ 1111 \\ \hline 1 \mid 0010 \end{array} + \qquad \begin{array}{r} 0100 \\ 1111 \\ \hline 1 \mid 0011 \end{array} +$$

Figuur 5.39: *Enkele optellingen met vier bits.*

Dezelfde optellingen zijn hieronder te zien maar nu zijn ze uitgevoerd in het decimale talstelsel (mod 2^4 betekent modulo 2^4):

[6] Deze sombit kan ook worden beschouwd als de uitgaande carrybit c_4.

$$\begin{array}{lcll} (1+15) \bmod 2^4 & = & 16 \bmod 2^4 & = & 0 \bmod 2^4 \\ (2+15) \bmod 2^4 & = & 17 \bmod 2^4 & = & 1 \bmod 2^4 \\ (3+15) \bmod 2^4 & = & 18 \bmod 2^4 & = & 2 \bmod 2^4 \\ (4+15) \bmod 2^4 & = & 19 \bmod 2^4 & = & 3 \bmod 2^4 \end{array}$$

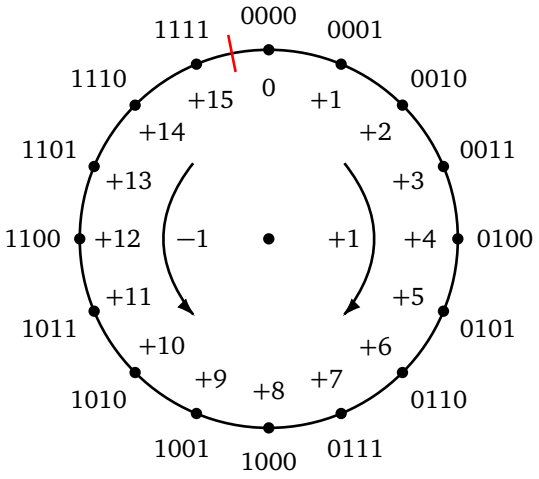
Figuur 5.40: *Enkele optellingen met vier bits (in decimale notatie).*

Wat opvalt is dat de optelling $1+15$ als antwoord 0 oplevert, de optelling $2+15$ levert als antwoord 1, $3+15$ levert 2, etc. Het optellen van 15 bij een getal levert dus een antwoord dat één lager is dan dat getal. Het lijkt wel alsof we -1 bij het getal hebben opgeteld, of anders gezegd, we hebben er 1 van afgetrokken. Dit is te verklaren omdat 15 te schrijven is als 2^4-1. Bij de eerder gegeven getallen wordt dus eigenlijk 2^4-1 opgeteld. De eerste optelling is dus te schrijven als:

$$\begin{aligned} 1+15 \mod 2^4 &= 1+(2^4-1) \mod 2^4 \\ &= 1-1+2^4 \mod 2^4 \\ &= 0+2^4 \mod 2^4 \\ &= 0 \mod 2^4 \end{aligned} \tag{5.12}$$

We kunnen het modulair rekenen inzichtelijk maken met een *getallencirkel*, te zien in figuur 5.41. In de cirkel zijn getallen 0 t/m 15 in zowel decimale als binaire vorm geplaatst. We beginnen bij 0 en lopen langs alle getallen door er steeds 1 bij op te tellen. Als we bij 15 één op tellen komen we weer uit bij 0 omdat er maar vier bits beschikbaar zijn voor het antwoord.

Als we nu 15 stappen vooruit lopen dan komen we bij een getal uit één lager dan waar we begonnen. Beginnen we bijvoorbeeld bij 12, dan komen we uit bij 11. In feite is dit hetzelfde als één stap achteruit gaan.



Figuur 5.41: *Getallencirkel voor 4-bits unsigned getallen.*

Uiteraard kunnen we ook 16 stappen vooruit (of achteruit) lopen. We komen dan weer op het vertrekpunt uit. Het laat zien dat optellen of aftrekken van 2^4 hetzelfde resultaat oplevert.

5.12 Two's complement representatie

In digitale systemen wordt de *two's complement representatie* gebruikt. Het is een slimme manier om gehele getallen met teken op te slaan en te gebruiken zodat optellen en aftrekken (de twee meest gebruikte bewerkingen in een computer) eenvoudig te realiseren zijn.

De representatie is gebaseerd op *modulair rekenen* dus alle getallen, ook het antwoord, hebben allemaal hetzelfde aantal bits. Laten we eens kijken naar optelling van 4-bits getallen. We kunnen de optelling van twee getallen A en B als volgt beschrijven:

$$A+B \mod 2^4 \tag{5.13}$$

Dit houdt in dat de optelling een antwoord van 4 bits oplevert. We kunnen nu gebruik maken van de mooie eigenschap:

$$A+B \mod 2^4 \quad = \quad A+(2^4+B) \mod 2^4 \tag{5.14}$$

We mogen B dus ook representeren als (2^4+B). Op het eerste gezicht lijkt dat niet zo zinvol totdat we bedenken dat dit ook geldt als B negatief is. We kunnen een negatief getal voorstellen door het op tellen bij 2^4. Zo mogen we -1 dus voorstellen als 2^4-1. Merk op dat 2^4-1 een *positief* getal is en dat B niet willekeurig groot kan zijn, er zijn immers maar vier bits beschikbaar. We doen dit alleen voor de negatieve getallen, voor positieve getallen verandert er niets.

Als voorbeeld nemen we de getallen $+5$ en -5 en representeren die met vier bits, te zien in figuur 5.42. We trekken $+5$ van 2^4 af om de two's complement representatie van -5 te berekenen:

$$\begin{array}{r} 16 \\ 5 \\ \hline 11 \end{array} - \qquad \begin{array}{r} 10000_2 \\ 0101_2 \\ \hline 1011_2 \end{array} - \qquad \begin{array}{r} 2^4 \\ +5 \\ \hline 2^4-5 \end{array} -$$

Figuur 5.42: *Two's complement representatie van het getal -5 met vier bits.*

Het binaire getal 0101_2 stelt de representatie van $+5$ voor, het binaire getal 1011_2 is de *representatie* van -5. (Eigenlijk stelt het getal natuurlijk 2^4-5 voor. Hierin is goed de relatie tussen de vier bits en -5 te zien.)

We kunnen de representaties van $+5$ en -5 gebruiken in een optelling. Dit is te zien in figuur 5.43. In de figuur zijn zowel de binaire als de decimale representaties gegeven. Optellen van 0101_2 met 1011_2 levert als antwoord 10000_2 op. We merken op dat de uitgaande carry 1 is. Dat komt omdat we eigenlijk de optelling $+5 + (2^4-5) = 2^4$ hebben uitgevoerd. De uitgaande carry moet dus geschrapt worden, die is het resultaat van de wijze waarop de two's complement representatie werkt. Merk op dat de getallen modulo 2^4 zijn, alleen de vier minst significante bits worden opgeslagen.

$$\begin{array}{rcr} 0101_2 & & +5 \\ 1011_2 & + & 2^4-5 \\ \hline 1)\,0000_2 & & 2^4+0 \\ \downarrow & & \\ 0000_2 & & 0 \end{array}$$

Figuur 5.43: *Two's complement optelling van +5 en −5.*

Wat opvalt is dat *alle* bits gebruikt zijn in de optelling, ook de meest significante bits. Dit was niet het geval bij de signed magnitude representatie.

Tot slot geven we nog twee voorbeelden. In figuur 5.44 zijn de representaties van −1 met vier bits en −97 met acht bits weergegeven. We gaan uit van de positieve varianten en voeren de omzetting uit zoals eerder is besproken.

$$\begin{array}{rcrcr} 16 & & 10000_2 & & 2^4 \\ 1 & - & 0001_2 & - & +1 \\ \hline 15 & & 1111_2 & & 2^4-1 \end{array} \qquad \begin{array}{rcrcr} 256 & & 100000000_2 & & 2^8 \\ 97 & - & 01100001_2 & - & +97 \\ \hline 159 & & 10011111_2 & & 2^8-97 \end{array} \; -$$

Figuur 5.44: *Two's complement representatie van de getallen −1 met vier bits en −97 met acht bits.*

Het binaire getal 1111_2 is de representatie van -1_{10}. Merk op dat de getallen modulo 2^4 zijn. Het binaire getal 10011111_2 is de representatie van -97_{10}. Merk op dat de getallen modulo 2^8 zijn.

5.12.1 Two's complement getallen met vier bits

We willen graag een evenwichtige verdeling van de positieve en negatieve representaties. De helft van de combinaties representeert positieve getallen (inclusief 0) en de andere helft representeert de negatieve getallen. Het bereik van een getal M gerepresenteerd als two's complement getal met vier bits ligt als volgt:

$$-8 \leq M \leq +7 \qquad \text{(met 4 bits)} \tag{5.15}$$

In tabel 5.2 zijn de two's complement representaties te zien met 4-bits getallen. In de linker kolommen staan de decimale waarden in de signed magnitude representatie. De middelste kolommen geven de two's complement representaties in decimaal formaat weer. De rechter kolommen geven de two's complement representaties in binair formaat weer.

Er zijn meer negatieve getallen dan positieve getallen, precies één negatief getal meer. Er is wel een representatie voor −8 maar niet voor +8. Dat komt omdat één representatie voor het getal 0 wordt gebruikt.

5.12.2 Tekenomkering

Met *tekenomkering* bedoelen we het negatief maken van een getal. Let erop dat een positief getal dan negatief wordt en een negatief getal wordt dan positief. Bij het omkeren

Tabel 5.2: *Two's complement representaties met vier bits.*

decimaal	2's C_{DEC}	2's C_{BIN}	decimaal	2's C_{DEC}	2's C_{BIN}
0	0	0000	−1	15	1111
1	1	0001	−2	14	1110
2	2	0010	−3	13	1101
3	3	0011	−4	12	1100
4	4	0100	−5	11	1011
5	5	0101	−6	10	1010
6	6	0110	−7	9	1001
7	7	0111	−8	8	1000

van het teken is het handiger om in plaats van 2^4 uit te gaan van $(2^4-1)+1$, want (2^4-1) levert allemaal enen op. Er moet nog wel 1 bij worden opgeteld. In figuur 5.45 is te zien hoe +5 wordt omgezet naar −5:

$$\begin{array}{rl} 1111_2 & \\ 0101_2 & - \quad +5 \\ \hline 1010_2 & \\ 1_2 & + \\ \hline 1011_2 & \quad -5 \end{array}$$

Figuur 5.45: *Two's complement representatie van het getal −5.*

Merk op dat we de schrijfwijze $2^4 - 5$ hebben vervangen door gewoon −5. Een binair getal aftrekken van $1\ldots1_2$ is heel gemakkelijk. Het antwoord is namelijk de inverse van de afzonderlijke bits van dat getal (Engels: complement). Daarna moet er 1 bij opgeteld worden. Dit werkt ook van negatief naar positief. Zie figuur 5.46. De procedure staat bekend als *het nemen van de two's complement*. Een informele benaming is *flip bits, add 1*.

$$\begin{array}{rlr} 0101_2 & & +5 \\ \downarrow & & \\ 1010_2 & & \\ 1_2 & + & \\ \hline 1011_2 & & -5 \end{array} \qquad \begin{array}{rlr} 1011_2 & & -5 \\ \downarrow & & \\ 0100_2 & & \\ 1_2 & + & \\ \hline 0101_2 & & +5 \end{array}$$

Figuur 5.46: *Omzetting van +5 naar −5 en van −5 naar +5.*

In figuur 5.47 worden van twee bijzondere getallen de tekens omgekeerd, namelijk −8 en 0. We voeren de procedure uit om het tegengestelde getal te krijgen: inverteren van alle bits en er dan +1 bij optellen. Een eventuele uitgaande carry wordt geschrapt.

Na uitvoering van de procedure wordt voor +8 dezelfde representatie als voor −8 gevonden. De procedure werkt dus niet voor −8 naar +8. Dit is het gevolg van de het *asymmetrisch bereik* van de representatie. Voor het getal 0 werkt de procedure wel.

$$
\begin{array}{lll}
1000_2 & -8 & \\
\downarrow & & \\
0111_2 & & \\
\quad 1_2 & + & \\
\hline
1000_2 & -8? &
\end{array}
\qquad\qquad
\begin{array}{lll}
0000_2 & +0 & \\
\downarrow & & \\
1111_2 & & \\
\quad 1_2 & + & \\
\hline
1)\ 0000_2 & -0 &
\end{array}
$$

Figuur 5.47: *Representaties van −8 en 0.*

5.12.3 Tekenbit

Uit tabel 5.2 volgt nog iets. Een two's complement getal is positief als de meest significante bit 0 is. Een two's complement getal is negatief als de meest significante bit 1 is. Het getal 0 wordt dus als positief gezien. De meeste significante bit wordt het *tekenbit* genoemd. We zullen later zien dat het teken een gewicht heeft.

5.12.4 Bereik two's complement

Het bereik van een n-bits getal M in de two's complement representatie is:

$$-2^{n-1} \leq M \leq 2^{n-1} - 1 \tag{5.16}$$

Het bereik is asymmetrisch. Er is geen tegengesteld getal voor -2^{n-1} en er is één representatie voor 0. Tabel 5.3 geeft enkele voorkomende bitbreedtes en bereiken weer.

Tabel 5.3: *Enkele voorkomende bitbreedtes en bereiken van two's complement getallen.*

4 bits	-8	$\leq M \leq$	$+7$
8 bits	-128	$\leq M \leq$	$+127$
16 bits	-32768	$\leq M \leq$	$+32767$
24 bits	-8388608	$\leq M \leq$	$+8388607$
32 bits	-2147483648	$\leq M \leq$	$+2147483647$

5.12.5 Tekenuitbreiding

Een two's complement getal kan met meer bits geschreven worden dan oorspronkelijk door middel van *tekenuitbreiding* (Engels: sign extension). De tekenbit moet gekopieerd worden naar de nieuw toe te voegen bits. Positieve getallen worden links aangevuld met nullen. Negatieve getallen worden links aangevuld met enen. In het onderstaande voorbeeld worden getallen +6 en −6 uitgebreid van vier naar acht bits. De onderstreping geeft de toegevoegde bits aan.

$$
\begin{array}{lll}
+6 & 0110 & \longrightarrow \quad \underline{0000}0110 \\
-6 & 1010 & \longrightarrow \quad \underline{1111}1010
\end{array}
$$

De tekenuitbreiding van negatieve getallen met enen is goed te zien bij het omzetten van een positief getal naar een negatief getal. De eerste stap in namelijk het inverteren van de afzonderlijke bits. Nullen worden dus enen. Dit is te zien in figuur 5.48.

$$
\begin{array}{rlr}
0110_2 & & +6 \\
\downarrow & & \\
1001_2 & & \\
\quad 1_2 & + & \\
\hline
1010_2 & & -6
\end{array}
\qquad\qquad
\begin{array}{rlr}
00000110_2 & & +6 \\
\downarrow & & \\
11111001_2 & & \\
\quad 1_2 & + & \\
\hline
11111010_2 & & -6
\end{array}
$$

Figuur 5.48: *De getallen +6 en −6 met vier en acht bits.*

5.12.6 Conversie tussen two's complement en decimaal

Om een decimaal getal om te zetten naar two's complement kunnen de volgende regels gebruikt worden. Is het decimale getal positief, dan volgt de omzetting zoals is besproken in paragraaf 2.7. Is het decimale getal negatief, dan moet eerst de positieve variant omgezet worden naar binair. Daarna volgt omzetting van het positieve binaire getal naar het negatieve binaire getal. Let erop dat de binaire getallen met voldoende bits moeten worden berekend. In dit geval gebruiken we acht bits.

$$+76_{10} \xrightarrow{\text{omzetting}} 01001100_2$$

$$-87_{10} \xrightarrow{\text{teken omkeren}} +87_{10} \xrightarrow{\text{omzetting}} 01010111_2 \xrightarrow{\text{flip bits, add 1}} 10101001_2$$

Om een two's complement getal om te zetten naar decimaal kunnen de volgende regels gebruikt worden. Is het two's complement getal positief, dan volgt de omzetting zoals is besproken in paragraaf 2.2. Is het two's complement getal negatief, dan volgt eerst de omzetting van het getal met dezelfde grootte maar positief. Daarna volgt omzetting van het positieve decimale getal naar het negatieve decimale getal.

$$01011011_2 \xrightarrow{\text{omzetting}} 91_{10}$$

$$10111111_2 \xrightarrow{\text{flip bits, add 1}} 01000001_2 \xrightarrow{\text{omzetting}} 65_{10} \xrightarrow{\text{teken omkeren}} -65_{10}$$

Alternatieve methoden voor conversie tussen binair en decimaal

We zien dat omzetten van (vooral) negatieve getallen nogal bewerkelijk is. Er zijn snellere methodes voor handen. Zo kan het omzetten van een two's complement getal naar een decimaal getal eenvoudig door het tekenbit als negatief gewicht te gebruiken. Gegeven een n-bits two's complement getal $A = a_{n-1}a_{n-2}\ldots a_1a_0$, dan is het decimale equivalent:

$$\begin{aligned} M &= -a_{n-1}\cdot 2^{n-1} + a_{n-2}\cdot 2^{n-2} + \ldots + a_1\cdot 2^1 + a_0\cdot 2^0 \\ &= -a_{n-1}\cdot 2^{n-1} + \sum_{i=0}^{n-2} a_i\cdot 2^i \end{aligned} \tag{5.17}$$

Let hierbij op het minteken bij $-a_{n-1}$. Voor $a_{n-1} = 0$ levert deze bit de bijdrage 0, voor $a_{n-1} = 1$ levert deze bit de bijdrage -2^{n-1}. Hieronder twee voorbeelden:

$$1011 \rightarrow -1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = -8 + 3 = -5$$

$$0110 \rightarrow \quad 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = \quad 0 + 6 = \quad 6$$

Omzetten van een negatief binair getal naar een decimaal equivalent kan ook als volgt. We zetten het binaire getal eerst gewoon om alsof het een unsigned getal betreft. Daarna trekken we er 2^n vanaf waarbij n het aantal bits is waaruit het binaire getal bestaat. We nemen als voorbeeld een 8-bits getal:

$$11010110_2 \xrightarrow{\text{omzetten}} 214_{10} \xrightarrow{2^8 \text{ aftrekken}} 214_{10} - 2^8 \xrightarrow{\text{uitrekenen}} -42_{10}$$

5.12.7 Two's complement en hexadecimaal

Hexadecimale getallen worden gebruikt om binaire getallen gecomprimeerd op te schrijven en kunnen dus ook negatieve getallen voorstellen. Uiteraard moeten de binaire getallen een veelvoud van vier bits zijn. Hexadecimale getallen die beginnen met de cijfers 8 t/m F zijn negatief, want ze beginnen met een binaire 1.

Voorbeeld:

$$\begin{aligned} D0_{16} &= 1101.0000_2 = -2^7 + 2^6 + 2^4 = -48_{10} \\ FFFFFFFF_{16} &= 111\ldots111_2 = -2^{31} + 2^{30} + \ldots + 2^1 + 2^0 = -1_{10} \end{aligned}$$

We kunnen direct gebruik maken van de two's complement representatie. Een hexadecimaal cijfer tussen 8 en F stelt een negatief getal voor. Dit geldt alleen voor het meest significante cijfer. Dus:

$$\begin{aligned} 8_{16} &= -8_{10} \quad & 9_{16} &= -7_{10} \quad & A_{16} &= -6_{10} \quad & B_{16} &= -5_{10} \\ C_{16} &= -4_{10} \quad & D_{16} &= -3_{10} \quad & E_{16} &= -2_{10} \quad & F_{16} &= -1_{10} \end{aligned}$$

Voorbeeld:

$$\begin{aligned} C5_{16} &= -4\cdot16 + 5 = -64 + 5 = -59_{10} \\ ACE_{16} &= -6\cdot16^2 + 12\cdot16^1 + 14\cdot16^0 = 1330_{10} \\ 8000_{16} &= -8\cdot16^3 = -32768_{10} \\ FFFF_{16} &= -1\cdot16^3 + 15\cdot16^2 + 15\cdot16^1 + 15\cdot16^0 = -1_{10} \end{aligned}$$

5.12.8 Bepalen van het aantal bits voor een two's complement getal

Om het aantal bits te bepalen van een getal moeten we uitgaan van het bereik. Voor een getal M weergegeven als een n-bits two's complement getal geldt dat:

$$-2^{n-1} \leq M \leq +2^{n-1} - 1 \tag{5.18}$$

We gaan uit van het getal M dat ligt tussen de negatieve waarde M_- en positieve waarde M_+. We kunnen dan schrijven dat:

$$\begin{aligned} -2^{n-1} &\leq M_- \\ M_+ &\leq +2^{n-1} - 1 \end{aligned} \tag{5.19}$$

We schrijven de ongelijkheden even opnieuw op en werken de mintekens weg:

$$\begin{aligned} 2^{n-1} &\geq |M_-| \\ 2^{n-1} &\geq M_+ + 1 \end{aligned} \tag{5.20}$$

waarbij $|M_-|$ de absolute waarde is van M_-. Het minimum aantal bits wordt bepaald door de grootste van de getallen $M_+ + 1$ en $|M_-|$.

Als voorbeeld bepalen we het aantal bits van het getal M dat waarden kan aannemen tussen -99 en 57. Dan is $|-99|$ groter dan $57+1$. Dus geldt:

$$2^{n-1} \geq |-99| \quad \longrightarrow \quad n \geq \frac{\log|-99|}{\log 2} + 1 \quad \longrightarrow \quad n \geq 7{,}629\ldots \tag{5.21}$$

Er zijn dus minstens acht bits nodig. Merk op dat de $+1$ afkomstig is van de exponent van de 2-macht. Daar staat immers $n-1$.

5.13 Optellen met two's complement

We zullen nu een aantal optellingen van twee two's complement getallen bespreken. Er zijn zeven varianten mogelijk: beide getallen zijn positief, beide getallen zijn negatief, een positief getal en een negatief getal met grotere magnitude, een positief getal en een negatief getal met een kleiner magnitude, een negatief getal en een positief getal met kleinere magnitude, een negatief getal en een positief getal met grotere magnitude en twee tegengestelde getallen. Deze voorbeelden leveren strikt genomen geen bewijs dat het voor alle gevallen geldig is. Zie voor een strikt bewijs [96].

Noot: bij een optelling kan *sign overflow* optreden als het antwoord buiten de representatie van vier bits ligt. Bij onderstaande voorbeelden gebeurt dat niet. Overflow wordt besproken in paragraaf 5.19.

Twee positieve getallen. Het optellen van twee positieve getallen is in feite identiek aan het optellen van twee unsigned getallen waarbij de tekenbit 0 is. Neem als voorbeeld de optelling van $+5$ en $+2$.

$$\begin{array}{rcr} +5 & & 0101_2 \\ +2 & + & 0010_2 \\ \hline +7 & & 0111_2 \end{array} \quad +$$

Figuur 5.49: *Two's complement optelling van* $+5$ *en* $+2$.

De tekenbit van het antwoord is 0. Dat betekent dat het antwoord positief is, zoals verwacht.

Twee negatieve getallen. Neem als voorbeeld de optelling van -3 en -4. De two's complement representatie van -3 is 1101_2, die van -4 is 1100_2.

$$\begin{array}{rcr} -3 & & 1101_2 \\ -4 & + & 1100_2 \ + \\ \hline -7 & & \not{1}\,|\,1001_2 \end{array}$$

Figuur 5.50: *Two's complement optelling van* -3 *en* -4.

De tekenbit van de som is nu 1. Dit geeft aan dat de som negatief is. De optelling genereert een uitgaande carry. Deze carry moet geschrapt worden want die is het resultaat van de wijze waarop de two's complement representatie werkt. Het antwoord is dus 0111_2 (-7).

Positief getal en een negatief getal met kleinere magnitude. Neem als voorbeeld de optelling van $+6$ en -3. De two's complement representatie van -3 is 1101_2.

$$\begin{array}{rcr} +6 & & 0110_2 \\ -3 & + & 1101_2 \ + \\ \hline +3 & & \not{1}\,|\,0011_2 \end{array}$$

Figuur 5.51: *Two's complement optelling van* $+6$ *en* -3.

De tekenbit van het antwoord is 0 dat aangeeft dat het antwoord positief is. Merk op dat de tekenbits onderdeel zijn van de optelling. Verder genereert de optelling een uitgaande carry. Deze carry moet geschrapt worden want die is het resultaat van de wijze waarop de two's complement representatie werkt. De uiteindelijk antwoord is dan 0011_2.

Positief getal en een negatief getal met grotere magnitude. Neem als voorbeeld de optelling van $+3$ en -8. De two's complement representatie van -8 is 1000_2.

$$\begin{array}{rcr} +3 & & 0011_2 \\ -8 & + & 1000_2 \ + \\ \hline -5 & & 1011_2 \end{array}$$

Figuur 5.52: *Two's complement optelling van* $+3$ *en* -8.

De tekenbit van het antwoord is nu 1 dat aangeeft dat het antwoord negatief is. Het antwoord van de optelling is dus 1011_2 (-5).

Een negatief getal en een positief getal met kleinere magnitude. Neem als voorbeeld de optelling van -5 en $+3$. De two's complement representatie van -5 is 1011_2, die van $+3$ is 0011_2.

$$\begin{array}{rcr} -5 & & 1011_2 \\ +3 & + & 0011_2 \ + \\ \hline -2 & & 1110_2 \end{array}$$

Figuur 5.53: *Two's complement optelling van* -5 *en* $+3$.

De tekenbit van het antwoord is 1 dat aangeeft dat het antwoord negatief is. Het antwoord is 1110_2 en dat stelt -2 voor.

Een negatief getal en een positief getal met grotere magnitude. Neem als voorbeeld -3 en $+5$. De two's complement representatie van -3 is 1101_2.

$$\begin{array}{rcr} -3 & & 1101_2 \\ +5 & + & 0101_2 \quad + \\ \hline +2 & & \not{1}\,|\,0010_2 \end{array}$$

Figuur 5.54: *Two's complement optelling van* -3 *en* $+5$*.*

De tekenbit van het antwoord is 0 dat aangeeft dat het antwoord positief is. De uitgaande carry moet worden geschrapt want die komt voor uit de wijze waarop de two's complement representatie werkt.

Twee tegengestelde getallen. Neem als voorbeeld de optelling van $+1$ en -1. De two's complement representatie van -1 is 1111_2.

$$\begin{array}{rcr} +1 & & 0001_2 \\ -1 & + & 1111_2 \quad + \\ \hline 0 & & \not{1}\,|\,0000_2 \end{array}$$

Figuur 5.55: *Two's complement optelling van* $+1$ *en* -1*.*

Het antwoord is, zoals verwacht, 0. Ook hier moet de uitgaande carry geschrapt worden.

5.14 Optelschakeling voor two's complement getallen

Two's complement getallen kunnen worden opgeteld met de eerder ontworpen opteller voor unsigned getallen. Dat is namelijk de reden voor het gebruik van two's complement getallen. Voor vier bits is dit te zien in figuur 5.12.

5.15 Aftrekken met two's complement

Een aftrekking met two's complement getallen is om te zetten in een optelling. We gaan uit van de gelijkheid:

$$A - B = A + (-B) \tag{5.22}$$

In feite moeten we van getal B het two's complement nemen en bij A optellen. Hieronder volgen vier voorbeelden. Bij elk voorbeeld is links de two's complement aftrekking $A-B$ te zien, rechts de optelling $A+(-B)$.

De aftrekkingen zijn uitgevoerd volgens de eerder besproken aftrekprocedure in paragraaf 5.4. Naast elke aftrekking is de corresponderende optelling geplaatst. Bij de

0111	0111			1101	1101		
0110	− 1010	+		1001	− 0111	+	
0 | 0001	1 | 0001			0 | 0100	1 | 0100		
0101	0101			0011	0011		
0101	− 1011	+		1100	− 0100	+	
0 | 0000	1 | 0000			1 | 0111	0 | 0111		

Figuur 5.56: *Enkele aftrekkingen in de two's complement representatie.*

optelling is de normale optelprocedure gebruikt. Beide bewerkingen leveren een correct resultaat op, maar de uitgaande borrows en carry's zijn elkaars inverse. Dat is ook niet verwonderlijk want de aftrekking wordt omgezet in een optelling met een tegengesteld getal in de two's complement representatie. Voor het voorbeeld linksboven geldt:

Aftrekken:

$+7-6=+1$ (uitgaande borrow is 0, geen leen nodig)

Optellen:

$+7+(2^4-6)=+7-6+2^4=+1+2^4$ (uitgaande carry is 1, resultaat is 2^4 te groot)

Om de aftrekking in een optelling om te zetten moet van B het two's complement bepaald worden. Dat lukt alleen niet bij het meest negatieve getal. Daar bestaat geen positieve evenknie van. De procedure voor het nemen van de two's complement levert precies hetzelfde getal. Zie hiervoor figuur 5.47.

Toch kan dit getal gebruikt worden in optellingen en aftrekkingen zoals te zien is in figuur 5.57. Dit is te verklaren omdat alle bewerkingen modulo 2^4 zijn:

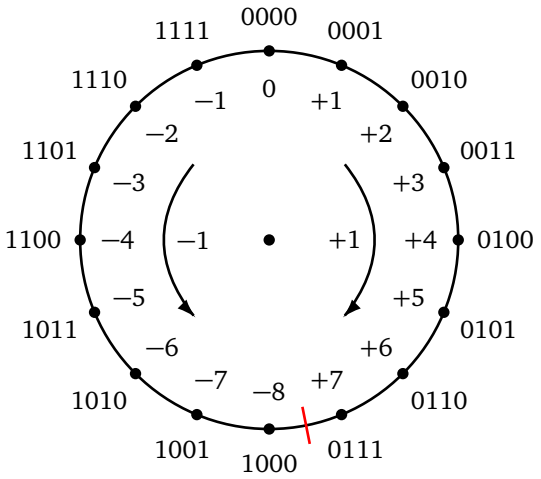
$$A-(-8) \mod 2^4 \quad = \quad A+(-8) \mod 2^4 \tag{5.23}$$

Het verschil tussen het linkerlid en rechterlid is precies 2^4.

1111	1111			1000	1000		
1000	− 1000	+		1000	− 1000	+	
0 | 0111	1 | 0111			0 | 0000	1 | 0000		

Figuur 5.57: *Enkele aftrekkingen met het meest negatieve getal.*

We kunnen het rekenen met two's complement getallen inzichtelijk maken met een two's complement getallencirkel. Deze cirkel is te zien in figuur 5.58. In de cirkel zijn getallen −8 t/m +7 in zowel decimale als binaire vorm weergegeven. Om de aftrekking $7-6$ uit te voeren, beginnen we bij +7 en lopen dat 6 stappen terug. We komen uit bij +1. Maar we kunnen ook 10 stappen vooruit lopen. We overschrijden dan de grens tussen 15 en 0 en komen weer uit bij +1. Het overschrijden van de grens zorgt voor een uitgaande carry.



Figuur 5.58: *De cirkel voor 4-bits two's complement getallen.*

5.16 Aftrekschakeling voor twee two's complement getallen

We hebben al eerder gezien dat de aftrekoperatie $A-B$ kan worden omgezet in de opteloperatie $A+(-B)$. Hiervoor moeten we van B het two's complement bepalen. Deze *negator* is eenvoudig te realiseren door alle bits van B te inverteren en er vervolgens één bij op tellen.

Een veelgebruikte uitdrukking is te vinden in onderstaande vergelijking:

$$\begin{aligned} D &= A-B \\ &= A+(-B) \\ &= A+(\overline{B}+1) \\ &= A+\overline{B}+1 \end{aligned} \tag{5.24}$$

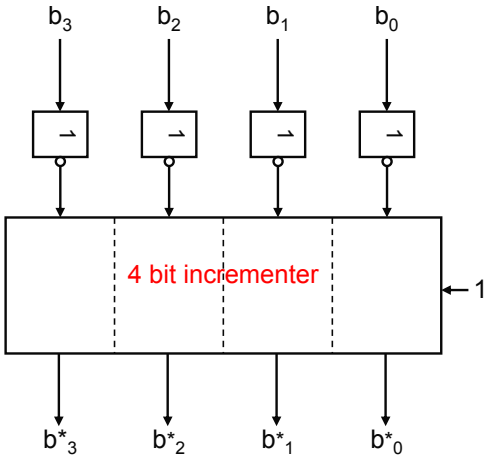
Hierin stelt $\overline{B}$ de bit-voor-bit inverse van B voor. We hebben hier rekenkundige en logische operaties met elkaar verweven. Dit is formeel niet helemaal juist maar het helpt ons om een digitale schakeling voor de aftrekker te realiseren.

De inversen van de bits zijn te realiseren met NOT-poorten. Voor het optellen van één kan de *incrementer* uit paragraaf 5.3.4 gebruikt worden. De schakeling is te zien in figuur 5.59. Hierin stellen $b_3 \ldots b_0$ de bits van het getal B en $b_3^* \ldots b_0^*$ de bits van de two's complement representatie voor.

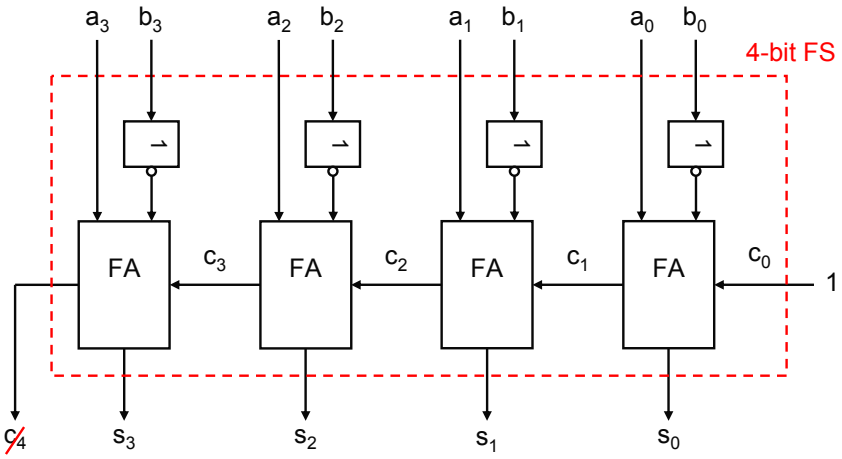
Vervolgens moeten we de uitkomst van de negator optellen bij getal A door middel van een 4-bits opteller. Dat zou een extra opteller opleveren. We kunnen echter voor het verhogen met één de ingaande carry c_0 gebruiken van de opteller. De incrementer komt hierdoor te vervallen. De volledige schakeling is te zien in figuur 5.60.

5.17 Optel-aftrekschakeling voor twee two's complement getallen

Een optelschakeling kan gecombineerd worden met een aftrekschakeling. Een (nieuw) besturingssignaal $\overline{o}/a$ geeft aan welke operatie uitgevoerd moet worden: als $\overline{o}/a = 0$ moet er worden opgeteld, als $\overline{o}/a = 1$ moet er worden afgetrokken.



Figuur 5.59: *Een 4-bits two's complement negator.*



Figuur 5.60: *Een 4-bits two's complement full subtractor.*

Bij optellen moeten de bits van B ongewijzigd aangeboden worden aan de opteller. Bij aftrekken moeten de bits van B geïnverteerd worden. Het al dan niet inverteren kan gerealiseerd worden met een omschakelbare buffer/inverter. Deze schakeling hebben we al eerder gezien, het is de bekende EXOR-poort.

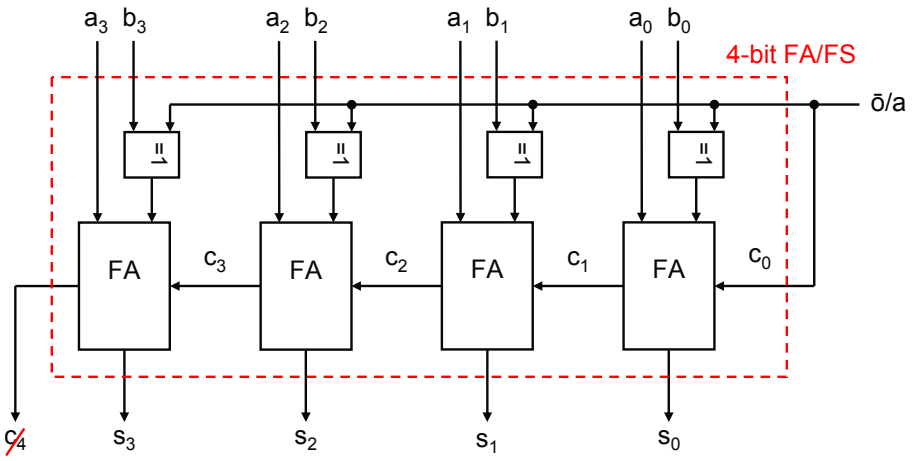
Het stuursignaal $\overline{o}/a$ moet met de carry-ingang verbonden worden, want bij optellen moet de carry-ingang verbonden zijn met een 0 en bij aftrekken moet de carry-ingang

COMPLEMENT EN NEGATION

In computerterminologie worden de termen complement en negation gebruikt. Complement betekent bijna altijd de logische inverse van een bitpatroon. Als het bitpatroon een getal voorstelt, dan wordt dit het one's complement genoemd. Met negation wordt het nemen van de two's complement bedoeld. Veel processoren hebben een zogenoemde `neg`-instructie. Wel even opletten: in de logica wordt met negation altijd het logisch complement bedoeld, de inverse dus. Logisch toch?

verbonden zijn met een 1.

In figuur 5.61 is de schakeling te zien. De EXOR-poorten geven onder besturing van $\overline{o}/a$ de bits van B ongewijzigd of geïnverteerd door. De carry-ingang c_0 is verbonden met signaal $\overline{o}/a$.



Figuur 5.61: *Een 4-bits two's complement full adder-subtractor.*

5.18 Unsigned versus two's complement

Hoe de getallen gebruikt en/of gelezen moeten worden, hangt af van de interpretatie van de gebruiker. Voor de digitale schakeling is er geen verschil (en dat is ook de bedoeling). In figuur 5.62 is de optelling van 1111_2 met 1111_2 te zien. We kunnen dit binaire getal interpreteren als $+15$ (unsigned) of -1 (signed). In het geval van de unsigned representatie is er sprake van een uitgaande carry, bij de two's complement representatie moet een uitgaande carry geschrapt worden.

unsigned		bits		2's com.		signed	
15		1111		2^4-1		-1	
15	$+$	1111	$+$	2^4-1	$+$	-1	$+$
$16+14$		$1 \mid 1110$		2^4+2^4-2		-2	

Figuur 5.62: *Unsigned versus two's complement.*

5.19 Overflow

Bij het het optellen van twee two's complement getallen kan het zijn dat het antwoord buiten het bereik van de representatie komt te liggen. Er wordt dan gesproken van *sign overflow* (Engels: overstromen, overlopen), vaak kortweg overflow genoemd. Overflow bij optellen *kan* voorkomen als twee getallen met *gelijk* teken worden opgeteld. Optellen van twee getallen met verschillend teken kan nooit een overflow opleveren.

In figuur 5.63 is een aantal optellingen te zien waarbij overflow optreedt. Tevens zijn de uitgaande carry's te zien.

$$\begin{array}{r} 0110 \\ 0011 \\ \hline 0\,|\,1001 \end{array} + \qquad \begin{array}{r} 0111 \\ 0001 \\ \hline 0\,|\,1000 \end{array} + \qquad \begin{array}{r} 1101 \\ 1001 \\ \hline 1\,|\,0110 \end{array} + \qquad \begin{array}{r} 1000 \\ 1000 \\ \hline 1\,|\,0000 \end{array} +$$

Figuur 5.63: *Enkele optellingen waarbij overflow optreedt.*

De twee voorbeelden links betreffen optellingen van positieve getallen. De tekenbits zijn 0. Kijken we naar de antwoorden dan zien we dat de tekenbits 1 zijn. Er heeft dus een *tekenwissel* plaatgevonden. Zo levert de meest linkse optelling van de getallen 0110_2 (+6) en 0011_2 (+3) als antwoord 1001_2 op. In de two's complement representatie is dat −7. Merk op dat de uitgaande carry 0 is. Hieraan is niet te zien of de optelling overflow oplevert.

In de twee rechter optellingen worden twee negatieve getallen opgeteld. We verwachten natuurlijk een negatief antwoord. Maar de antwoorden zijn positief. Ook hier heeft een tekenwissel plaatsgevonden. Merk op dat de uitgaande carry nu 1 is, maar hieraan is niet te zien of de optelling overflow oplevert.

Overflow is gemakkelijk te detecteren. Als de tekens van de op te tellen getallen gelijk zijn en anders zijn dan dat van het antwoord is er sprake van overflow. We bekijken het geval van twee 4-bits getallen $A = a_3a_2a_1a_0$ en $B = b_3b_2b_1b_0$. Het antwoord is de 4-bits som $S = s_3s_2s_1s_0$. Er is sprake van overflow als:

$$(a_3 = b_3) \neq s_3 \tag{5.25}$$

Een andere formulering is:

$$(a_3 = b_3) \cdot (a_3 \neq s_3) \tag{5.26}$$

De logische functie voor overflow van een 4-bit full adder is:

$$OV_{op} = \overline{a_3} \cdot \overline{b_3} \cdot s_3 + a_3 \cdot b_3 \cdot \overline{s_3} \tag{5.27}$$

Bij een aftrekking (die is omgezet in een optelling) *kan* overflow optreden als de af te trekken getallen *verschillende* tekens hebben, getal B wisselt immers van teken. De bits van B (dus ook de tekenbit) worden namelijk geïnverteerd aangeboden aan de opteller, zie figuur 5.60. Daarnaast geldt dat de som en getal A verschillende tekens moeten hebben. Er is sprake van overflow als:

$$a_3 \neq (b_3 = s_3) \tag{5.28}$$

Een andere formulering is:

$$(a_3 \neq b_3) \cdot (a_3 \neq s_3) \tag{5.29}$$

We moeten hier even goed opletten bij het getal 0 als aftrekker want de tekenbit van 0 verandert in een 1 bij inverteren (merk op dat de benodigde optelling van +1 bij het nemen

van de two's complement via de inkomende carry verloopt). Gelukkig zijn de tekenbits van A en de som identiek zodat er geen overflow wordt constateert. Het meest negatieve getal -8 vormt, ondanks dat het tegengestelde getal niet in 4-bits two's complement te representeren is, geen probleem omdat de inverse van de tekenbit een 0 oplevert.

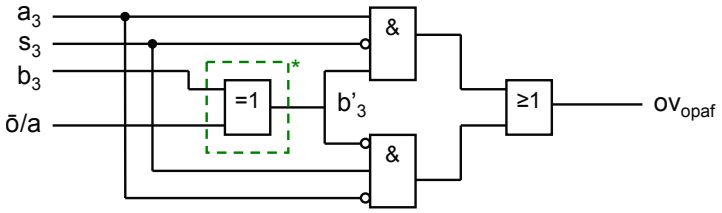
De functie voor overflow van een 4-bits full subtractor is:

$$OV_{af} = \overline{a_3} \cdot b_3 \cdot s_3 + a_3 \cdot \overline{b_3} \cdot \overline{s_3} \tag{5.30}$$

We kunnen functies voor overflow van de opteller en aftrekker combineren voor de opteller/aftrekker:

$$OV_{opaf} = \overline{a_3} \cdot \overline{(\overline{o}/a \oplus b_3)} \cdot s_3 + a_3 \cdot (\overline{o}/a \oplus b_3) \cdot \overline{s_3} \tag{5.31}$$

Het schema is te vinden in figuur 5.64. De EXOR-poort is al beschikbaar in de schakeling voor de opteller/aftrekker van figuur 5.61.



Figuur 5.64: *Overflowdetectie two's complement opteller/aftrekker.*

Detectie van overflow op basis van carry's

We kunnen overflow ook nog op een andere manier detecteren. Hiervoor onderzoeken we de waarheidstabel van de full adder voor de tekenbits van de getallen, zie tabel 5.4. Zoals eerder is opgemerkt zijn de tekenbits onderdeel van de optelling. Aan de tabel is een kolom V toegevoegd. Een 1 in de kolom geeft aan dat er overflow is opgetreden, een 0 betekent geen overflow.

Tabel 5.4: *Waarheidstabel 1-bit full adder voor de tekenbits van een 4-bits opteller.*

c_3	a_3	b_3	c_4	s_3	V
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	1
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	1	0

Uiteraard is de functie V identiek aan (5.27). Maar V is ook uit te drukken in de carry's. Er treedt namelijk overflow op als de inkomende carry bij de tekenbit ongelijk is aan de uitgaande carry bij de tekenbit. In formulevorm:

$$V = c_4 \oplus c_3 \tag{5.32}$$

Bij een aftrekking in two's complement worden de b-bits geïnverteerd aan een opteller aangeboden zoals te zien is in figuur 5.60. De optellingen van de tekenbits is weergegeven in tabel 5.5. De tabel wordt gevonden door de optelling $a_3 + \overline{b_3} + c_3$ uit te voeren.

Tabel 5.5: *Waarheidstabel 1-bit full adder voor de tekenbits van een 4-bits aftrekker.*

c_3	a_3	b_3	c_4	s_3	V
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	1	0	0

De twee enen in de kolom V volgen uit formule (5.30). De 1 in de derde regel komt voort uit het feit dat een negatief getal ($a_3 = 1$) min een positief getal ($b_3 = 0$) nooit een positief getal ($s_3 = 0$) kan opleveren. De 1 in de zesde regel geeft aan dat een positief getal ($a_3 = 0$) min een negatief getal ($b_3 = 1$) nooit een negatief getal ($s_3 = 1$) kan opleveren.

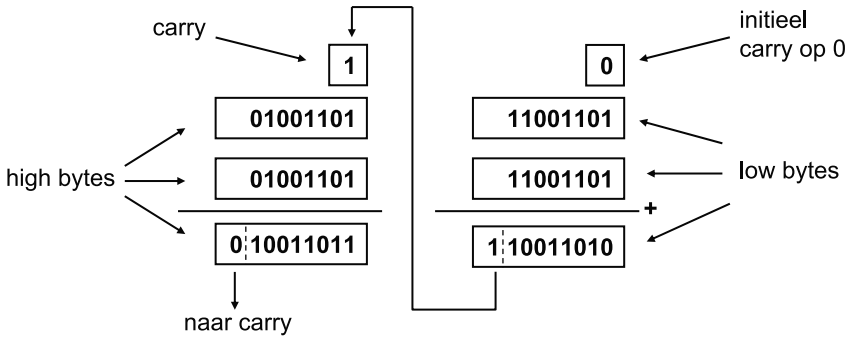
Voor de functie van de overflow bij two's complement aftrekking vinden we de eerder besproken overflowfunctie (5.32).

*5.20 Optellen en aftrekken in een processor

Het is niet verwonderlijk dat in moderne processoren gerekend wordt in de two's complement representatie. Optellen en aftrekken kunnen immers door dezelfde digitale schakeling worden uitgevoerd. Hoewel moderne processor met 32 of 64 bits rekenen, blijven de principes hetzelfde. Er zijn ook processoren op de markt die met 8 bits rekenen en ook daar blijven de principes van two's complement hetzelfde.

In eenvoudige microprocessors worden de getallen in 8 bits eenheden opgeslagen. Een optelling van 16 bits kan niet in één keer worden uitgevoerd maar moet in twee stukken worden opgedeeld. We noemen dit een multi-byte optelling. Stel dat we twee 16-bits getallen optellen. Dan moeten eerst de minst significante bytes worden opgeteld. Daarbij kan een uitgaande carry optreden (de uitgaande carry is 1). Bij de optelling van de meest significante bytes moet deze carry meegenomen worden. De uitgaande carry moet bij de eerste optelling dus even tijdelijk opgeslagen worden. Dat gebeurt in de *carry flag*.

De optelling van twee 16-bits getallen middels 8-bits eenheden is te zien in figuur 5.65. De carry flag wordt eerst op 0 gezet. Daarna worden de twee minst significante bytes opgeteld. De uitgaande carry wordt opgeslagen in de carry flag. Daarna worden de twee meest significante bytes opgeteld. De uitgaande carry wordt dan weer opgeslagen in de carry flag.



Figuur 5.65: *Optelling van twee 16-bits getallen middels twee optellingen van 8 bits.*

*5.21 Vermenigvuldigen en delen met two's complement

Het is mogelijk om schakelingen te ontwerpen om two's complement getallen te vermenigvuldigen en te delen. Dat is echter niet zo eenvoudig. Negatieve getallen zijn namelijk niet echt negatief, maar worden voorgesteld door de positieve variant van een macht van twee af te trekken. Zo wordt -1 met vier bits voorgesteld door 15. De vermenigvuldiging -1×-1 levert met de procedure in paragraaf 5.6 een verkeerd antwoord op.

Er zijn zeker algoritmen voor het vermenigvuldigen van two's complement getallen. Thijssen laat in [97] zien dat door de getallen via tekenuitbreiding met twee keer zoveel bits te representeren een vermenigvuldiger voor unsigned getallen gebruikt kan worden. Er wordt uitgegaan van even veel bits voor beide getallen. Het antwoord levert natuurlijk vier keer zoveel bits op dan de originele getallen, maar alleen de minst significante helft wordt opgeslagen.

Een andere mogelijkheid is om uit te gaan van de representatie volgens formule (5.17). Er kunnen nu gewone optellers worden gebruikt waarbij moet worden opgelet dat er extra bits nodig zijn om geen overflow te krijgen. De tekenbits moeten anders worden verwerkt omdat ze andere waarden hebben dan overige bits, namelijk 0 en -1 in plaats van 0 en $+1$.

Er zijn ook algoritmes die het heel anders doen. Bekend zijn het Booth-algoritme en het Modified Booth-algoritme. Deze zijn gebaseerd op het hercoderen van de getallen met positieve en negatieve gewichten. Het voert te ver om hier op in te gaan. Zie voor meer informatie [98, 99, 100, 101]

Een andere mogelijkheid is om negatieve getallen om te zetten naar positieve getallen (tekens onthouden) en dan de procedures voor unsigned getallen te gebruiken. Het antwoord moet eventueel weer omgezet worden naar een negatief getal. De procedure werkt niet voor het meest negatieve getal.

Dezelfde problematiek geldt ook voor delen. Eigenlijk is de signed magnitude representatie veel beter geschikt voor vermenigvuldigen en delen. Dan kunnen de procedures voor unsigned getallen gebruikt worden. De tekens van de antwoorden kunnen makkelijk bepaald worden.

5.22 One's complement representatie

In het begin van het computertijdperk werd ook nog een andere representatie voor getallen met teken gebruikt: het one's complement. Hierin werd een negatief getal weergegeven door alle bits van de positieve variant alleen te inverteren. Dat levert wel wat problemen op. Zo zijn er twee representaties van het getal 0, namelijk +0 ($0\cdots0_2$) en −0 ($1\cdots1_2$). Daarnaast blijkt dat een opteller een extra slag moet maken om het juiste resultaat te bereiken, de zogenoemde *end around carry* [102]. Dat komt omdat in het one's complement *modulo* 2^n-1 wordt gerekend [103]. Het voordeel mag wel duidelijk zijn: er zijn alleen inverters nodig om het one's complement te realiseren, er hoeft geen +1 bij opgeteld te worden. Dat levert behoorlijke snelheidswinst op bij grote getallen. Omzetten gaat sneller t.o.v. two's complement getallen maar bewerken gaat trager. Omdat een getal meestal vaker wordt bewerkt dan omgezet is two's complement meestal effectiever. Mede door het verwerken van de end around carry is het one's complement in ongebruik geraakt. Af en toe wordt de term nog gebruikt om aan te geven dat alle bits van een getal geïnverteerd moeten worden. Zie bijvoorbeeld de instructieset van de AVR microcontroller [104].

*5.23 Carry lookahead

Een 4-bits full adder ontworpen als ripple carry adder heeft als nadeel dat het veel tijd kost voordat c_4 beschikbaar is, ongeveer 8 poortvertragingen. Maar c_4 kan natuurlijk ook geschreven worden als functie van de ingangen a_3 t/m a_0, b_3 t/m b_0 en c_0. We stellen een waarheidstabel op met 512 regels en werken de functies voor c_4 en s_3 t/m s_0 uit. Dit levert echter een flinke schakeling op.

Slimmer is om uit te gaan van de functie van de carry uit functie (5.4). De uitgaande carry voor de minst significante 1-bit full adder kan geschreven worden als:

$$c_1 = a_0 \cdot b_0 + (a_0 + b_0) \cdot c_0 \tag{5.33}$$

Er worden nu twee hulpfuncties geïntroduceerd:

$$\begin{aligned} G_0 &= a_0 \cdot b_0 \\ P_0 &= a_0 + b_0 \end{aligned} \tag{5.34}$$

In de functies staat G staat voor *carry generate* want het genereert een uitgaande carry c_1 onafhankelijk van de inkomende carry c_0. P staat voor *carry propagate* want het geeft de inkomende carry c_0 door aan c_1.

De functie is nu te schrijven als:

$$c_1 = G_0 + P_0 \cdot c_0 \tag{5.35}$$

Maar dan kunnen we de functie van c_2 schrijven als:

$$c_2 = G_1 + P_1 \cdot c_1 \tag{5.36}$$

We vullen nu de functie van c_1 in:

$$\begin{aligned} c_2 &= G_1 + P_1 \cdot c_1 \\ &= G_1 + P_1 \cdot (G_0 + P_0 \cdot c_0) \\ &= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot c_0 \end{aligned} \tag{5.37}$$

En dan kan de functie voor c_3 ook uitgewerkt worden:

$$\begin{aligned} c_3 &= G_2 + P_2 \cdot c_2 \\ &= G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot c_0) \\ &= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot c_0 \end{aligned} \tag{5.38}$$

En natuurlijk uiteindelijk de functie voor c_4:

$$\begin{aligned} c_4 &= G_3 + P_3 \cdot c_3 \\ &= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot c_0) \\ &= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot c_0 \end{aligned} \tag{5.39}$$

De functie voor c_4 is nu te maken met AND- en OR-poorten met maximaal vijf ingangen. Samen met de P- en G-hulpfuncties levert dit een schakeling die maximaal drie poortvertragingen heeft. Deze realisatie van carry-propagatie heet *carry lookahead*. Het wordt onder andere gebruikt in de 4-bitz Full Adder ic 74283 [105].

Uit de functies van de carry's blijkt dat het aantal variabelen per productterm toeneemt. In de praktijk zijn AND- en OR-poorten met vier of vijf ingangen nog net te realiseren. Bij meer ingangen moeten alternatieve realisaties van de poorten gebruikt worden. Dat loont echter niet omdat de vertragingen dan groter worden.

Slimmer is om opnieuw het concept van de hulpfuncties G en P toe te passen. We kunnen inzien dat:

$$\begin{aligned} G_{0-3} &= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 \\ P_{0-3} &= P_3 \cdot P_2 \cdot P_1 \cdot P_0 \end{aligned} \tag{5.40}$$

zodat voor c_4 geschreven kan worden:

$$c_4 = G_{0-3} + P_{0-3} \cdot c_0 \tag{5.41}$$

Ook hier zijn G_{0-3} en P_{0-3} onafhankelijk van de carry's. Op vergelijkbare wijze kunnen nu de carry's voor c_8, c_{12} en c_{16} gerealiseerd worden. Een 16-bits full adder wordt dan met behulp van twee lagen *carry lookahead generators* gerealiseerd. Zie ook [106].

5.24 Opteller voor BCD-gecodeerde getallen

De eerder besproken binaire opteller kan alleen zuivere binaire getallen optellen (er wordt ook wel gesproken van normale binaire code = NBC). Optellen van BCD-getallen gaat niet direct. We laten dit zien aan de hand van een voorbeeld.

45		0100 0101		079		0000 0111 1001	
43	+	0100 0011	+	054	+	0000 0101 0100	+
88		1000 1000		133		0000 1100 1101	
dec		BCD		dec		? BCD ?	

Figuur 5.66: *Pogingen om BCD-gecodeerde getallen op te tellen met een binaire opteller.*

In figuur 5.66 zijn twee optellingen te zien. De getallen staan in de BCD-code en worden met behulp van een gewone binaire opteller opgeteld. Links worden de getallen 45 en 43 opgeteld met als resultaat 88. Dit antwoord klopt. Rechts is de optelling van 079 en 054 te zien. De uitkomst is 133 maar dat is niet te zien bij de binaire variant. Bij de posities van de tientallen en eenheden staan bitcombinaties die niet tot de BCD-code behoren.

Bij het optellen van twee BCD-cijfers en een inkomende carry kan het resultaat groter dan 9 zijn. Er moet dan een (BCD-)carry naar de volgende kolom[7] worden doorgegeven. We laten de correcte BCD-optelling van 079 en 054 zien in figuur 5.67. Tevens zijn ook de BCD- carry's te zien.

11			1		1		
079			0000		0111		1001	
054	+		0000		0101		0100	+
133			0001		0011		0011	

Figuur 5.67: *Correcte optelling van de BCD-getallen 079 en 054.*

Bij de eenheden wordt 1001_{BCD} bij 0100_{BCD} opgeteld. Het resultaat is $1\,0011_{\text{BCD}}$. We schrijven 0011_{BCD} in de kolom van de eenheden en zetten een 1 boven kolom van de tientallen. We tellen nu de tientallen bij elkaar inclusief de carry. Dat levert (toevallig) weer $1\,0011_{\text{BCD}}$ op. Er wordt een 1 boven de honderdtallen geplaatst. Het eindresultaat is $0001\,0011\,0011_{\text{BCD}}$.

Laten we nu de BCD-optelling van één kolom bekijken. We gaan uit van een *binaire* 4-bits optelling van de twee cijfers $A = a_3a_2a_1a_0$ en $B = b_3b_2b_1b_0$ en inkomende carry c_0. Het antwoord is $Z = z_4z_3z_2z_1z_0$, een 5-bits getal. Bit z_4 is de uitgaande carry dus die noemen we liever c_4. Aangezien geen van de aangeboden cijfers groter is dan 9, ligt het resultaat Z tussen 0 en 19, in binaire notatie tussen $0\,0000_2$ en $1\,0011_2$.

In tabel 5.6 zijn alle mogelijke binaire antwoorden en overeenkomstige BCD-antwoorden gegeven. Wanneer we naar de tabel kijken dan zien we dat het binaire antwoord identiek is aan het BCD-antwoord als het binaire antwoord kleiner is dan of gelijk is aan 9. Als het binaire antwoord groter is dan 9, dan krijgen we een incorrect antwoord en moet er een correctie worden toegepast. Vanuit de tabel is ook te zien dat de correctie kan worden uitgevoerd door 6 bij het binaire antwoord op te tellen. Dat komt omdat een binaire (4-bits) optelling modulo 16 (2^4) werkt en een decimale optelling modulo 10.

In figuur 5.68 zijn twee optellingen te zien. Links is de optelling van 6+7+0. Het binaire

[7] Een kolom bestaat uit vier bits.

Tabel 5.6: *Binaire waarden en bijhorende BCD-waarden.*

binair	BCD	dec	binair	BCD	dec
0 0000	0 0000	0	0 1010	1 0000	10
0 0001	0 0001	1	0 1011	1 0001	11
0 0010	0 0010	2	0 1100	1 0010	12
0 0011	0 0011	3	0 1101	1 0011	13
0 0100	0 0100	4	0 1110	1 0100	14
0 0101	0 0101	5	0 1111	1 0101	15
0 0110	0 0110	6	1 0000	1 0110	16
0 0111	0 0111	7	1 0001	1 0111	17
0 1000	0 1000	8	1 0010	1 1000	18
0 1001	0 1001	9	1 0011	1 1001	19

resultaat is 1101_2 maar dat is niet het juiste resultaat in BCD-notatie. In BCD-notatie is het juiste antwoord $1\,0011_{\text{BCD}}$. Rechts is de optelling van $9+9+1$ te zien. Het binaire antwoord is $1\,0011_2$. Het juiste antwoord in BCD-notatie is $1\,1001_{\text{BCD}}$. In beide gevallen moet het resultaat van de optelling gecorrigeerd worden door 6 op te tellen bij het binaire antwoord.

```
  0          0            1            1
  6       0110            9         1001
  7  +    0111       +    9    +    1001    +
 13       1101           19      1 0011
           0110  +                 0110    +
          1 0011                  1 1001
```

Figuur 5.68: *Twee optellingen met BCD-getallen waarbij correctie noodzakelijk is.*

We kunnen de BCD-optelling als volgt omschrijven:

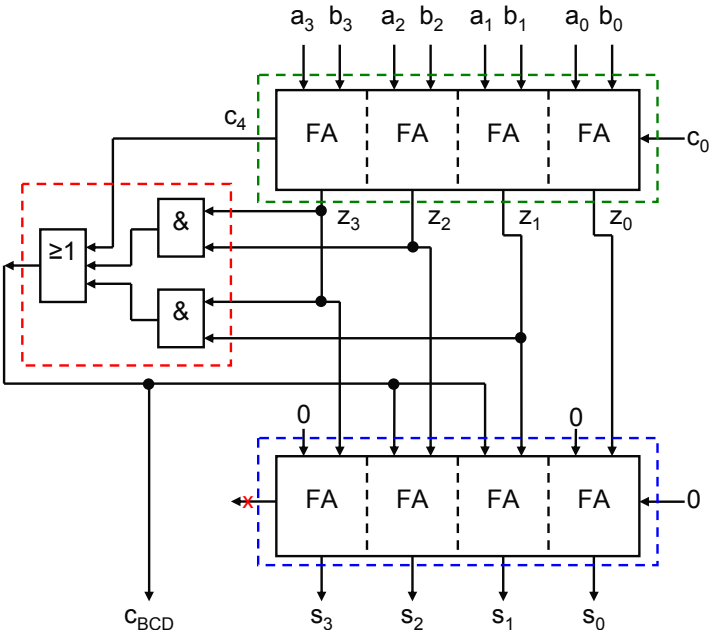
$$\begin{aligned} &Z = A + B + c_0 \\ &\text{Als } Z \leq 9 \text{ dan} \quad S = Z,\ c_{BCD} = 0 \\ &\text{Als } Z > 9 \text{ dan} \quad S = Z + 6,\ c_{BCD} = 1 \end{aligned} \tag{5.42}$$

Hierin is Z het binaire antwoord, S is het BCD-antwoord en c_{BCD} is de uitgaande BCD-carry. Er zijn twee gevallen waarin een correctie moet worden gemaakt: wanneer Z groter dan 9 maar geen uitgaande carry wordt gegenereerd met vier bits, en wanneer Z groter is dan 15 zodat een uitgaande carry wordt gegenereerd met vier bits. Er moet gecorrigeerd worden als $c_4 = 1$ of als $z_3 = 1$ en $z_2 = 1$ of $z_1 = 1$ (of beide). De voorwaarde voor een correctie en een uitgaande BCD-carry kan in een logische functie worden uitgedrukt:

$$c_{\text{BCD}} = c_4 + z_3 \cdot z_2 + z_3 \cdot z_1 \tag{5.43}$$

In figuur 5.69 is het schema te zien voor een BCD-opteller voor twee cijfers. Het bestaat uit twee gewone 4-bits binaire optellers en een detectieschakeling. Eerst worden de cijfers opgeteld met de bovenste opteller (inclusief de inkomende carry). De detectieschakeling

links signaleert of het resultaat van de optelling groter is dan 9. De correctie-opteller onderaan past de correctie toe als dat nodig is. Merk op dat de uitgaande carry van de correctie-opteller niet gebruikt wordt. De correctie-opteller kan verder nog geoptimaliseerd worden. Te zien is dat de full adder voor de eenheden gewoon weggelaten kan worden. Het telt immers twee nullen op bij de uitkomst van de eerste opteller. De full adder voor de tweetallen kan vervangen worden door een half adder. Hier worden namelijk maar twee bits bij elkaar opgeteld. De full adder voor de achttallen kan vervangen worden door een EXOR-poort met twee ingangen die alleen de sombit bepaalt.



Figuur 5.69: *Opteller voor een BCD-cijfer.*

Het is niet nodig om voor de correctie een 5-bits opteller te gebruiken. Een 4-bits opteller is genoeg omdat $1\,0000_2$ (16), $1\,0001_2$ (17), $1\,0010_2$ (18) en $1\,0011_2$ (19) de enige resultaten zijn die groter zijn dan 15. De viertallen en achttallen zijn 0 zodat er bij correctie nooit een uitgaande carry wordt gegenereerd. Zie ook tabel 5.6. De uitgaande BCD-carry komt van de detectieschakeling.

*5.25 Ten's complement

De complement representatie is ook voor andere talstelsels te gebruiken. Zo kan de decimale talstelsel gebruikt worden met de *ten's complement* representatie. Het ten's complement van een positief getal met bijvoorbeeld vijf cijfers wordt gemaakt door het getal af te trekken van 10^5. In figuur 5.70 is te zien hoe het nemen van de ten's complement representatie met vijf cijfers van -1234 wordt uitgevoerd.

$$\begin{array}{r} 100000_{10} \\ 01234_{10} \\ \hline 98766_{10} \end{array} \quad - \qquad \begin{array}{r} 10^5 \\ +1234 \\ \hline 10^5 - 1234 \end{array}$$

Figuur 5.70: *Ten's complement representatie van het getal* -1234 *met vijf decimale cijfers.*

Slimmer is om het getal +1234 af te trekken van $10^5 - 1$. Dit levert namelijk allemaal negens op. Per kolom hoeft er dan niet geleend te worden. Er moet nog wel +1 bij worden opgeteld om het juiste resultaat te krijgen. Dit is te zien in figuur 5.71.

$$\begin{array}{rcl} 99999_{10} & & 10^5-1 \\ 01234_{10} & - & +1234 \\ \hline 98765_{10} & & \\ 1_{10} & + & \\ \hline 98766_{10} & & -1234 \end{array}$$

Figuur 5.71: *Ten's complement representatie van het getal* −1234*.*

We kunnen de getallen nu gebruiken in optellingen. Aftrekkingen worden gedaan door uit te gaan van de gelijkheid $A-B=A+(-B)$. Zie figuur 5.72.

$$\begin{array}{rcrcrcrcr} +2 & & 00002 & & +1849 & & 01849 & & \\ +1 & - & 99999 & + & +679 & - & 99321 & + & \\ \hline -1 & & 1 \mid 00001 & & +1170 & & 1 \mid 01170 & & \\ & & & & & & & & \\ +457 & & 00457 & & -3456 & & 96544 & & \\ +2345 & - & 97655 & + & +1234 & - & 98766 & + & \\ \hline -1888 & & 98112 & & -4690 & & 0 \mid 95310 & & \end{array}$$

Figuur 5.72: *Enkele aftrekkingen in de ten's complement representatie.*

De signed magnitude representatie kan een onbeperkt aantal getallen weergeven omdat we een onbeperkt aantal cijfers kunnen gebruiken. Het bereik is onbegrensd. Bij het ten's complement is het bereik wel begrensd omdat we moeten afspreken in welk cijfer het teken gecodeerd wordt. Dat houdt in dat getallen met vijf decimalen alle berekeningen *modulo* 10^5 worden uitgevoerd. Dit geeft een bereik van −50000 (50000) tot en met +49999 (49999). Er is dus geen tegengesteld getal voor −50000. Het bereik is asymmetrisch. Opgemerkt moet worden dat ook in de ten's complement representatie overflow kan voorkomen. We gaan hier niet verder op in. Ten's complement wordt gebruikt bij BCD-rekenschakelingen maar is in ongebruik geraakt door het gebruik van de two's complement representatie.

*5.26 Fixed point en floating point representaties

Tot nu toe hebben we alleen met gehele getallen gewerkt. Getallen kunnen echter ook een fractie hebben. We bespreken twee vormen om getallen met fracties weer te geven, de *fixed point* en *floating point* representaties.

Bij de fixed point representatie wordt de komma op een vaste plaats in het getal gepositioneerd. De getallen hebben dan een vaste lengte voor het gehele deel en de fractie. De ontwerper kan deze positie zelf kiezen. Het voordeel van de fixed point representatie is dat de ontwikkelde hardware voor de vier basisoperaties van gehele getallen kan worden gebruikt. Ook de two's complement representatie kan gewoon worden gebruikt. Het

nadeel is dat het bereik is begrensd en dat kan tot overflow leiden. Er moet daarnaast worden afgerond bij vermenigvuldigen en delen. Bij vermenigvuldigen is de lengte van het resultaat de som van de lengtes van de getallen. Er zal dan een aantal bits van het gehele deel en de fractie moeten worden geschrapt. Dat geldt ook voor delen. Voor meer informatie over fixed-point getallen, zie [107].

Het bereik tussen hele grote getallen en hele kleine getallen is met de fixed point representatie beperkt. Om dit probleem op te vangen wordt de floating point representatie gebruikt. De representatie kent een teken, een *mantisse* (fractie) en een exponent. Een decimaal getal wordt voorgesteld door:

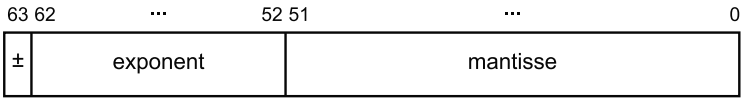
$$M = \pm\text{mantisse} \cdot 10^{\text{exponent}} \tag{5.44}$$

Als voorbeeld geven we het onderstaande getal:

$$17.040.000 = 1704 \cdot 10^4 = 1{,}704 \cdot 10^7 = 0{,}1704 \cdot 10^8$$

Het wordt nu duidelijk dat een getal op meer dan één manier kan worden voorgesteld. Dit leidt in de praktijk tot veel conversies. Om ervoor te zorgen dat er slechts éé representatie van getal gebruikt wordt, zorgen we ervoor dat de mantisse van het getal altijd tussen 0,1 en 0,999... ligt. De exponent van de 10-macht neemt dus toe of af naar gelang de cijfers van de mantisse naar links of naar rechts geschoven worden. Als de mantisse tussen 0,1 en 0,999... ligt, noemen we het floating point getal *genormaliseerd*. Verder moet worden opgemerkt dat de mantisse een eindig aantal cijfers bevat. De nauwkeurigheid is dus ook eindig. In wezen scheidt de representatie het bereik van de nauwkeurigheid [108].

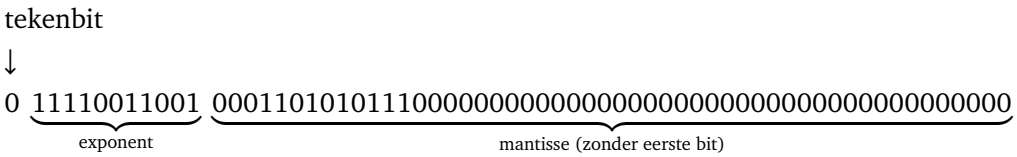
Voor floating point getallen in het binaire talstelsel is een standaard ontworpen om ervoor te zorgen dat er één representatie wordt gebruik. De ANSI/IEEE 754 standaard [109] kent onder andere een 32-bits *single format* en een 64-bits *double format*. We laten alleen het 64-bits formaat zien, zie figuur 5.73.



Figuur 5.73: *Formaat van een 64-bits floating point getal.*

De linker bit geeft het teken van het getal aan. Voor een positief getal is deze bit 0 en voor een negatief getal is deze bit 1. De volgende 11 bits zijn gereserveerd voor de exponent. Het exponent stelt een macht van twee voor, we werken immers met bits. Een exponent kan ook negatief zijn maar er is geen tekenbit beschikbaar. Dit is ondervangen door de exponent met 1023 te verhogen. Dit wordt *Excess-1023* genoemd. Alle exponenten zijn dus 1023 te groot. De overige 52 bits zijn bestemd voor de mantisse. Deze mantisse wordt *genormaliseerd*, d.w.z. dat de mantisse naar links of naar rechts wordt opgeschoven totdat het eerste bit links van de komma 1 is. De mantisse is dus altijd te representeren als $1{,}m_{-1}m_{-2}m_{-3}\cdots$ zodat deze 1 kan worden weggelaten. Voor het getal 0 geeft dit een probleem, daar is in de standaard een speciale voorziening voor getroffen.

We geven de representatie van het binaire getal $+1000110{,}1010111 \cdot 2^{1110010100}$ als voorbeeld. In figuur 5.74 is het getal in het 64-bits formaat te zien. Eerst is het getal genormaliseerd. De komma wordt dus zes plaatsen naar links verschoven. De exponent wordt dus met zes verhoogd. Daarna wordt bij de exponent 1023 opgeteld vanwege de Excess-1023 notatie.



Figuur 5.74: *Voorbeeld van een 64-bits floating point getal.*

Verder moet opgemerkt worden dat decimale getallen niet altijd exact met een binair floating point getal kan worden worden weergegeven. Zo is het decimale getal 3,14 niet perfect te reproduceren. Omrekenen naar 64-bits floating point binair en terugrekenen naar decimaal levert $3{,}14000000000000124\cdots$ op. Hieraan is te zien dat het getal met ongeveer 16 significante decimale cijfers wordt opgeslagen. Het vertelt ons ook dat met veelvuldig rekenen met floating point getallen de accuratesse erbij in schiet. Zie [110] voor een inleiding op floating point accuratesse.

Digitale schakelingen voor floating point getallen leveren veel hardware op. Ook de vertragingstijden van de schakelingen zijn aanzienlijk. Het beste is om floating point getallen te vermijden. Zie voor meer informatie over floating point [111].

5.27 Opgaven

5.1. Tel de volgende unsigned binaire getallen op:

$1011001 + 0111011 \qquad 01010 + 01010 \qquad 01111 + 00001$

* **5.2.** Voer de onderstaande unsigned aftrekkingen uit:

$1011001 - 0111011 \qquad 110000 - 010011 \qquad 010110 - 100101$

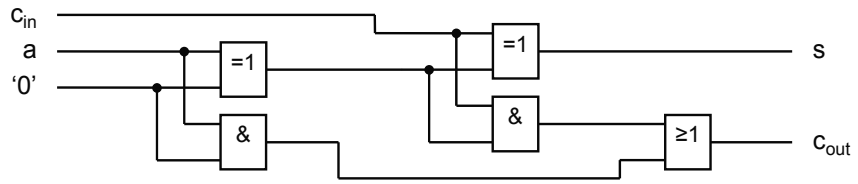
5.3. Toon aan dat: $c_{out} = \overline{c_{in}} \cdot (a \cdot b) + c_{in} \cdot (a + b) = a \cdot b + a \cdot c_{in} + b \cdot c_{in}$

5.4. Toon aan dat: $c_{out} = a \cdot b + a \cdot c_{in} + b \cdot c_{in} = a \cdot b + c_{in} \cdot (a \oplus b)$

5.5. Als de functies van s en c_{out} vanuit de nullen zouden worden gemaakt, wordt de functie dan kleiner (minder poorten)?

5.6. In figuur P5.1 is de full adder die eerder is besproken nog eens afgebeeld, maar nu is de b-ingang aan een logische 0 gekoppeld. Vereenvoudig het schema ("minimaliseer b weg"). Doe hetzelfde voor b is logisch 1.

5.7. Ontwerp een 2x2-bits unsigned vermenigvuldiger. Stel de waarheidstabel op en leid de schakelfuncties af.



Figuur P5.1: *Full adder met ingang b aangesloten op een logische 0.*

5.8. Ontwerp een schakeling die test of twee unsigned 4-bits getallen gelijk zijn.

5.9. Ontwerp een 4x4 bits carry save multiplier (tip: uiteraard heeft iemand dat allang gedaan).

5.10. Hoeveel optellers zijn er nodig voor een 5x3-bits vermenigvuldiger? Hoe breed zijn de optellers?

5.11. Reken om van decimaal naar 8-bits two's complement:

$+5, \quad -5, \quad -100, \quad -64, \quad +25, \quad -25$

5.12. Reken uit in two's complement en let daarbij op overflow. Gebruik tekenuitbreiding om de getallen uit evenveel bits te laten bestaan.

$011001 + 100100$	$110011 + 100011$
$011001 - 001110$	$101010 - 010101$
$1101 + 111001$	$101011 - 1100111$
$011100 - 011001$	$10001 + 1100111$

5.13. Geef het decimale equivalent van de volgende hexadecimale two's complement getallen:

$FFFF_{16} \quad 53BA_{16} \quad CB_{16} \quad 7EA3_{16}$

5.14. Wat is de kleinste decimale waarde en de grootste decimale waarde van een hexadecimaal two's complement getal van 8 cijfers?

5.15. Schrijf als hexadecimale two's complement getallen met vier cijfers. Maak gebruik van tekenuitbreiding.

$F_{16} \quad 6_{16} \quad 7A_{16} \quad CB_{16} \quad 35B_{16} \quad D73_{16}$

* **5.16.** Negatieve getallen in BCD-formaat worden weergegeven in ten's complement. Dit is rekenkundig te doen door het BCD-getal af te trekken van 9...9 en daarna er 1 bij op te tellen (uiteraard met een BCD-opteller). Dit aftrekken van 9...9 is het zogenoemde nine's complement. Het voordeel van hiervan is dat per kolom niet geleend hoeft te worden. Ontwerp een digitale schakeling waarin een aangeboden BCD-cijfer wordt afgetrokken van 9. Stel een waarheidstabel op, minimaliseer de functies m.b.v. Karnaughdiagrammen en teken de schakeling van de functies met poorten naar eigen keuze.

5.17. Een nadeel van two's complement is dat het bereik asymmetrisch is, bijvoorbeeld -8 t/m $+7$. Is het mogelijk om met een 4-bits FA toch $+8$ op te tellen bij een getal? Motiveer het antwoord.

5.18. Tel de volgende BCD-gecodeerde getallen bij elkaar op:

10011001 + 00111001 0011 + 0111 010110000110 + 000001110001

5.19. Het probleem van BCD-getallen is dat er per BCD-cijfer zes binaire codecombinaties niet gebruikt worden. De bekende excess-3-code is een methode om snellere BCD-optellers te maken. Van elk BCD-cijfer kan het excess-3-cijfer worden gemaakt door bij het BCD-cijfer 3 op tellen dus 4 (0100) wordt dan 7 (0111). Ontwerp een digitale schakeling voor de BCD-to-excess-3 encoder. Wat gebeurt er als alle bits van de (geldige) excess-3-code-cijfer worden geïnverteerd?

* **5.20.** Een ouderwetse auto doet mee aan een achteruitrijrace. De kilometerteller staat op 0. De auto rijdt 123 kilometer achteruit. Wat is de kilometerstand als de kilometerteller vijf decimale cijfers heeft? Ga ervan uit dat de kilometerteller terug telt.

* **5.21.** Een 32-bits single format floating point getal heeft een mantisse van 23 bits. Toon aan dat de decimale variant van de mantisse met ongeveer 7 significante cijfers kunnen worden weergegeven.

6

Geheugenschakelingen

Tot nu toe zijn alleen combinatorische schakelingen behandeld. Dit zijn schakelingen waarbij de uitgangswaarden alleen afhankelijk zijn van de ingangswaarden. Een verandering op de ingangen zorgt voor een verandering aan de uitgangen. Uiteraard kost het de schakeling enige tijd om bij een bepaalde ingangswaarde de bijbehorende uitgangswaarde de produceren.

Schakelingen die geheugenwerking vertonen worden *sequentiële schakelingen* genoemd. De uitgangswaarden van deze schakelingen zijn niet alleen afhankelijk van de ingangswaarden maar ook van de inhoud of *stand* van het geheugen. Een kenmerk van sequentiële schakelingen is dat de uitgangen meerdere uitgangscombinaties kunnen hebben bij gelijke ingangscombinaties.

In dit hoofdstuk behandelen we verschillende typen geheugenelementen:

- een (direct werkende) SR-latch;
- een gated latch (SR- en D-);
- flankgevoelige flipflop (D- en JK-), (master-slave).

De SR-latch is het meest eenvoudig te realiseren geheugenelement. Het tijdgedrag geeft echter wel problemen bij het gebruik. Dat heeft geleid tot de gated latches. Ook hier

NAMING AND SHAMING

Vooruitlopend op de tekst merken we op dat de term *latch* uitsluitend gebruikt wordt voor geheugenelementen die op signaalniveaus reageren, de zogenoemde level-triggered latches. De term *flipflop* wordt uitsluitend gebruikt voor geheugenelementen die op signaalflanken reageren, de zogenoemde edge-triggered flipflops.

In veel Engelstalige boeken worden alle geheugenelementen flipflops genoemd. Dat leidt gemakkelijk tot verwarring.

leidt het tijdgedrag tot problemen. Dat heeft geresulteerd in de flankgevoelige flipflops.

Van de genoemde geheugenelementen worden de opbouw, de logische werking en het gedrag in tijd besproken. We introduceren voor dit laatste het timingdiagram.

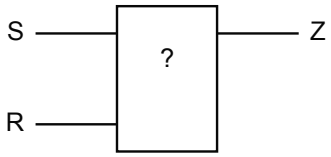
6.1 SR-latch

Algemeen gezegd heeft een eenvoudig geheugenelement de volgende acties:

- Een actie waarbij de stand van het geheugenelement op 1 gezet (set) wordt.
- Een actie waarbij de stand van het geheugenelement op 0 gezet (reset) wordt.
- Een actie waarbij de laatst ingebrachte stand van het geheugenelement onthouden wordt.

Deze drie acties kunnen met twee (stuur-)signalen gecodeerd worden. De namen van deze twee signalen zijn S (set) en R (reset). Het mag wel duidelijk zijn dat deze namen een directe relatie met de acties hebben. Met de twee signalen zijn vier combinaties mogelijk, dus één combinatie wordt niet gebruikt. De uitgang wordt meestal Z genoemd, maar sommige boeken gebruiken ook wel Q. We kunnen het geheugenelement tekenen als een black box. Dit is te zien in figuur 6.1.

Van de te ontwerpen *SR-latch* (grendel) kan een functietabel worden opgesteld, zie tabel 6.1. Merk op dat er altijd een combinatie van S en R wordt aangeboden om de SR-latch in de goede stand te krijgen. Er zijn drie combinaties nodig. In feite is de codering vanzelfsprekend. De combinatie $SR = 00$ zorgt ervoor dat de latch de laatst ingebracht stand onthoudt. De combinatie $SR = 10$ wordt gebruikt om de latch op 1 te zetten (set). De combinatie $SR = 01$ wordt gebruikt om de latch op 0 te zetten (reset). De combinatie $SR = 11$ wordt (voorlopig) niet gebruikt.



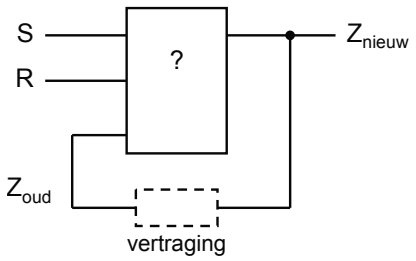
Figuur 6.1: *SR-latch als black box.*

Tabel 6.1: *Functietabel SR-latch.*

S	R	functie
0	0	onthouden
0	1	reset
1	0	set
1	1	niet gebruikt

Bij de set- en resetactie moet de stand van de latch worden aangepast, dus de huidige stand wordt vervangen door een nieuwe stand. Bij de onthoudactie moet de huidige stand behouden blijven, dus de nieuwe stand is hetzelfde als de huidige stand. Kijken we naar de uitgang Z, dan wordt er ook wel gesproken van de oude (huidige) waarde Z_{oud} en de nieuwe waarde Z_{nieuw}.

We kunnen de schakeling voor de SR-latch modelleren door middel van combinatorische logica en een vertraging. Dit is te zien in figuur 6.2. De te ontwerpen schakeling heeft dus eigenlijk drie ingangen S, R en Z_{oud} en één uitgang Z_{nieuw}. De realisatie is gebaseerd op het Huffman-model [112] voor sequentiële schakelingen. Te zien is dat de uitgang



Figuur 6.2: *Realisatiemodel van de SR-latch.*

Tabel 6.2: *Waarheidstabel van de SR-latch.*

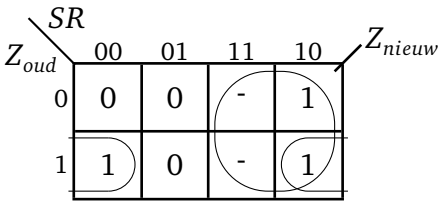
S	R	Z_{oud}	Z_{nieuw}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	-
1	1	1	-

Z_{nieuw} via een *terugkoppeling* verbonden is met ingang Z_{oud}. Dit is kenmerkend voor een schakeling met geheugenwerking. In de terugkoppeling is een vertragingselement opgenomen. De vertraging zorgt ervoor dat de waarde van Z_{oud} nog beschikbaar is totdat de schakeling de nieuwe waarde heeft gerealiseerd. In de praktijk hebben poorten altijd enige traagheid zodat de expliciete vertraging weggelaten kan worden.

In tabel 6.2 is de specificatie van de SR-latch weergegeven. In de eerste twee regels zijn de stuursignalen S en R allebei 0, de codering voor onthouden. De uitgang Z_{nieuw} volgt dus Z_{oud}. Bij de derde en vierde regel ($SR = 01$) wordt de SR-latch op 0 gezet (reset), de nieuwe waarde van Z is 0. Bij de codering voor het op 1 zetten van de latch ($SR = 10$) in de vijfde en zesde regel wordt de uitgang Z op 1 gezet. Zoals afgesproken wordt de combinatie $SR = 11$ niet gebruikt. De uitgang wordt als don't care gespecificeerd.

SR-latch met overheersende set

Met behulp van een Karnaughdiagram is de formule van Z_{nieuw} uit te werken, zie figuur 6.3.



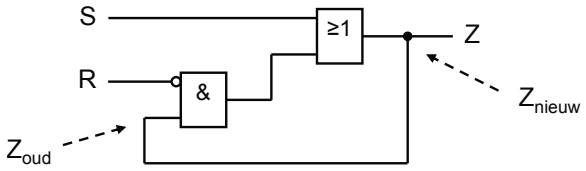
Figuur 6.3: *Karnaughdiagram van de SR-latch.*

Bij het uitwerken van het Karnaughdiagram worden de don't cares meegenomen bij de omranding van de twee enen aan de rechterkant. Dit levert de meest eenvoudige functie voor Z_{nieuw} op. Duidelijk is te zien dat Z_{nieuw} een functie is van Z_{oud}:

$$Z_{nieuw} = S + \overline{R} \cdot Z_{oud} \tag{6.1}$$

De combinatie $SR = 11$ zorgt er nu voor dat de nieuwe waarde van de SR-latch op 1 gezet wordt, identiek aan de set-operatie. Dit is een SR-latch met *overheersende set*.

Vanuit de functie kan het schema van de latch getekend worden. Zie figuur 6.4. De schakeling is te realiseren met slechts twee poorten als we de inverse voor R niet meerekenen. Kenmerkend voor de schakeling van de SR-latch is de *lus*. Deze lus loopt van de uitgang Z (Z_{nieuw}) naar de onderste ingang van de AND-poort (Z_{oud}), via de uitgang van de AND-poort naar de onderste ingang van de OR-poort en eindigt dan aan de uitgang van de OR-poort. Zoals al eerder is gezegd, wordt de vertraging niet expliciet opgenomen in de schakeling.

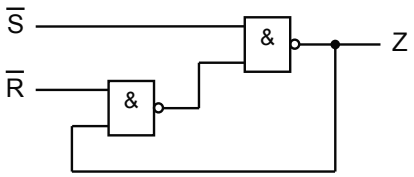


Figuur 6.4: *Schema van de SR-latch met overheersende set.*

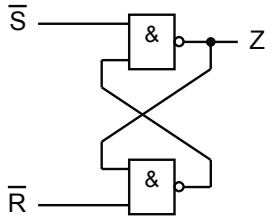
Met behulp van de stellingen van De Morgan is de functie van de SR-latch om te werken tot een NAND-NAND-vorm:

$$Z_{nieuw} = S + \overline{R} \cdot Z_{oud} = \overline{\overline{S} \cdot \overline{\overline{R} \cdot Z_{oud}}} \tag{6.2}$$

De schakeling is nu te bouwen met twee NAND-poorten. Voor de stuursignalen S en R zijn wel de inversen nodig. Dat is gewoonlijk niet bezwaarlijk. Realisatie van deze vorm is te zien in figuur 6.5(a). De variant in figuur 6.5(b) wordt doorgaans in de literatuur gebruikt.



(a) *Realisatie met NAND-poorten.*



(b) *Alternatieve tekenwijze.*

Figuur 6.5: *NAND-NAND realisatie van de SR-latch.*

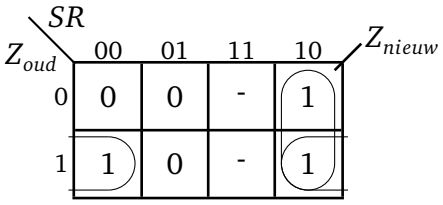
SR-latch met overheersende reset

We kunnen het Karnaughdiagram opnieuw uitwerken maar nu worden de don't cares ingevuld met nullen, zie figuur 6.6.

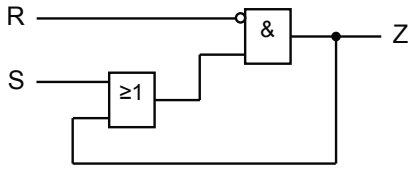
De ingangscombinatie SR = 11 wordt na uitwerking gebruikt als resetoperatie. Dit geheugenelement heeft een *overheersende reset*. De functie is:

$$Z_{nieuw} = \overline{R} \cdot S + \overline{R} \cdot Z_{oud} = \overline{R} \cdot (S + Z_{oud}) \tag{6.3}$$

Door gebruik te maken van de distributieve wet kan $\overline{R}$ buiten de haakjes gehaald worden zodat we een product-van-sommen-vorm krijgen. Ook deze schakeling is met slechts twee poorten te realiseren als we de inverse voor R niet meerekenen. Zie figuur 6.7.



Figuur 6.6: *Karnaughdiagram van de SR-latch.*



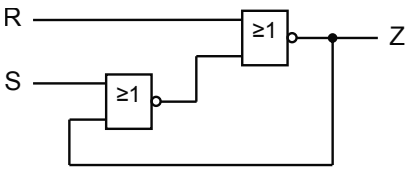
Figuur 6.7: *Schema van de SR-latch met overheersende reset.*

Als we goed kijken dan zien we dat de AND- en OR-poorten in feite van plaats verwisseld zijn. Ook de stuursignalen S en R zijn van plaats verwisseld.

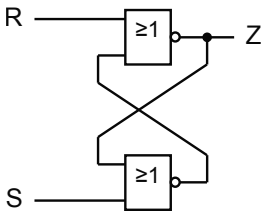
De functie voor de SR-latch met overheersende reset kan worden omgewerkt tot een NOR-NOR-vorm:

$$Z_{nieuw} = \overline{R} \cdot (S + Z_{oud}) = \overline{R + \overline{S + Z_{oud}}} \tag{6.4}$$

Voor het realiseren van deze schakeling zijn nu twee NOR-poorten nodig. Voor de stuursignalen zijn *geen* inversen nodig. Realisatie van deze vorm is weergegeven in figuur 6.8(a). De variant in figuur 6.8(b) wordt doorgaans in de literatuur gebruikt maar dan soms zó getekend dat signaal S weer boven staat.



(a) *Realisatie met NOR-poorten.*



(b) *Alternatieve tekenwijze.*

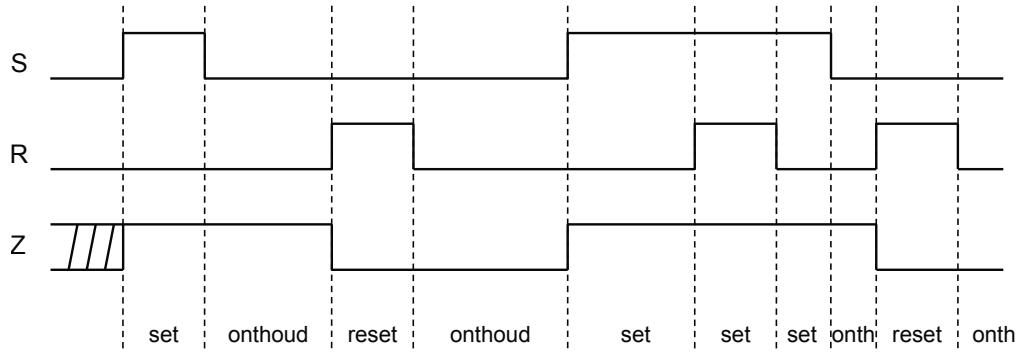
Figuur 6.8: *NOR-NOR realisatie van de SR-latch*

De ingangscombinatie SR = 11

In veel literatuur wordt de combinatie $SR = 11$ de *verboden stand* genoemd. Deze combinatie is dan wel niet verboden, maar wordt in de praktijk nauwelijks gebruikt. Alle benodigde operaties van de SR-latch zijn immers voor handen. Daarnaast levert het problemen op als van $SR = 11$ naar $SR = 00$ gegaan wordt. Hiervoor zijn twee bitwisselingen tegelijk nodig. Dit is in de praktijk nooit goed te realiseren (zie paragraaf 2.13). In de praktijk wisselt een van de twee stuursignalen het eerst zodat onbedoeld een set- of reset-operatie kan worden gestart. Voor de realisatie van een goedwerkende SR-latch

mag tussen de codering van de set- en onthoudopdracht én tussen de codering van de reset- en onthoudopdracht slechts één bitwisseling verschil zitten.

De werking van een SR-latch is te demonstreren met een signaaldiagram. In een signaaldiagram staan de signalen en bijbehorende waarden op de y-as genoteerd, op de x-as is de voortschrijdende tijd genoteerd. In tegenstelling tot een tijdvolgordediagram of timingdiagram wordt bij een signaaldiagram geen inzicht gegeven in de exacte vertragingstijden maar alleen wat de uitgangswaarden worden als gevolg van veranderingen van de ingangssignalen. In feite geeft een signaaldiagram de logische werking van de schakeling weer.



Figuur 6.9: *Signaaldiagram van de SR-latch met overheersende set.*

In figuur 6.9 is een signaaldiagram weergegeven. Het diagram begint met de waarden $SR = 00$, de combinatie voor onthouden. De voorgeschiedenis van uitgang Z is niet bekend. Dit wordt aangegeven met de twee logische waarden 0 en 1 met daartussen een gearceerd gebied. Vervolgens wordt ingang S op 1 gezet, R blijft 0. Dit is de combinatie door de set-opdracht zodat de uitgang $Z = 1$ wordt. Hierna wordt S weer op 0 gezet (R was al 0), waardoor het SR-latch de laatst ingebrachte stand onthoudt. Uitgang Z was op 1 gezet en blijft dus 1. Vervolgens wordt $R = 1$ (S blijft 0). Dit is de combinatie voor de reset-opdracht. De uitgang wordt 0. Daarna wordt R weer 0 en zal de latch de ingebrachte 0 onthouden. Vervolgens wordt S weer op 1 gezet dat ervoor zorgt dat Z weer 1 wordt. De enige overgebleven, niet gebruikte combinatie, is $SR = 11$. De uitgang blijft 1 omdat het hier de signaaldiagram betreft van een SR-latch met overheersende set.

In het signaaldiagram is het niet mogelijk om de werking van de SR-latch af te beelden als $SR = 11$ naar $SR = 00$ gaat. De interne werking ligt niet vast (we kunnen uitgaan van het schema in figuur 6.4 maar het kan zijn dat de latch intern anders is opgebouwd) zodat het niet mogelijk is om een uitspraak te doen over de nieuwe waarde van Z.

Wat opvalt in het signaaldiagram is dat de uitgang *verschillende* uitgangswaarden heeft bij de ingangswaarden $SR = 00$. Dat kan alleen als de schakeling *geheugen* bezit.

6.2 Gated SR-latch

SR-latches zijn *transparant* voor hun ingangssignalen. Eventuele veranderingen van de ingangen hebben direct een gevolg voor de stand van de latch en de waarde van

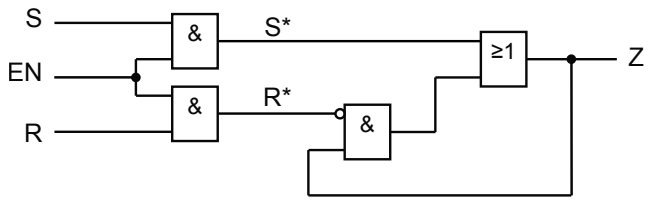
de uitgang. Als de SR-latch onderdeel is van een grotere schakeling dan worden de signalen voor S en R opgewekt door combinatorische schakelingen. Deze schakelingen kunnen *hazards* (zie paragraaf 4.11) op de uitgangen vertonen, waardoor onbedoelde combinaties voor S en R worden aangeboden aan de latch. De SR-latch reageert hier direct op.

Beter is het om de ingangen van een SR-latch ongevoelig te maken voor deze verandering door te wachten totdat de signalen S en R een definitieve waarde hebben. Hiertoe breiden we de SR-latch uit met een enable-signaal (EN). De enable werkt als een toegangscontrole. Als het enable-signaal logisch 0 is, houdt de latch de huidige stand vast (onthouden), ook al veranderen S en/of R. De latch is dan vergrendeld. Als het enable-signaal logisch 1 is, heeft de latch de werking van een gewone SR-latch. De latch is dan transparant. Dit wordt een *gated SR-latch* genoemd. Een gated latch wordt een *niveaugevoelig* of *level-triggered* geheugenelement genoemd.

Bij een SR-latch op basis van AND en OR (zie figuur 6.4) is ingangscombinatie $SR = 00$ de onthoudactie. Als het enable-signaal logisch 0 is, moet de aangeboden ingangscombinatie voor S en R geblokkeerd en vervangen worden door de ingangscombinatie $SR = 00$. Dit kan gerealiseerd worden door AND-poorten. De functies worden dan:

$$\begin{aligned} S^* &= S \cdot EN \\ R^* &= R \cdot EN \end{aligned} \tag{6.5}$$

Hierin zijn S^* en R^* de directe ingangen van de SR-latch zoals beschreven in paragraaf 6.1 en S en R de ingangen van de gated SR-latch. Op basis van de formules in (6.5) en het schema van de SR-latch in figuur 6.4 kunnen we het schema van de gated SR-latch opstellen. Dit schema is te zien in figuur 6.10.

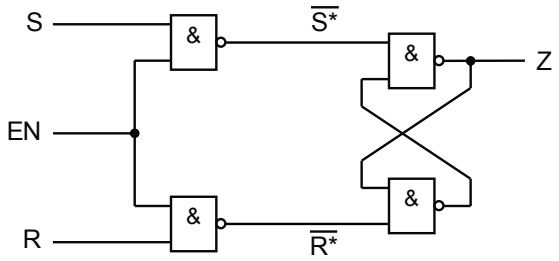


Figuur 6.10: *Gated SR-latch met AND en OR.*

Tijdens $EN = 0$ is de latch niet meer transparant. Ingangsveranderingen op S en R worden genegeerd tot in de periode dat $EN = 1$. Tijdens $EN = 1$ werkt de gated SR-latch als een gewone SR-latch.

Het is mogelijk om de gated SR-latch met alleen NAND-poorten te realiseren. Het voordeel van deze realisatie is dat de inversen van de stuursignalen S^* en R^* niet meer nodig zijn want die worden al gegenereerd door de twee linker NAND-poorten. Zie figuur 6.11.

Verder moet opgemerkt worden dat een gated SR-latch ook kan worden opgebouwd op basis van NOR-poorten. Er zijn dan uiteraard vier NOR-poorten nodig. Bij deze latch is de enable actief laag en moeten de stuursignalen R en S geïnverteerd worden aangeboden. We behandelen deze latch niet. Zie voor een korte uiteenzetting [113].



Figuur 6.11: *Gated SR-latch met NAND-poorten.*

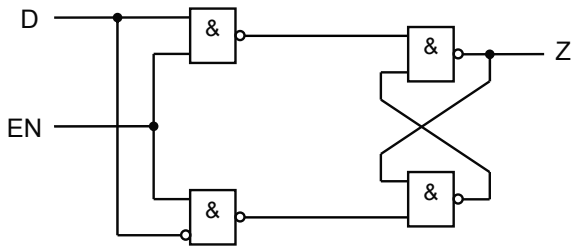
6.3 Gated D-latch

De nu ontwikkelde gated SR-latch heeft eigenlijk vijf ingangscombinaties die tot onthouden leiden:

- $SR = 00$ tijdens $EN = 1$
- $EN = 0$, hierbij kunnen S en R als don't care gespecificeerd worden.

Deze observatie levert de mogelijkheid om de gated SR-latch te vereenvoudigen. We zijn immers alleen geïnteresseerd in een schakeling die een ingebrachte waarde kan onthouden of een nieuwe waarde kan aannemen. Uit de opsomming blijkt dat de combinatie $SR = 00$ geschrapt kan worden, want die wordt overgenomen door $EN = 0$. De combinatie $SR = 11$ wordt niet gebruikt en kan ook geschrapt worden. Er blijven twee combinaties over: $SR = 01$ voor reset en $SR = 10$ voor set (dit alles tijdens $EN = 1$ natuurlijk).

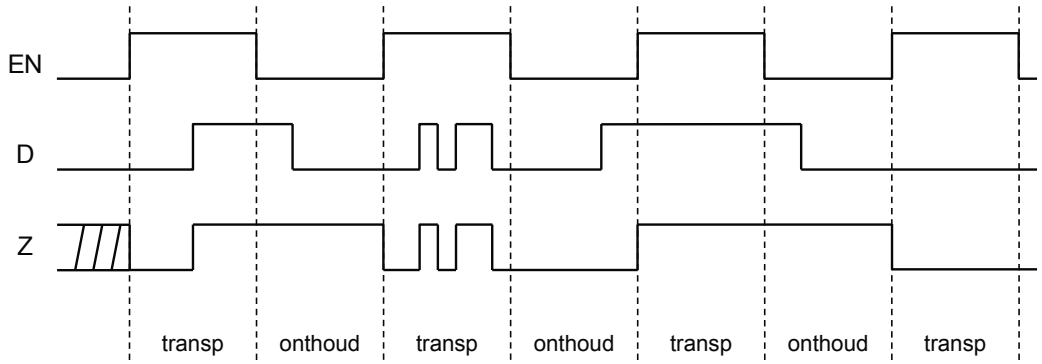
Nu blijkt dat S en R complementair zijn. De waarde van R is te realiseren door de inverse van S te nemen. De namen S en R verdwijnen (er is immers nog maar één ingang over) en wordt nu D genoemd[1]. De in figuur 6.12 gepresenteerde schakeling is opgebouwd met NAND-poorten heet *gated D-latch*. Merk op dat er nog een NOT-poort nodig is voor de inverse van D.



Figuur 6.12: *Gated D-latch met NAND-poorten.*

In figuur 6.13 is een signaaldiagram te zien van een gated D-latch met een actief hoge enable. Het signaaldiagram geeft de logische werking van de latch weer, het geeft geen inzicht in de timing. Als het signaal $EN = 1$ is, volgt de uitgang Z de ingang D want de latch is transparant. Als $EN = 0$ is, dan is de latch gesloten en onthoudt het de laatst ingebrachte waarde op de D-ingang.

[1] Het is niet geheel duidelijk waar de naam D vandaan komt. Er zijn twee mogelijke bronnen: van het woord *data* en van het woord *delay*.



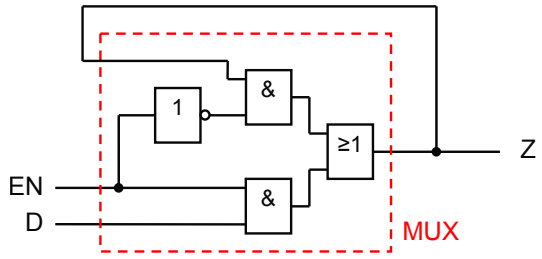
Figuur 6.13: *Signaaldiagram van een gated D-latch met een actief hoge enable.*

*6.4 Gated D-latch met multiplexer

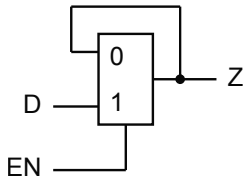
We kunnen een gated D-latch ook op een andere manier realiseren. Een gated D-latch behoudt z'n huidige stand als $EN = 0$ ($Z_{nieuw} = Z_{oud}$) en is transparant als $EN = 1$ ($Z_{nieuw} = D$). Dit is samen te vatten in de logische functie:

$$Z_{nieuw} = Z_{oud} \cdot \overline{EN} + D \cdot EN \tag{6.6}$$

De realisatie van deze D-latch op basis van formule (6.6) is te zien in figuur 6.14(a). Uit zowel dit schema als de formule blijkt dat de realisatie gebaseerd is op een multiplexer. Zie figuur 6.14. Ook is in deze figuren duidelijk de terugkoppeling te zien.



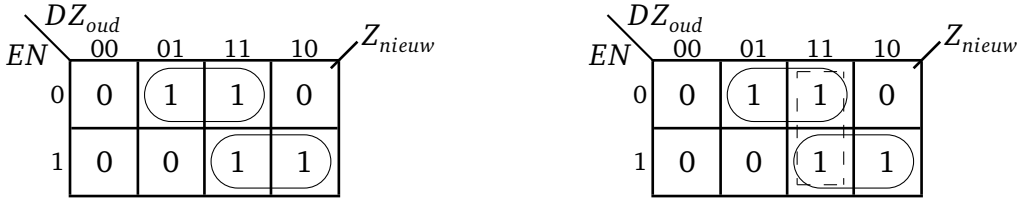
(a) *Schema gated D-latch.*



(b) *Gated D-latch met multiplexer.*

Figuur 6.14: *Gated D-latch op basis van een multiplexer.*

Helaas blijkt deze implementatie in de praktijk niet goed te werken. Nadere inspectie leert dat deze opzet last heeft van een statische-1 hazard (zie paragraaf 4.11) wanneer D en Z_{oud} beide 1 zijn en EN verandert van 1 naar 0. Dit is goed te zien in het Karnaughdiagram in figuur 6.15(a). De hazard is te verhelpen door de productterm $D \cdot Z_{oud}$ toe te voegen die de twee andere producttermen overlapt, zoals te zien is in het Karnaughdiagram in figuur 6.15(b). Op basis van de consensuswet (3.19a) van de schakelalgebra is deze term overbodig, maar voor het praktisch gebruik van de latch noodzakelijk.



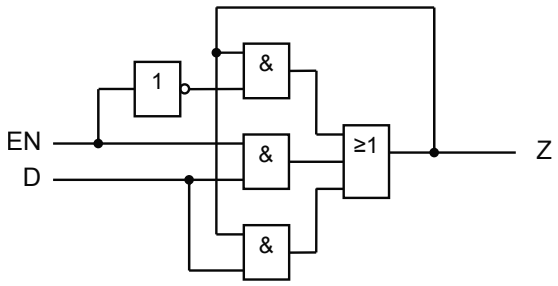
(a) *Oorspronkelijke functie.* **(b)** *Met overlappende term.*

Figuur 6.15: *Karnaughdiagram van de gated D-latch op basis van een multiplexer.*

De functie voor de te realiseren, praktische gated D-latch is:

$$Z_{nieuw} = Z_{oud} \cdot \overline{EN} + D \cdot EN + D \cdot Z_{oud} \tag{6.7}$$

Het bijbehorende schema is te zien in figuur 6.16. Het nadeel van deze realisatie is dat testen lastig is [114].



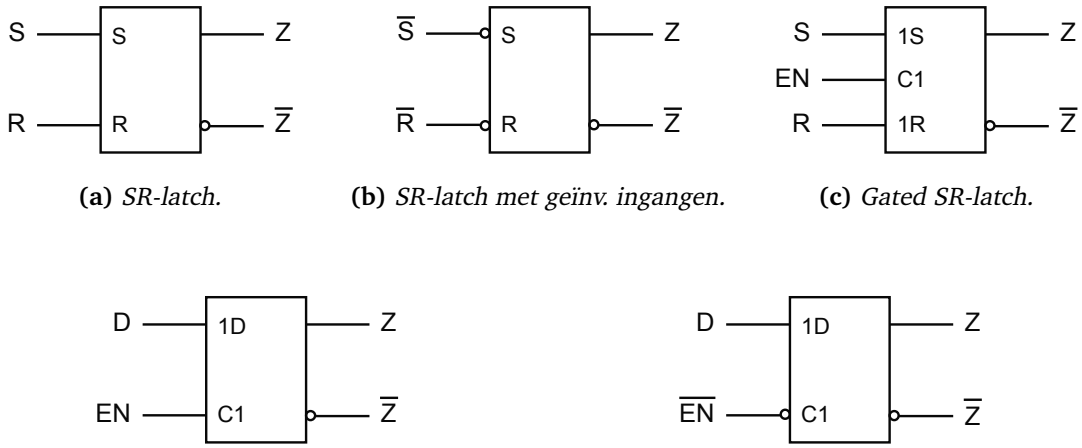
Figuur 6.16: *Praktische uitvoering gated D-latch.*

Hazards beperken zich niet tot de gated D-latch. Alle schakelingen die zijn opgebouwd met *asynchrone logica* hebben last van dit verschijnsel. In de praktijk wordt daarom veel gewerkt met *synchrone logica*. De logica voert acties uit op een gemeenschappelijk stuursignaal, de klok. Zie paragraaf 6.10.

6.5 Symbolen voor latches

Voor het tekenen van latches in schema's worden de symbolen gebruikt zoals ze te zien zijn in figuur 6.17. Deze symbolen zijn gedefinieerd in de normbladen voor IEEE-symbolen [37, 38].

In figuur 6.17(a) is het symbool getekend voor een SR-latch. Merk op dat de signalen S en R zowel binnen als buiten het kader te zien zijn. De definitie van de symbolen heeft in principe alleen betrekking op de signaalnamen binnen het kader. De signaalnamen buiten het kader zijn zogenoemde externe signaallijnen. Een logische 1 op een externe signaal wordt vertaald naar een logische 1 op het bijbehorende interne signaal. Het symbool zegt ook iets over de interne werking van de schakeling. De werking is gelijk aan de definitie zoals is gegeven in tabel 6.2. Uit dit symbool is niet af te leiden wat de werking van de latch is als beide externe signalen logisch 1 zijn. Dan moeten er uitgebreidere symbolen toegepast worden.



(a) *SR-latch.* **(b)** *SR-latch met geïnv. ingangen.* **(c)** *Gated SR-latch.*

(d) *Gated D-latch met actief hoge enable.* **(e)** *Gated D-latch met actief lage enable.*

Figuur 6.17: *IEEE-symbolen diverse latches.*

De negatie-indicatoren in figuur 6.17(b) geven aan dat er een verschil is tussen de interne en externe waarde van de ingangen. Een logische 0 op een extern signaal wordt vertaald naar een logische 1 op een intern signaal en andersom. De interne werking is verder identiek aan het symbool in figuur 6.17(a).

Bij de drie overige symbolen is een enable-ingang te zien. Dit wordt de *command input* genoemd. Deze command input heeft zelf geen directe invloed op de interne stand van de latches, maar geeft aan wanneer de afhankelijke ingangen invloed op de stand van de latch kunnen uitoefenen. Hiervoor moet de interne waarde van de command input logisch 1 zijn. Aan de command input is een nummer gekoppeld, in dit geval nummer 1. Signalen die worden vooraf gegaan door hetzelfde nummer zijn afhankelijk van de command input. Als de interne waarde van de command input logisch 0 is, dan verliezen de afhankelijke ingangen hun invloed.

Figuur 6.17(c) is het symbool van een gated SR-latch getekend. De interne signaalnamen $1S$ en $1R$ beginnen met nummer 1 en zijn afhankelijke van control input $C1$. Als de control input logisch 1 is, heeft de gated SR-latch dezelfde werking als de latch in figuur 6.17(a).

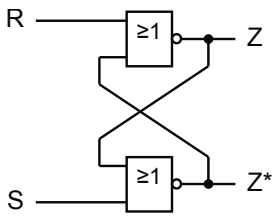
In de figuren 6.17(d) en 6.17(e) zijn de gated D-latches getekend met respectievelijk een actieve hoge enable en een actieve lage enable. Ook hier geldt dat de latches transparant zijn als het externe enablesignaal logisch 1 respectievelijk logisch 0 is.

Verder valt op te merken dat alle latches twee complementaire uitgangen hebben. Dat geldt alleen voor de combinatie voor een onthoud-, set- en reset-opdracht. De combinatie $SR = 11$ wordt uitgesloten. Daarnaast worden bij de uitgangen geen interne signaalnamen beschreven.

6.6 Timing SR-latch

Net als bij poorten heeft ook een SR-latch een minimale en maximale vertragingstijd. We kunnen een globaal tijdmodel van de latch opstellen met de poortvertragingen waarmee de latch is opgebouwd. Bij ontwerpen op transistorniveau kunnen de parameters nog

gedetailleerder worden uitgewerkt. In figuur 6.18 is een SR-latch te zien op basis van NOR-poorten. In deze figuur is naast de uitgang Z ook een uitgang Z^* opgenomen. Merk op dat uitgang Z^* bij $SR = 11$ niet de inverse van Z is.



Figuur 6.18: *SR-latch met NOR-poorten.*

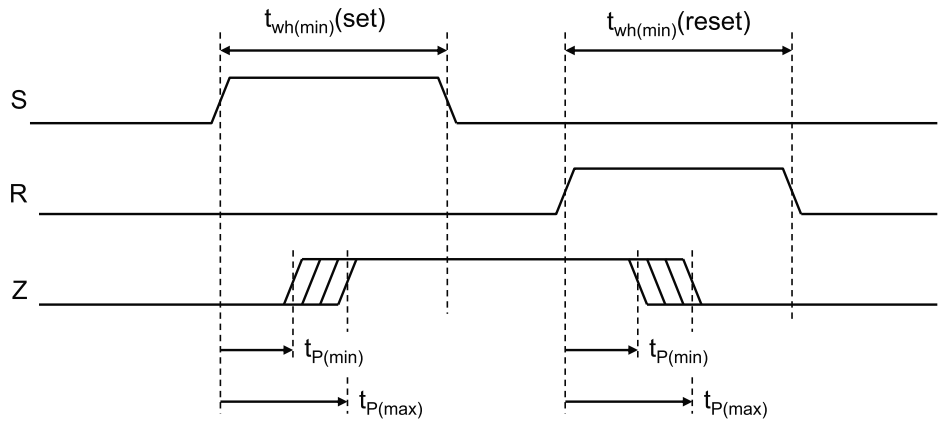
Voor het tijdmodel gaan we uit van uitgang Z. Het kortste pad (in tijd) is uiteraard van R naar Z, namelijk één poortvertraging. Het langste pad is van S naar Z: twee poortvertragingen. In principe is er een verschil in timingparameters van S naar Z en van R naar Z. Dat levert vier vertragingstijden tijden op: $t_{P(min)}$ van R naar Z en van S naar Z en $t_{P(max)}$ van R naar Z en van S naar Z. We zijn echter alleen geïnteresseerd in het worst case tijdmodel zodat we voor de SR-latch kunnen opstellen:

$$\begin{aligned} t_{P(min)}(\text{latch}) &= t_{P(min)}(\text{NOR}) = t_{P(min)}(\text{R-to-Z}) \\ t_{P(max)}(\text{latch}) &= 2 \cdot t_{P(max)}(\text{NOR}) = t_{P(max)}(\text{S-to-Z}) \end{aligned} \tag{6.8}$$

Ook de duur van de set- en resetopdracht is uit te drukken in poorttijden. Vanuit S of R is te zien dat een signaal minstens drie poorttijden logisch 1 moet blijven om de lus in te stellen. Stel dat we de latch willen resetten, dan moet R logisch 1 worden. S blijft logisch 0. Na één poortvertraging is uitgang Z logisch 0 geworden. Dit zorgt voor een logische 0 op de bovenste ingang van de onderste NOR-poort. Samen met de logische 0 van S wordt uitgang Z^* logisch 1. Dit zorgt weer voor een logische 1 op onderste ingang van de bovenste NOR-poort. Na nog een poortvertraging is deze logische 1 "ingewerkt" op de bovenste NOR-poort en kan R weer logisch 0 worden want de onderste ingang neemt nu de werking van R over. We zouden verwachten dat R al na twee poortvertragingen logisch 0 gemaakt kan worden. Dat hangt echter af van de interne opbouw van de latch. Eenzelfde verhaal kan over de duur van de setopdracht worden gehouden. De minimale duur van de set-en resetopdracht is:

$$\begin{aligned} t_{wh(min)}(\text{set}) &= 3 \cdot t_{P(max)}(\text{NOR}) \\ t_{wh(min)}(\text{reset}) &= 3 \cdot t_{P(max)}(\text{NOR}) \end{aligned} \tag{6.9}$$

In figuur 6.19 is een timingdiagram te zien van de SR-latch. De signalen S en R hebben een minimale pulsduur zodat de latch zich correct kan instellen. De uitgang Z verandert tussen $t_{P(min)}(\text{latch})$ en $t_{P(max)}(\text{latch})$. Dit wordt in de figuur aangegeven met een arcering. Wat niet in het timingdiagram is aangegeven, is dat tussen het einde van de set-puls en het begin van de reset-puls een zekere tijd moet zitten. Als deze tijd te kort is, functioneert de latch niet correct.

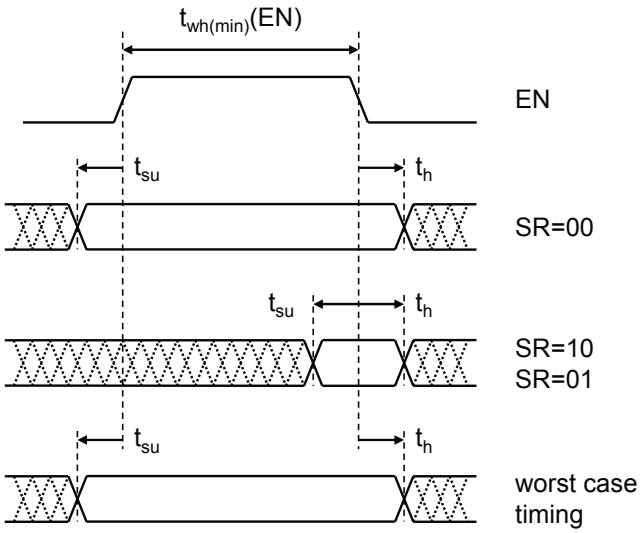


Figuur 6.19: *Timing van de SR-latch.*

*6.7 Timing gated SR-latch

De gated SR-latch heeft naast de ingangssignalen S en R nog het ingangssignaal EN. De timing van een gated SR-latch wordt hierdoor complexer dan die van een SR-latch.

In figuur 6.20 is de timing van de ingangssignalen van de gated SR-latch te zien. Voor een correcte werking van de latch moet het enable-signaal een minimale pulsduur hebben. Dit is in de figuur aangegeven met de timingparameter $t_{wh(min)}$(EN). We gaan ervan uit dat aan deze voorwaarde wordt voldaan.



Figuur 6.20: *Timing van de ingangen van een gated SR-latch.*

We onderscheiden drie situaties: $SR = 00$, $SR = 01$ en $SR = 10$. De situatie $SR = 11$ wordt buiten beschouwing gelaten.

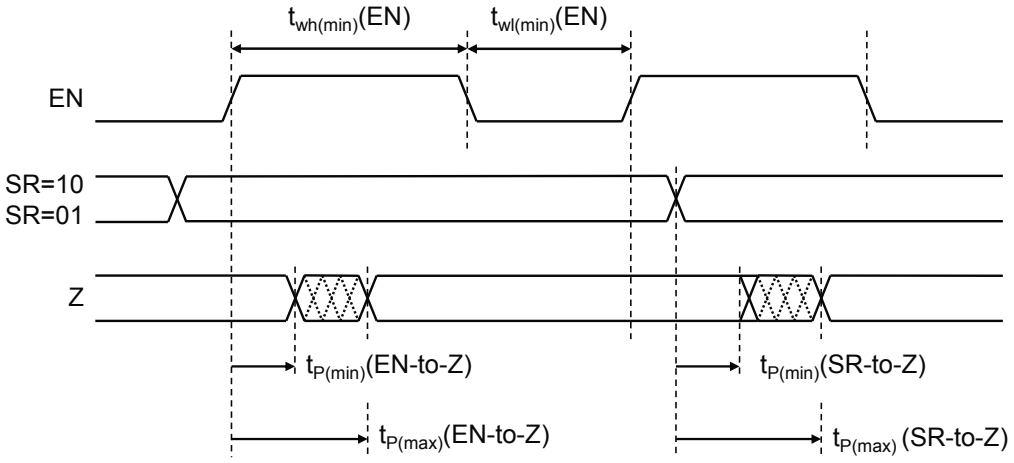
Bij $SR = 00$ onthoudt de latch. De stand van de latch verandert dan niet. Tijdens $EN = 1$ mogen S en/of R dan niet veranderen, anders wordt onbedoeld een set- of reset-opdracht gegeven. Denk hierbij aan hazards van sturende logica. Net voordat de latch transparant wordt, moeten S en R stabiel zijn zodat de latch zich nog netjes kan instellen. Dit wordt

de *setuptijd* van de latch genoemd. Merk op dat de setuptijd t.o.v. de *opgaande flank* van EN (EN gaat van 0 naar 1) is gedefinieerd. Na het sluiten van de latch moeten S en R nog een korte tijd stabiel blijven. Dit wordt de *holdtijd* genoemd. Merk op dat de holdtijd t.o.v. de *neergaande flank* van EN (EN gaat van 1 naar 0) is gedefinieerd. In het gebied setuptijd – holdtijd moeten S en R stabiel blijven.

Bij $SR = 01$ of $SR = 10$ wordt de latch gereset of geset, dat is een bedoelde operatie. Tijdens $EN = 1$ mogen S en R zich instellen. De stand van de latch en de uitgang nemen dan nieuwe waarden aan. Rond het sluiten van de latch moeten S en R stabiel zijn zodat de latch zich netjes kan instellen. Ook hier zijn de setuptijd en holdtijd gedefinieerd. Merk op dat de setuptijd nu t.o.v. de neergaande flank is gedefinieerd.

Tijdens gebruik van de latch moeten we ervan uitgaan dat alle ingangscombinaties van S en R worden aangeboden ($SR = 11$ sluiten we uit). Hierdoor zijn we genoodzaakt de strengste timingseisen aan te houden (*worst case timing*), dat zijn de eisen die horen bij $SR = 00$.

In figuur 6.21 is het timingdiagram getekend van de uitgang van de gated SR-latch. Hier onderscheiden we de volgende situaties. De ingangssignalen S en R zijn ingesteld op $SR = 01$ of $SR = 10$ voordat de latch transparant wordt. De uitgangsreactie is dan als gevolg van de opgaande flank van EN. Dit wordt weergegeven met de minimale en maximale propagatietijden van uitgang Z t.o.v. EN. Als S en R zich instellen tijdens $EN = 1$ kan de stand van de latch, afhankelijk van de huidige stand, veranderen. De uitgangsreactie is dan als gevolg van het veranderen van S of R. Dit wordt weergegeven met de minimale en maximale propagatietijden van uitgang Z t.o.v. S en R. In de figuur is nog de parameter $t_{wh(min)}$(EN) weergegeven. Dit is de tijd dat het enable-signaal minimaal logisch 0 moet zijn.



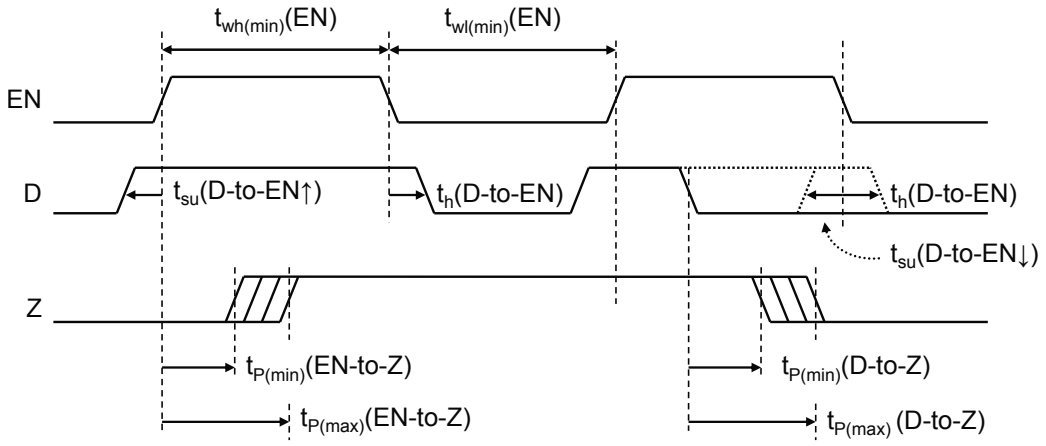
Figuur 6.21: *Timing van de uitgang van een gated SR-latch.*

Bij de situatie $SR = 00$ gaan we ervan uit dat S en R stabiel blijven tijdens $EN = 1$ waardoor de stand van de latch en de uitgang niet verandert. Deze situatie is niet in de figuur weergegeven.

Mede door de complexe timing worden gated SR-latches in de praktijk niet gebruikt.

6.8 Timing gated D-latch

De gated D-latch heeft twee ingangssignalen, D en EN. De timing van een D-latch is complexer dan die van de SR-latch. We onderscheiden twee situaties: D is stabiel tijdens $EN = 1$ en D verandert tijdens $EN = 1$. Tijdens $EN = 0$ mag D veranderen, dit heeft geen invloed op de stand van de latch en de uitgangswaarde. Een timingdiagram is te zien in figuur 6.22.



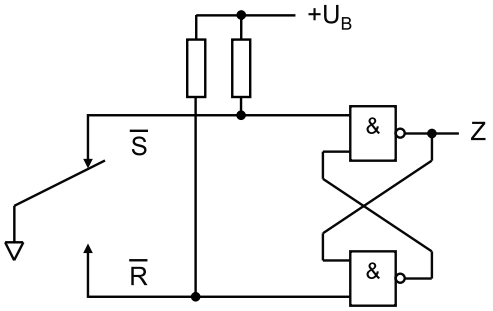
Figuur 6.22: *Timing van een gated D-latch.*

In het eerste geval is D stabiel tijdens $EN = 1$. Voor een betrouwbare werking moet D iets voor de opgaande flank van EN zijn definitieve waarde aangenomen hebben zodat de latch zich kan instellen. Deze insteltijd wordt de *setuptijd* van de latch genoemd, weergegeven met de timingparameter t_{su}(D-to-EN↑). Let op de opgaande pijl in de parameter, het gaat hier om de setuptijd t.o.v. de opgaande flank van EN. D moet stabiel blijven tot na de neergaade flank van EN. Deze tijd wordt de *holdtijd* van de latch genoemd en wordt weergegeven met de timingparameter t_h(D-to-EN). De timing van de uitgang is alleen gerelateerd aan EN. De minimale propagatietijd van uitgang Z, weergegeven met $t_{P(min)}$(EN-to-Z), is de tijd waarop de uitgang op zijn vroegst veranderd, of anders gezegd, de tijd waarin de oude waarde van Z nog beschikbaar is. De maximale propagatietijd van uitgang Z, weergegeven met $t_{P(max)}$(EN-to-Z), is de tijd waarop de uitgang zijn definitieve (nieuwe) waarde heeft. In het gebied tussen $t_{P(min)}$(EN-to-Z) en $t_{P(max)}$(EN-to-Z) is de waarde van de uitgang onzeker. Dit is in de figuur weergegeven met het gearceerde gebied.

In het tweede geval verandert D tijdens $EN = 1$. De latch is nu transparant en zal elke verandering op D direct doorgeven naar de uitgang. De timingparameters van de uitgang worden nu ten opzichte van D gegeven, $t_{P(min)}$(D-to-Z) voor de minimale propagatietijd en $t_{P(max)}$(D-to-Z) voor de maximale propagatietijd. D moet zijn definitieve waarde aannemen rond het sluiten van de latch als EN van 1 naar 0 gaat. Ook hier zijn een setuptijd en een holdtijd gespecificeerd. De setuptijd t_{su}(D-to-EN↓) is nu t.o.v. de neergaande flank gespecificeerd.

6.9 Toepassing SR-latch: ontdenderschakeling

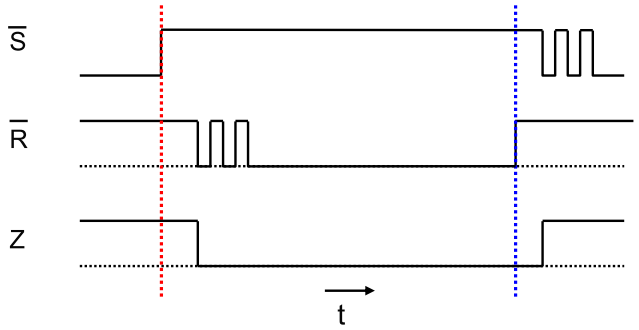
Mechanische schakelaars en druktoetsen worden in veel systemen gebruikt om opdrachten te geven. Denk hierbij aan een startoperatie en een stopoperatie. Aan deze mechanische schakelaars kleeft een nadeel. Tijdens het omschakelen tussen de twee standen (of meer, maar we beschouwen hier alleen twee standen) dendert de contactveer. De achterliggende schakeling ontvangt zo meerdere pulsen die de besturing in de war brengen. Met behulp van de schakeling in figuur 6.23 is dit probleem te verhelpen.



Figuur 6.23: *Ontdenderschakeling met SR-latch.*

De schakeling bestaat uit een tweestandenschakelaar, een SR-latch met NAND-poorten en twee weerstanden. De weerstanden zorgen voor een gedefinieerd signaalniveau als er geen geleidend pad is naar van een ingang van de SR-latch naar de referentie. Tijdens het omschakelen, als de contactveer geen verbinding maakt met de twee uiteinden, zijn de ingangen van de SR-latch beide hoog. Dat is de onthoudopdracht van de latch.

In figuur 6.24 is het signaaldiagram van de gehele schakeling geschetst. De schakelaar staat eerst in de stand $\overline{S}$ zodat dit signaal laag is. De waarde van signaal $\overline{R}$ is hoog. Dit is de set-opdracht van de SR-latch. Op een gegeven moment wisselt de contactveer van positie. De contactveer verbreekt de verbinding met $\overline{S}$ zodat deze ingang van de SR-latch hoog wordt. Het duurt even voordat de contactveer verbinding maakt met $\overline{R}$. Beide ingangen zijn dus hoog en dat komt overeen met de onthoudopdracht van de SR-latch. Bij het verbinden van $\overline{R}$ wordt een resetoperatie aan de SR-latch gegeven. Uitgang Z wordt dan laag. De contactveer stuitert nog enige tijd waarbij afwisselend reset- en onthoudopdrachten worden gegegeven.



Figuur 6.24: *Signaaldiagram van de ontdenderschakeling.*

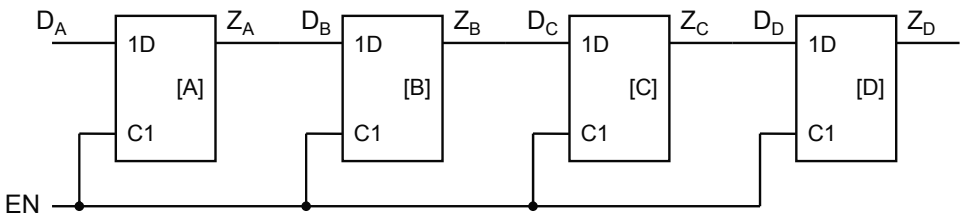
Als de schakelaar wordt omgezet zal de contactveer enkele keren tegen ingang $\overline{S}$ stuiteren. De SR-latch krijgt dan afwisselend een set- en een onthoudopdracht.

Aan een correcte werking zijn enkele voorwaarden verbonden [115]:

- De waarde van de weerstanden hangen af van de gebruikte technologie. Het gaat hierbij om de ingangsstromen bij hoog en laag. Een goede keuze voor CMOS is tussen 1 kΩ en 10 kΩ per weerstand.
- De schakelaar moet van het type verbreek-voor-maak zijn anders worden de beide ingangen van de SR-latch kortstondig laag. De SR-latch wordt dan niet correct gebruikt.
- De schakelaar mag tijdens het denderen niet geheel terugveren, want dan worden er onbedoeld meerdere pulsen op $\overline{S}$ en $\overline{R}$ opgewekt.
- De SR-latch moet snel genoeg zijn om in een nieuwe stabiele toestand te komen anders kan de uitgang gaan zweven. De uitgang wordt dan even metastabiel.

6.10 D-flipflop

De gated D-latch is transparant zolang $EN = 1$, d.w.z. dat een signaalwisseling op ingang D direct en verandering op de uitgang geeft. Op basis van deze latch is het niet makkelijk om data van een latch in een andere over te brengen. Een veel voorkomende digitale schakeling is een *schuifregister*. Hierbij wordt een aantal identieke geheugenelementen zó geschakeld dat de data geschoven kan worden onder besturing van een enablesignaal. Zie figuur 6.25.



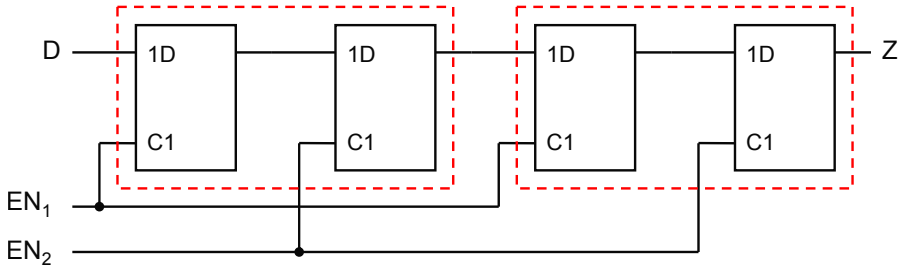
Figuur 6.25: *Poging tot realisatie van een schuifregister op basis van latches.*

Het idee is vrij simpel. Als EN logisch 0 is, onthouden alle latches de laatst ingebrachte waarde. De inhoud van het schuifregister is beschikbaar op de uitgangen Z_A t/m Z_D. Op het moment dat EN logisch 1 is, moeten alle latches de waarde overnemen die op hun dataingangen staan. Latch A neemt de waarde D_A over, de latches B, C en D nemen de waarden over van de latches ervoor. De oorspronkelijke waarde van latch D gaat verloren ("eruit geschoven").

De geschetste situatie gaat echter fout als EN van 0 naar 1 gaat. *Alle* latches zijn dan tegelijk transparant en data op D_A wordt, met enige vertraging, direct naar Z_D doorgevoerd. Als oplossing zou ervoor gekozen kunnen worden om de duur van de enable-puls zodanig aan te passen zodat de latch al weer vergrendeld is net voordat de nieuwe waarde van de voorgaande latch beschikbaar is [116, 117]. Helaas is dat in de praktijk niet goed te realiseren. Ten eerste is het lastig om kortdurende pulsen door verbindingen te zenden,

de pulsen vervormen door capacitieve en inductieve werking. Ten tweede worden de pulsen vertraagd (skew). Daarnaast is het nog maar de vraag of de latch zich wel correct heeft ingesteld als de enablepuls van 1 naar 0 gaat. De timingparameters hangen onder meer af van de belasting aan de uitgang, de voedingsspanning en de bedrijfstemperatuur.

Dit transparant zijn kan worden tegengehouden door twee latches afwisselend te plaatsen, elk met een eigen enable-signaal. Er zijn dan ook twee enable-signalen nodig. In figuur 6.26 is de opbouw te zien van een schuifregister met twee secties (we spreken liever niet van bits, want de schakeling bestaat uit vier geheugenelementen). Elke sectie vertegenwoordigt één bit van het schuifregister.



Figuur 6.26: *Realisatie van een schuifregister op basis van D-latches en twee enablepulsen.*

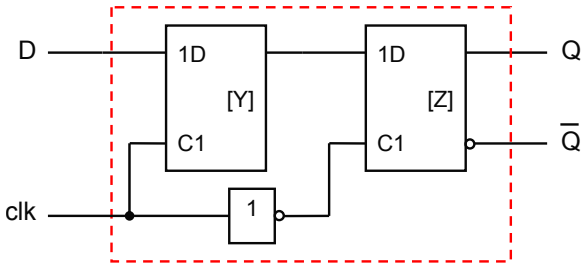
De eerste en derde latch staan onder besturing van $EN1$ en de tweede en vierde latch staan onder besturing van $EN2$. Voorwaarde voor juiste werking is dat $EN1$ en $EN2$ elkaar niet mogen overlappen, d.w.z. dat $EN1$ en $EN2$ niet tegelijk logisch 1 mogen zijn. We sluiten deze combinatie dus uit. Alle latches zijn vergrendeld als $EN1 = EN2 = 0$. Als $EN1 = 1$ en $EN2 = 0$ zal de eerste latch data overnemen van ingang D en de derde latch data overnemen van de tweede latch. De eerste en derde latch zijn transparant. De inhoud van de tweede en vierde latch blijven ongewijzigd. Als $EN1 = 0$ en $EN2 = 1$ zal de tweede latch data overnemen van de eerste latch en de vierde latch data overnemen van de derde latch. De inhoud van de eerste en derde latch blijven ongewijzigd. Voor het maken van een 4-bits schuifregister zijn dus acht latches nodig.

Een geheugenelement op basis van twee latches die niet tegelijk transparant zijn, wordt een *master-slave flipflop*[2] (meester-slaaf) genoemd, in dit geval een master-slave D-flipflop.

De schakeling in figuur 6.27 is een meester-slaaf flipflop waarbij de $EN2$-puls is afgeleid van de $EN1$-puls door middel van een NOT-poort zodat $EN2 = \overline{EN1}$. Dat is mogelijk omdat de combinatie $EN1 = EN2 = 0$ niet gebruikt hoeft te worden en de combinatie $EN1 = EN2 = 1$ uitgesloten is.

Over de signaalnamen in de figuur zijn twee opmerkingen te maken. Ten eerste is er slechts één enable-signaal. Gewoonlijk is dit een centraal opgewekt signaal met een *duty cycle* (pulsduur/pulsbreedte-verhouding) van 50% en is periodiek. Dit wordt de *klok* van de schakeling genoemd. Hiervoor gebruiken we de naam clk. Ten tweede worden de uitgangen Q en $\overline{Q}$ genoemd. Deze namen reserveren we voor flipflop-uitgangen.

[2] maar welke is nou de master en welke de slave?



Figuur 6.27: *Realisatie van een negative edge triggered master-slave D-flipflop.*

De werking van de master-slave D-flipflop is als volgt. Tijdens $clk = 0$ is de Y-latch gesloten. Veranderingen op de D-ingang worden door de Y-latch niet overgenomen en de laatst ingebrachte stand is op de uitgang van de Y-latch beschikbaar. De Z-latch is transparant en neemt de waarde van de uitgang van de Y-latch over. Rond de opgaande flank van clk, dat is wanneer clk van 0 naar 1 gaat, wordt de Y-latch transparant en zal de waarde op de D-ingang doorgeven op de uitgang. De Z-latch wordt gesloten en onthoudt de huidige waarde. Wanneer $clk = 1$ is volgt de Y-latch de waarde op de D-ingang, de Y-latch is immers transparant. De Z-latch is gesloten en zal de laatst ingebrachte stand beschikbaar stellen op de uitgang. Rond de neergaande flank van clk, dat is wanneer clk van 1 naar 0 gaat, wordt de Y-latch gesloten. De D-ingang moet rond die tijd z'n definitieve waarde hebben. De Z-latch wordt transparant en neemt de huidige waarde van de Y-latch over, dat is de waarde rond de neergaande flank van de klok. Tegelijk verandert ook de uitgang Q van de D-flipflop.

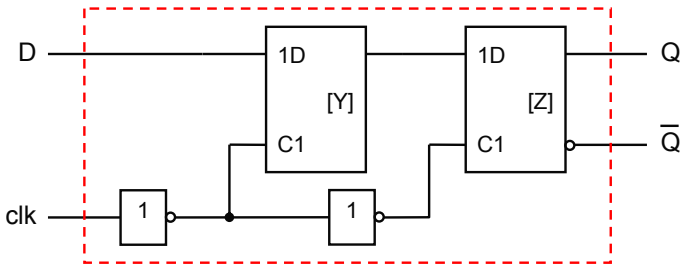
Een voorwaarde waaronder dit is toegestaan is:

$$t_{P(max)}(\text{NOT}) + t_h(\text{Z-latch}) < t_{P(min)}(\text{Y-latch}) \tag{6.10}$$

Dat volgt uit het feit dat voor correcte werking van de flipflop de Z-latch weer vergrendeld moet zijn voordat de uitgang van de Y-latch gaat veranderen.

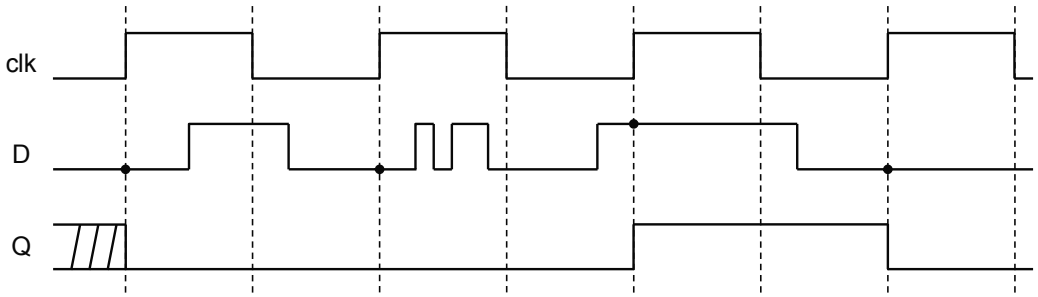
Veranderingen van de stand van de D-flipflop zijn alleen zichtbaar rond *neergaande flank* van het kloksignaal. De D-flipflop is een *flankgevoelig* (edge-triggered) geheugenelement. Het overnemen van data door de flipflop wordt *inklokken* genoemd.

In figuur 6.28 is een D-flipflop te zien die op de opgaande flank data inklokt. Dit wordt gerealiseerd door een NOT-poort in de kloklijn op te nemen vóór het verbindingspunt naar de Y-latch.



Figuur 6.28: *Realisatie van een positive edge triggered master-slave D-flipflop.*

In figuur 6.29 is een signaaldiagram te zien van een positive edge triggered D-flipflop. Het signaaldiagram geeft de logische werking van de flipflop weer, het geeft geen inzicht in de timing. In de figuur is met stippen aangegeven welke waarde de flipflop inklokt op de opgaande flank.

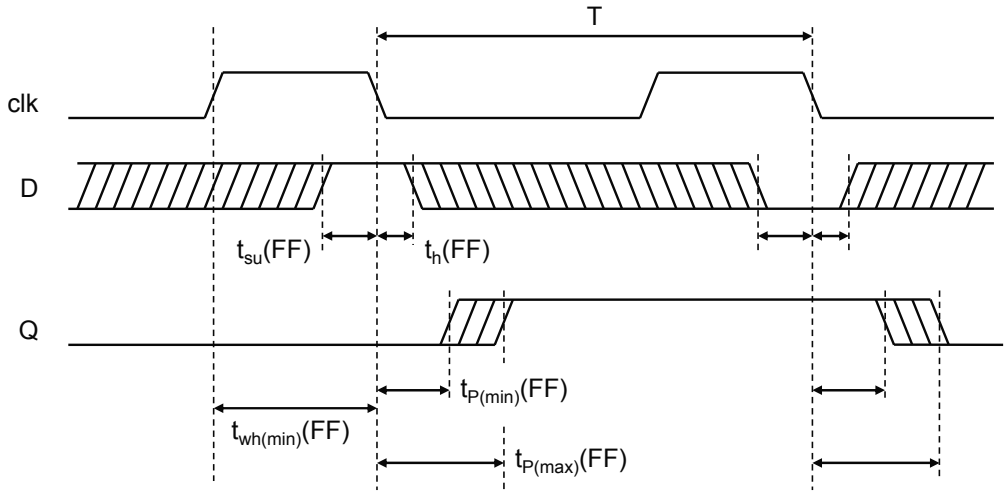


Figuur 6.29: *Signaaldiagram van een positive edge triggered D-flipflop.*

6.11 Timing D-flipflop

Het mag duidelijk zijn dat rond de neergaande flank van klok in figuur 6.27 de waarde van de *D*-ingang stabiel moet zijn. Dat is een voorwaarde voor het betrouwbaar overnemen van de data. In feite wordt de Y-latch dan gesloten.

In figuur 6.30 is het timingdiagram van de negative edge-triggered D-flipflop gegeven. Bij een edge-triggered timing refereren alle timingparameters aan dezelfde flank van het kloksignaal. Dit wordt de *actieve flank* genoemd.



Figuur 6.30: *Timingsdiagram negative edge-triggered D-flipflop.*

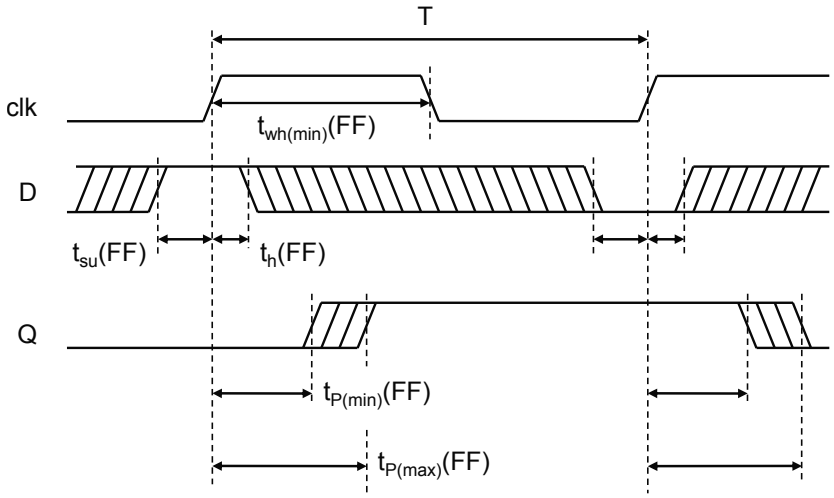
Enige tijd voor de neergaande flank moet de data op de *D*-ingang een definitieve waarde aannemen. Hierdoor kan de Y-latch zich netjes instellen op de laatst aangebrachte waarde. Dit wordt de setuptijd van de flipflop genoemd. De parameter wordt in formules met t_{su}(FF) aangegeven. De data op de *D*-ingang moet ook na de neergaande flank nog even stabiel blijven. Dit wordt wordt de holdtijd genoemd en wordt in formules met t_h(FF) aangegeven. In het gebied t_{su}(FF) $-$ t_h(FF) moet de data op de *D*-ingang stabiel zijn.

Buiten dat gebied mag de data op de D-ingang veranderen. Dit is aangegeven met de gearceerde gebieden in de figuur. Het tijdsinterval waarin de data klaar moet staan is in de regel zeer klein, bij de huidige generatie flipflops minder dan een nanoseconde.

Als we ervan uitgaan dat de data op de D-ingang stabiel is in het setup- en holdtijdgebied, dan verandert de Q-uitgang van de flipflop niet vroeger dan na de minimale propagatietijd $t_{P(min)}$(FF) en niet later dan de maximale propagatietijd $t_{P(max)}$(FF). In dit gebied is de waarde van de uitgang nog niet definitief. Ook dit is aangegeven met de gearceerde gebieden. Merk op dat direct na de neergaande flank de oude (huidige) waarde nog enige tijd beschikbaar is.

Verder moet de tijdsduur $clk = 1$ voldoende lang zijn zodat de Y-latch de data op de D-ingang overneemt. In de figuur is deze parameter te zien als $t_{wh(min)}$(FF). Eenzelfde parameter geldt ook voor $clk = 0$; dit is niet in de figuur aangegeven. Aan de vorm van het kloksignaal worden ook eisen gesteld. De opgaande en neergaande flanken moeten een zekere steilheid hebben, respectievelijk de *rise time* en *fall time* genoemd. De flanken moeten niet te “slap” zijn.

De schakeling in figuur 6.28 klokt de data op de opgaande flank in. Alle timingseisen zijn nu gerelateerd aan de opgaande flank. Zie figuur 6.31.



Figuur 6.31: *Timingsdiagram positive edge-triggered D-flipflop.*

6.12 Logische functie D-flipflop

Edge-triggered flipflops reageren op flanken. Een verandering van de D-ingang is dus niet direct zichtbaar, er moet gewacht worden tot de actieve flank passeert. Om aan te geven dat een huidige waarde van D pas ná de klokflank de stand van de flipflop verandert (die dan zichtbaar wordt op uitgang Q), wordt gebruik gemaakt van de volgende notatie:

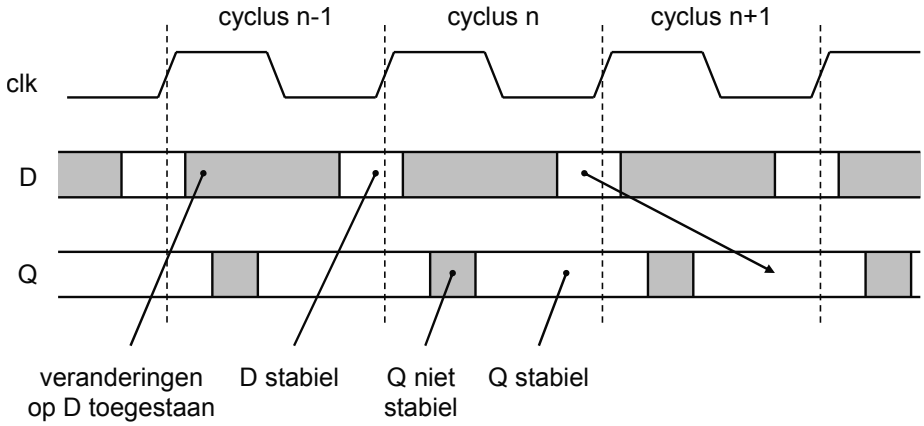
$$Q^{n+1} = D^n \tag{6.11}$$

In de functie van de D-flipflop wordt het kloksignaal niet meer expliciet meegenomen. Het kloksignaal is een hulpsignaal en heeft geen logische betekenis.

Merk op dat de superscripten niets te maken hebben met machtsverheffen. Het is een *notatie* om aan te geven dat de nieuwe waarde van de flipflop gelijk is aan de huidige waarde van de D-ingang ná de klokflank van de huidige cyclus.

In figuur 6.32 zijn de signaalvormen van de klok, de D-ingang en de Q-uitgang rond klokpuls n getekend. De grijze gebieden in de signaalvorm van D geven aan dat D niet stabiel hoeft te zijn. Alleen rond de setup- en holdtijd is dat noodzakelijk. In de signaalvorm van Q geven de grijze gebieden aan dat Q nog niet een definitieve waarde heeft, in de witte gebieden is de waarde op Q wel definitief.

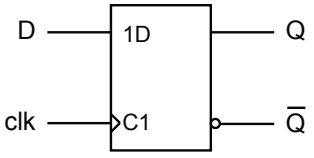
In de figuur is duidelijk te zien dat de waarde van D tijdens klokcyclus n pas beschikbaar is in klokcyclus $n + 1$. Ook is hier goed te zien dat de huidige uitgangswaarde na de klokflank nog even beschikbaar blijft.



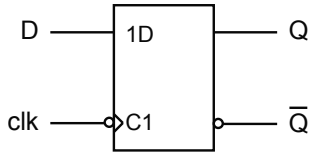
Figuur 6.32: *Signaalnamen edge-triggered D-flipflop.*

6.13 Symbolen voor D-flipflops

In de figuren 6.33(a) en 6.33(b) zijn de symbolen voor een positive edge-triggered en een negative edge-triggered D-flipflop te zien. Ook hier is sprake van een control input C en een van de control input afhankelijke data-ingang D. Voor de control input $C1$ staat een driehoekje (dynamic input indictor) waarmee aangegeven wordt dat $C1$ slechts actief is op een interne 0→1 overgang. Om aan te geven dat een flipflop op de negatieve flank triggert, moet een negatie-indicator worden toegevoegd.



(a) *Positive edge-triggered D-flipflop.*



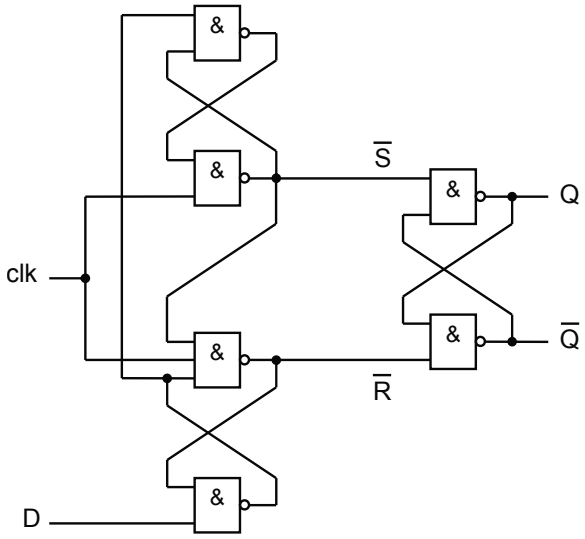
(b) *Negative edge-triggered D-flipflop.*

Figuur 6.33: *IEC/IEEE-symbolen D-flipflop.*

Merk op dat bij de uitgangen geen interne signaalnamen zijn geplaatst.

*6.14 Alternatieve opbouw positive edge-triggered D-flipflop

In de praktijk komen we ook een D-flipflop tegen zoals is weergegeven in figuur 6.34.

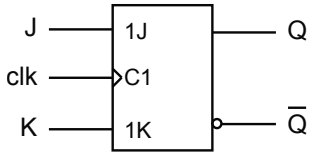


Figuur 6.34: *Alternatieve opbouw positive edge-triggered D-flipflop.*

In deze figuur zijn drie SR-latches te herkennen. Uit commercieel oogpunt is deze flipflop goedkoper te produceren dan een flipflop die op het master-slave-principe is gebaseerd. We zullen de werking niet bespreken. Meer informatie over de werking van deze flipflop is te vinden in [118, 119].

*6.15 JK-flipflop

In de praktijk komt nog een andere flipflop voor, de JK-flipflop. Deze flipflop heeft twee stuuringangen J en K. In tabel 6.3 is de functietabel te zien. Bij $JK = 00$ onthoudt de flipflop de laatst ingebrachte stand. Bij $JK = 01$ wordt de flipflop gereset en bij $JK = 10$ wordt de flipflop geset. Deze besturing heeft veel weg van een SR-latch. De combinatie $JK = 11$ wordt gebruikt om de uitgang eenmaal van stand te veranderen, de geïnverteerde waarde van de uitgang wordt ingeklokt. Zo zijn er eenvoudig *tweedelers* te maken. In figuur 6.35 is het symbool te zien. De beide stuuringangen zijn afhankelijk van de command input $C1$.



Figuur 6.35: *IEC/IEEE-symbool JK-flopflop.*

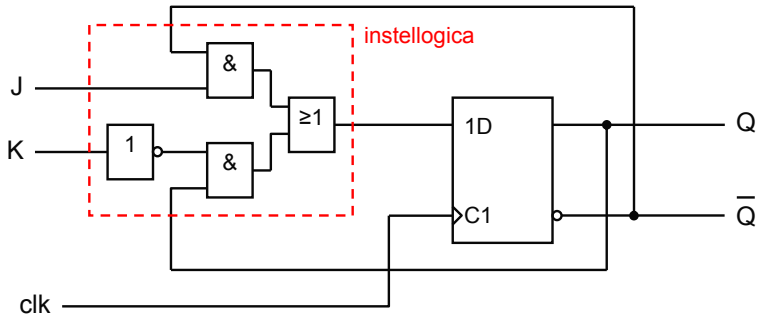
Tabel 6.3: *Functietabel JK-flipflop.*

J	K	Q^{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Oude ontwerpen gaan uit van master-slave SR-flipflop, zie bijvoorbeeld [120]. Deze hebben echter slechte timingeigenschappen. JK-flipflops zijn te maken op basis van

D-flipflops met instellogica. Deze instellogica zorgt ervoor dat de juiste waarde op de *D*-ingang van de D-flipflop wordt aangeboden. Zie figuur 6.36. De logische functie is:

$$Q^{n+1} = J^n \cdot \overline{Q_n} + \overline{K^n} \cdot Q^n \tag{6.12}$$



Figuur 6.36: *Opbouw van een JK-flipflop op basis van een D-flipflop.*

6.16 Asynchrone set en reset

Bij het opkomen van de voedingsspanning is het onduidelijk wat de stand van een flipflop wordt. Vandaar dat flipflops worden voorzien van twee *asynchrone* ingangen. Asynchroon wil zeggen dat de ingangen de stand van de flipflop onafhankelijk van de klok instellen. De set-ingang zorgt ervoor dat stand van de flipflop logisch 1 wordt. De reset-ingang zorgt ervoor dat de stand van de flipflop logisch 0 wordt.

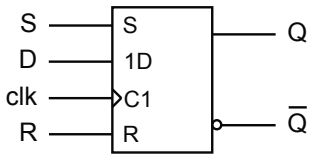
Deze asynchrone set- en reset-ingangen zijn storingsgevoelig en moeten goed afgeschermd worden. Deze ingangen moeten alleen gebruikt worden voor *power-on reset* (en set) en mogen *nooit* gestuurd worden vanuit combinatorische logica. Combinatorische schakelingen kunnen last hebben van hazards en daardoor onbedoeld een van de twee ingangen activeren. Voor set en reset gelden aanvullende timing-eisen zoals pulsduur op de ingangen en de *recovery time*[3] en *removal time*[4]. Dit is de tijd dat de twee ingangen inactief moeten zijn voordat de actieve klokflank mag passeren.

In de figuren 6.37(a) en 6.37(b) zijn de symbolen van de D-flipflop getekend. De interne signalen *S* en *R* zijn niet voorzien van een nummer dat aangeeft dat ze niet afhankelijk zijn van de control input.

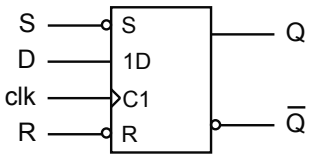
In Engelstalige literatuur worden doorgaans andere symbolen en signaalnamen gebruikt. In figuur 6.38(a) is het symbool te zien van een D-flipflop met actief lage *preset*- en *clear*-ingangen. De preset-ingang zorgt ervoor dat de stand van de flipflop logisch 1 wordt en de clear-ingang zorgt ervoor dat de stand van de flipflop logisch 0 wordt.

[3] recovery time: de tijd tussen het moment dat het asynchrone signaal inactief wordt en de actieve klokflank, werkt als een setuptijd voor asynchrone signalen.

[4] removal time: de tijd tussen de actieve klokflank en het moment dat het asynchrone signaal inactief wordt, werkt als een hold-tijd voor asynchrone signalen.



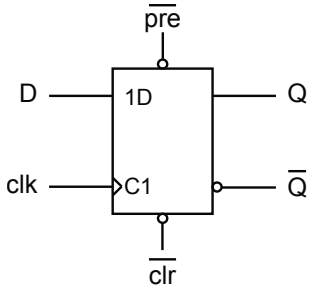
(a) *Positive edge-triggered D-flipflop met asynchrone actief hoge set en reset.*



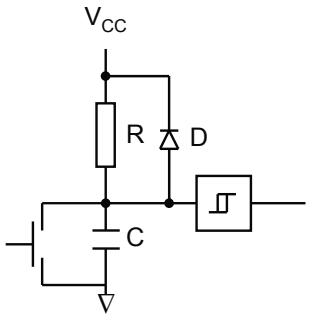
(b) *Positive edge-triggered D-flipflop met asynchrone actief lage set en reset.*

Figuur 6.37: *IEEE-symbolen D-flipflop met set en reset.*

In figuur 6.38(b) is een schema van een eenvoudige power-on reset-schakeling te zien. Er is ook een handbediening aanwezig. De schakeling bestaat uit een weerstand R, een condensator C, een diode D, een Schmitt-trigger en een drukschakelaar. De ingangsstroom van de Schmitt-trigger is verwaarloosbaar. De Schmitt-trigger wordt gevoed met dezelfde voeding.



(a) *D-flipflop met asynchrone preset en clear.*



(b) *Power-on reset-schakeling.*

Figuur 6.38: *Power-on reset en symbool.*

In het begin is de spanning over de condensator 0 V, de uitgangsspanning van de Schmitt-trigger is ook 0 V. Als de voedingsspanning opkomt zal de condensator langzaam opgeladen worden via de weerstand. De diode staat in sper. Na enige tijd is de spanning over de condensator groter dan de omslagspanning van de Schmitt-trigger en zal de uitgang logisch 1 worden. De omslagspanning moet dan wel rond de minimale werkspanning van de logische schakeling liggen. Merk op dat de diode nog steeds in sper staat.

Met behulp van de drukschakelaar kan de condensator snel ontladen worden. Indrukken zorgt voor een galvanisch pad naar de referentie. Na loslaten van de schakelaar zal de condensator opnieuw opladen. De condensator heeft nog een tweede functie. Het zorgt ervoor dat de drukschakelaar ontdenderd wordt [121].

Als de voeding uitgezet wordt, zal de spanning over de hele schakeling afnemen. De condensator is echter nog geladen en de spanning over de condensator kan dan hoger worden dan de (afnemende) voedingsspanning. Dat levert problemen aan de ingang van de Schmitt-trigger omdat de ingangsspanning niet (veel) hoger mag zijn dan de voedingsspanning. De diode komt dan in geleiding en zal de condensator snel ontladen.

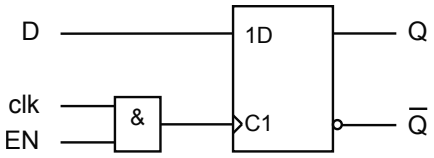
Er zijn wat bedenkingen bij deze schakeling. Ten eerste moet de Schmitt-trigger correct werken, ook als de voedingsspanning opkomt. Ten tweede is het vaak zo dat de achterliggende schakeling nog enige tijd gereset moet worden nadat de voeding de juiste

bedrijfsspanning heeft aangenomen. In de praktijk wordt daarom wel gebruik gemaakt van speciale Power-on Reset ic's. Zie bijvoorbeeld [122, 123].

6.17 Flipflops met enable

Soms moet de stand van een flipflop behouden blijven "over de klokflanken heen". De stand van de flipflop verandert niet ondanks dat er een actieve klokflank passeert. Denk hierbij aan interne geheugens van een microprocessor die niet op elke klokflank een nieuwe waarde moeten krijgen. Hiertoe breiden we de flipflop uit met een enable-ingang EN die ervoor zorgt dat bij $EN = 0$ de flipflop de huidige stand vasthoudt. Bij $EN = 1$ gedraagt de schakeling zich als een gewone D-flipflop. Let erop dat signaal EN niets te maken heeft met het moment van inklokken van data zoals dat bij de latch het geval is.

Een D-flipflop met enable-faciliteit kan op twee manieren gerealiseerd worden. De eerste manier is om de klok uit te schakelen zodat geen klokflank passeert. Dit kan eenvoudig door in de kloklijn een AND-poort op te nemen met als ingangssignalen clk en EN. Zie figuur 6.39. Als $EN = 0$ dan is de uitgang van de AND-poort ook logisch 0. Deze techniek wordt *clock gating* genoemd.



Figuur 6.39: *D-flipflop met enable-faciliteit via het kloksignaal (slecht ontwerp).*

Het nadeel van deze oplossing is dat het enable-signaal niet mag veranderen tijdens $clk = 1$, anders worden onbedoeld kloksignaalovergangen opgewekt. Het resultaat is een flipflop met *pulse-triggered* eigenschappen. Daarnaast zorgt de AND-poort voor vertraging van het kloksignaal. *Dit is absoluut verboden en een slecht ontwerp.*

De tweede manier is om ervoor te zorgen dat de flipflop zijn eigen stand opnieuw inklokt. Dat kan gerealiseerd worden met behulp van een multiplexer. Het schema is te zien in

GATING THE CLOCK

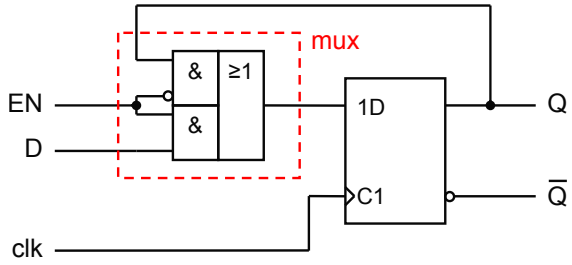
In de praktijk wordt clock gating wel toegepast. Dit heeft alles te maken met het verminderen van dissipatie. De meeste ic's worden gemaakt in CMOS en verbruiken energie als er signaalwisselingen zijn, denk hierbij aan het kloksignaal dat naar flipflops wordt gedistribueerd. Het afschakelen van de klok kan een flinke besparing geven op het energiegebruik. Alle moderne processoren zijn met een vorm van clock gating uitgerust, meestal in combinatie met het aanpassen van de klokfrequentie (*dynamic frequency scaling*) omdat energieverbruik recht evenredig is met de frequentie. Deze technieken mogen alleen door ervaren ontwerpers worden toegepast. Er is veel geschreven over clock gating. Zie voor meer informatie onder andere [124, 125, 126, 127]. Thijssen beschrijft een schakeling voor het afschakelen van de klok in [128].

figuur 6.40. Als $EN = 0$ dan klokt de flipflop zijn eigen waarde opnieuw in. Bij $EN = 1$ klokt de flipflop de nieuwe externe waarde in. Er is wel meer logica nodig dan in het vorige ontwerp, maar het resultaat is een edge-triggered flipflop.

De gerealiseerde functie is:

$$Q^{n+1} = Q^n \cdot \overline{EN^n} + D^n \cdot EN^n \tag{6.13}$$

Deze functie lijkt sterk op functie (6.6) van de D-latch. Er is nu geen overlappende term nodig. De multiplexer kan weliswaar hazards vertonen, maar dat is geen probleem zolang er geen actieve klokflank passeert, m.a.w. we stellen de klokflank uit totdat de multiplexer een definitieve waarde op de D-ingang van de flipflop gerealiseerd heeft. Ook hier heeft signaal EN niets te maken heeft met het moment van inklokken van data zoals dat bij de latch het geval is. EN is een synchroon stuursignaal dat bepaalt welke waarde de flipflop inklokt. Voor EN en D gelden uiteraard setup- en holdtijden ten opzichte van de actieve klokflank. Verder is er geen vertraging van het kloksignaal.



Figuur 6.40: *D-flipflop met enable-faciliteit via het datasignaal.*

6.18 Ontwerpen en gebruik van flipflops

Moderne D-flipflops worden tegenwoordig allemaal ontworpen volgens het master-slave principe. Dat kan redelijk efficiënt in de CMOS-technologie. Een D-flipflop is met zo'n 16 transistoren te realiseren [129].

Het ontwerpen van een betrouwbare flipflop is een lastige aangelegenheid en moet gedaan worden door ervaren ontwerpers. Het zelf ontwerpen van een flipflop met losse poorten of in een beschrijvingstaal als VHDL is niet aan te raden. VHDL heeft taalconstructies voor het beschrijven van klokflanken (en dus van flipflops). Zie listing 6.1.

Bij gebruik van reconfigureerbare logica (FPGA, CPLD) is het helemaal niet gewenst, de fabrikant heeft namelijk al kant-en-klare flipflops aangebracht. Door gebruik van *library modules* kan een flipflop gerealiseerd worden. Zie figuur 6.41.

Voor de logische werking van een schakeling maakt het niet uit op welke flank een flipflop z'n data inklokt. Het door elkaar gebruiken van positive en negative edge-triggered flipflops is meestal niet de bedoeling.

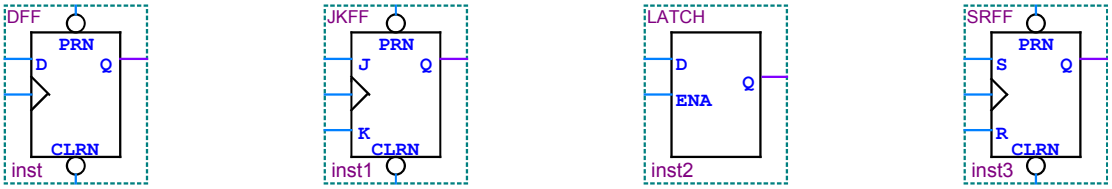
Het gebruik van flipflops met verschillende timing-parameters levert problemen op. Binnen één ic is dat echter niet het geval.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity dff is
    port (clk : in std_logic;
          d   : in std_logic;
          q   : out std_logic
         );
end entity dff;

architecture behavioral of dff is
begin
    -- process only sensitive to clk
    process (clk) is
    begin
        -- clk has changed and clk = 1 then pos. edge
        if clk'event and clk = '1' then
            q <= d;
        end if;         -- no else!
    end process;
end architecture behavioral;
```

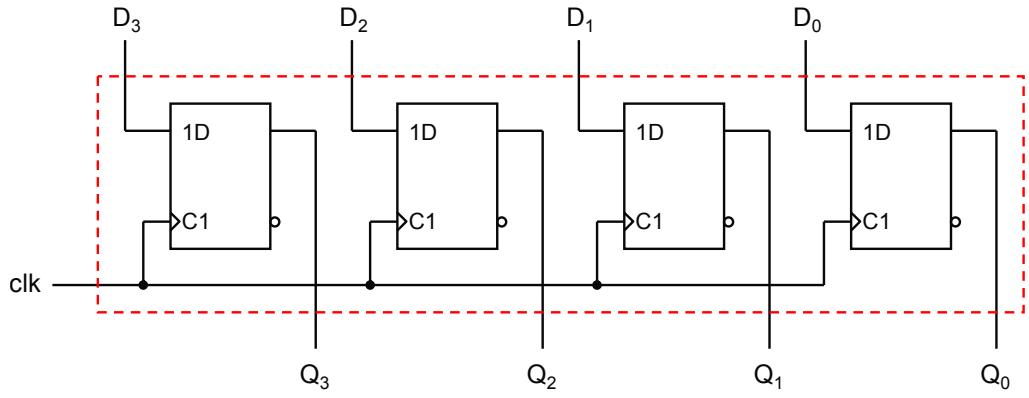
Listing 6.1: *VHDL-beschrijving van een positive edge-triggered D-flipflop.*



Figuur 6.41: *Een aantal geheugenelementen als library modules (Altera).*

6.19 Registers en schuifregisters

We kunnen de D-flipflops combineren tot grotere schakelingen. We bespreken er twee. Een *register* is een groep (D-)flipflops die alle getriggerd worden op hetzelfde kloksignaal. Zie als voorbeeld figuur 6.42. Er zijn verschillende uitvoeringen van registers. Sommige



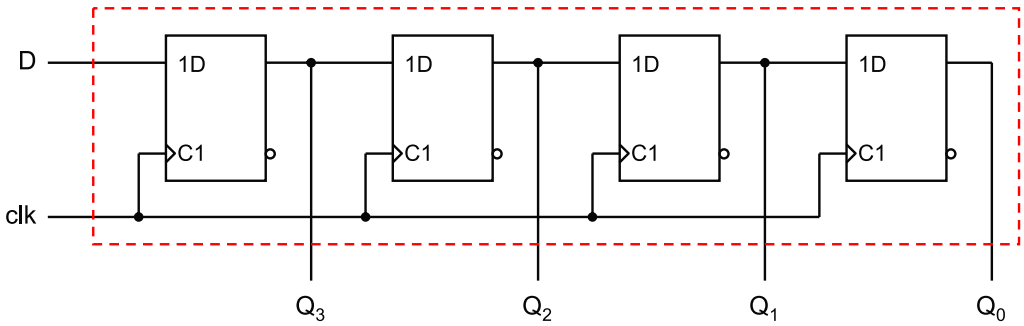
Figuur 6.42: *Schema van een 4-bits register.*

hebben een enable-faciliteit, zodat data selectief ingeklokt kan worden. Andere uitvoeringen hebben een *output enable*, zodat de uitgangen in een hoge-impedante toestand gebracht kunnen worden. De term register wordt veel gebruikt in het low-level programmeren. Zo kan de instructie gegeven worden om een 32-bits register, bijvoorbeeld in een microprocessor, te laden met een getal.

In de digitale techniek en met name in de digitale communicatie wordt veel gebruik gemaakt van seriële transmissie. Het voordeel ten opzichte van parallelle transmissie is overduidelijk: er is slechts een beperkt aantal verbindingen nodig om informatie (bits) te verzenden of te ontvangen. Het nadeel is dat het versturen van een byte of word veel langer duurt dan bij parallelle transmissie. Voorbeelden van seriële transmissiesystemen zijn USB en Ethernet.

Bij het gebruik van seriële transmissie worden *schuifregisters* gebruikt. Een schuifregister is een groep (D-)flipflops waarvan de data-uitgang van een flipflop is verbonden met de data-ingang van de naastgelegen flipflop.

In figuur 6.43 is een een 4-bits schuifregister te zien. De data op ingang *D* wordt op elke opgaande flank ingeklokt in de linker flipflop. De overige flipflops nemen hun data over van de uitgang van links gelegen flipflip. Er zijn maar vier flipflops beschikbaar dus na verloop van tijd verdwijnen bits aan de rechterkant. We noemen dat “eruit geschoven”.



Figuur 6.43: *Schema van een 4-bits schuifregister.*

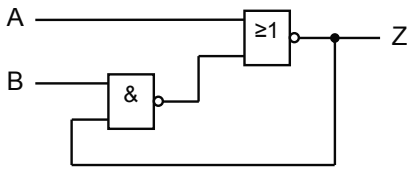
Schuifregisters worden in een later hoofdstuk besproken.

6.20 Opgaven

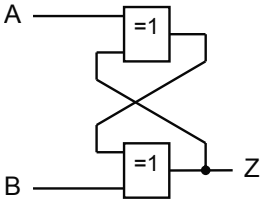
6.1. De SR-latches met overheersende set en reset zijn reeds behandeld. De don’t cares waren dan ingevuld als respectievelijk twee enen en twee nullen. Het zijn twee don’t cares, dus er zijn vier combinaties mogelijk. Ontwerp SR-latches voor de twee overgebleven combinaties. Zijn de ontwerpen zinvol?

6.2. Gegeven is de schakeling in figuur P6.1. Realiseert deze schakeling een goedwerkend geheugenelement? Motiveer het antwoord.

6.3. Gegeven is de schakeling in figuur P6.2. Realiseert deze schakeling een goedwerkend geheugenelement? Motiveer het antwoord.



Figuur P6.1: *Een poging om een latch te realiseren.*



Figuur P6.2: *Een poging om een latch te realiseren.*

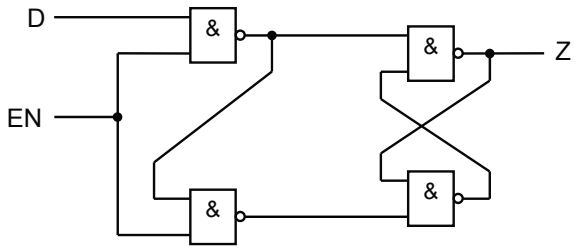
6.4. Is het mogelijk om een SR-latch te maken met de onderstaande SR-combinaties? Motiveer het antwoord.

$SR = 11$ onthouden
$SR = 10$ niet gebruikt
$SR = 01$ reset
$SR = 00$ set

6.5. Eerder is het signaaldiagram van de SR-latch met overheersende set besproken. Vul hetzelfde signaaldiagram in voor een SR-latch met overheersende reset.

6.6. In de symbolen van de SR-latches komt de uitgang $\overline{Z}$ voor. Dat suggereert dat dit de inverse is van Z. Geldt dat voor elke combinatie van S en R?

6.7. De schakeling in figuur P6.3 lijkt op een gated D-latch. Is dit een correct werkende gated D-latch? Motiveer het antwoord.



Figuur P6.3: *Een gated D-latch.*

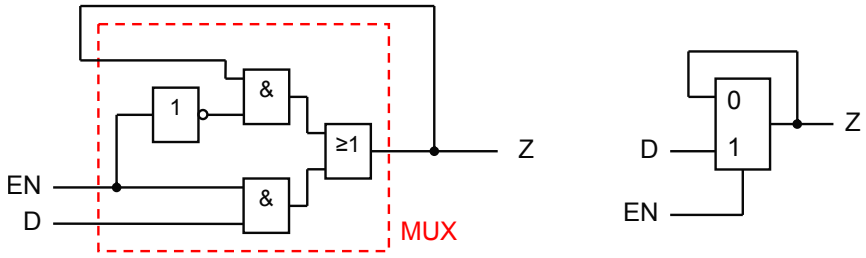
6.8. Een *tweedeler* is een schakeling die op commando van een enable-puls geacht wordt eenmaal van uitgangswaarde te veranderen. Als de uitgang 1 is wordt deze 0 en omgekeerd. Is een tweedeler te maken met één latch en poorten? En met twee latches en poorten? Motiveer het antwoord.

6.9. Voor een master-slave flipflop moet gelden dat:

$$t_{P(max)}(\text{NOT}) + t_h(\text{Z-latch}) < t_{P(min)}(\text{Y-latch})$$

Toon deze voorwaarde aan met een timingdiagram en verklaar dit.

6.10. Een gated D-latch voldoet aan de functie $Z_{nieuw} = \overline{EN} \cdot Z_{oud} + EN \cdot D$ en kan worden gebouwd met behulp van een *multiplexer*, zie figuur P6.4.



Figuur P6.4: *D-latch op basis van een multiplexer.*

Toon aan dat dataoverdracht *niet* betrouwbaar verloopt. Hint: bekijk de situatie $Z = 1$, $D = 1$ en EN gaat van 1→0.

6.11. Probeer een double edge-triggered D-flipflop te ontwerpen. Dit is een flipflop die op *beide* flanken reageert.

6.12. Een T-flipflop is een flipflop die van stand (waarde) verandert onder besturing van een stuursignaal T. Zie voor de functie tabel P6.1. Het (steeds weer) veranderen van stand wordt "toggelen" genoemd. Ontwerp deze T-flipflop, er mag uiteraard *niet* geschakeld worden in de kloklijn.

Tabel P6.1: *Functietabel T-flipflop.*

T	Q^{n+1}
0	Q_n
1	$\overline{Q_n}$

* **6.13.** De functie van een JK-flipflop is:

$$Q^{n+1} = J^n \cdot \overline{Q^n} + \overline{K^n} \cdot Q^n$$

Toon dit aan met behulp van een Karnaughdiagram.

6.14. Van een gated D-latch zijn gegeven t_{su}(EN-to-Z) = 15 ns, t_h(EN-to-Z) = 10 ns. De klok loopt op 5 MHz en heeft een duty cycle van 50%. Bereken de tijd dat D stabiel moet blijven als ervan uitgegaan wordt dat D niet mag veranderen tijdens het transparant zijn van de latch.

6.15. Maak een SR-latch van een edge-triggered D-flipflop met asynchrone preset en clear.

6.16. Eerder is een ontwerp van een schuifregister aan bod gekomen. Deze schakeling schuift (althans op tekening) naar rechts. Ontwerp nu een schuifregister dat zowel rechtsom als linksom kan schuiven (gebruik een stuursignaal).

Bibliografie

Veel materiaal is tegenwoordig (alleen) via Internet beschikbaar. Voorbeelden hiervan zijn de datasheets van ic's die alleen nog maar via de website van de fabrikant beschikbaar worden gesteld. Dat is veel sneller toegankelijk dan boeken en tijdschriften. De keerzijde is dat websites van tijd tot tijd veranderen of verdwijnen. De geciteerde weblinks werken dan niet meer. Helaas is daar niet veel aan te doen. Er is geen garantie te geven dat een weblink in de toekomst beschikbaar blijft.

[1] LaTeX Project Team. *LaTeX – A document preparation system*. URL: http://www.latex-project.org/ (bezocht op 26-08-2015) (blz. iii).

[2] Tex User Groups. *TeX Live Website*. URL: https://www.tug.org/texlive/ (bezocht op 26-08-2015) (blz. iii).

[3] P. Brachet. *Texmaker, The universal LaTeX editor*. URL: http://www.xm1math.net/texmaker/ (bezocht op 26-08-2015) (blz. iii).

[4] Bitstream. *Charter Fonits*. URL: https://www.ctan.org/pkg/charter (bezocht op 04-11-2016) (blz. iv).

[5] Artifex. *Nimbus 15 Mono*. URL: http://www.tug.dk/FontCatalogue/nimbus15mono/ (bezocht op 04-11-2016) (blz. iv).

[6] J. Hoffman. *listings – Typeset source code listings using LaTeX*. URL: https://www.ctan.org/pkg/listings (bezocht op 04-11-2016) (blz. iv).

[7] J.E.J. op den Brouw. *askmaps -– Typeset American style Karnaugh maps*. Dec 2013. URL: https://www.ctan.org/pkg/askmaps (bezocht op 16-08-2016) (blz. iv).

[8] T. Tantau. *pgf – Create PostScript and PDF graphics in TeX*. URL: https://www.ctan.org/pkg/pgf (bezocht op 04-11-2016) (blz. iv).

[9] J.F. Wakerly. *Digital Design: Principles and Practices*. 3de ed. Prentence Hall, 2000. ISBN: 0-13-769191-2 (blz. 2).

[10] Business Week. *The Digital Revolution Ahead for the Audio Industry*. Mrt 1981 (blz. 2).

[11] M. Broekman. *Wat is muziek streamen*. 2018. URL: http://gratismuziekdownload.nl/begrippen/wat-is-muziek-streamen/ (bezocht op 25-07-2018) (blz. 2).

[12] N. Hasted. *The rapacity of the record revival*. Aug 2012. URL: http://www.independent.co.uk/arts-entertainment/music/features/the-rapacity-of-the-record-revival-8026756.html (bezocht op 18-08-2015) (blz. 2).

[13] N. Garnham en A. Aksoy. *European Telecommunications Policy Research: Proceedings of the Communications Policy Research Conference, 22-24 June 1988, Windsor, UK*. European Communication Policy Research Series. IOS, 1989, p. 76–77. ISBN: 9789051990133 (blz. 2).

[14] H. Lintsen. *Made in Holland: een techniekgeschiedenis van Nederland (1800-2000)*. Walburg Pers, 2005, p. 249. ISBN: 9789057303494 (blz. 2).

[15] H. Lörzing. *Mijn generatie: een ode aan de babyboomer*. Singel Uitgeverijen, 2015. ISBN: 9789025300517 (blz. 2).

[16] United Consumers. *De gsm: van het ontstaan naar wereldwijd mobiel bellen*. URL: https://www.unitedconsumers.com/gsm/ontstaan-mobiele-telefoon/index.jsp (bezocht op 25-07-2018) (blz. 2).

[17] Consultancy.nl. *Smartphonebezit gegroeid naar 93% van Nederlanders, veelvuldig gebruik storend*. Jan 2018. URL: https://www.consultancy.nl/nieuws/15292/smartphonebezit-gegroeid-naar-93-van-nederlanders-v (bezocht op 25-07-2018) (blz. 3).

[18] Authoriteit Consument & Markt. *De stand van de Nederlandse postmarkt*. Dec 2015. URL: https://www.acm.nl/nl/publicaties/publicatie/15115/De-stand-van-de-Nederlandse-postmarkt/ (bezocht op 25-07-2018) (blz. 3).

[19] The Radicati Group. *Email Statistics Report, 2015-2019*. URL: http://www.radicati.com/wp/wp-content/uploads/2015/02/Email-Statistics-Report-2015-2019-Executive-Summary.pdf (bezocht op 25-07-2018) (blz. 3).

[20] R. Tomlinson. *The First Network Email*. Mrt 2015. URL: http://openmap.bbn.com/~tomlinso/ray/firstemailframe.html (bezocht op 25-07-2018) (blz. 3).

[21] D.H. Crocker. *Standard for the Format of ARPA Internet Text Messages*. Aug 1982. URL: http://www.rfc-editor.org/rfc/rfc822.txt (bezocht op 25-07-2018) (blz. 3).

[22] C. Carson. *How Much Data Does Google Store?* Nov 2014. URL: https://www.cirrusinsight.com/blog/much-data-google-store (bezocht op 25-07-2018) (blz. 3).

[23] Peter W. *Digitale TV naar 87%*. Feb 2016. URL: http://www.mediaonderzoek.nl/7114/tv-in-nederland-2015/ (bezocht op 25-07-2018) (blz. 3).

[24] Radio.NL. *FM in Nederland mogelijk in 2023 uit, mits DAB+ aanslaat*. Nov 2013. URL: http://radio.nl/784904/fm-in-nederland-mogelijk-in-2023-uit-mits-dab-aanslaat (bezocht op 25-07-2018) (blz. 3).

[25] D.A. Huffman. „A Method for the Construction of Minimum-Redundancy Codes". In: *Proceedings of the IRE* 40.9 (sep 1952), p. 1098–1101 (blz. 3).

[26] G.E. Moore. „Cramming more components onto integrated circuits". In: *Electronics* 38 (apr 1965) (blz. 5).

[27] M. Bushnell en V. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Frontiers in Electronic Testing. Springer US, 2004, p. 613. ISBN: 9780792379911 (blz. 5).

[28] „IEEE Standard for Test Access Port and Boundary-Scan Architecture". In: *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)* (mei 2013) (blz. 5).

[29] H. Bleeker, P. van den Eijnden en F. de Jong. *Boundary-Scan Test: A Practical Approach*. IEEE std. Kluwer Academic, 1993. ISBN: 9780792392965 (blz. 5).

[30] J. Graham-Cumming. *The 100-year leap*. Okt 2010. URL: http://radar.oreilly.com/2010/10/the-100-year-leap.html (bezocht op 25-07-2018) (blz. 6).

[31] *Building Charles Babbage's Anaytical Engine*. URL: http://plan28.org/ (bezocht op 25-07-2018) (blz. 6).

[32] J. Julen. „Bijna 300.000 studenten opgeleid voor de werkloosheid". In: Trouw nr. 22042 (sep 2016), p. 17 (blz. 8).

[33] H. Nyquist. „Certain Topics in Telegraph Transmission Theory". In: *American Institute of Electrical Engineers, Transactions of the* 47.2 (apr 1928), p. 617–644 (blz. 10).

[34] J.G. Rouland en J.J. Schrage. *Digitale techniek, analyse en synthese van schakelingen*. Educaboek, 1967, p. 219–241 (blz. 16).

[35] J. Vossebeld. *Digitale techniek*. 3de ed. Educaboek, 1986, p. 33–38. ISBN: 90-11-012194 (blz. 22).

[36] Texas Instruments. *Datasheet SN7408*. Mrt 1988. URL: http://www.ti.com/lit/ds/symlink/sn74s08.pdf (bezocht op 27-03-2018) (blz. 24).

[37] IEC. *Normblad 617-12, Graphical Symbols for Diagrams, Part 12: Binary Logic Elements*. Nederlands Normalisatie Instituut, 1984 (blz. 24, 192).

[38] IEEE. *IEEE Graphic Symbols for Logic Functions (Includes IEEE Std 91a-1991 Supplement, and IEEE Std 91-1984)*. 1991 (blz. 24, 192).

[39] I. Kampel. *Practical Design of Digital Circuits: Basic Logic to Microprocessors*. Elsevier Science, 1983, p. 296. ISBN: 9781483135564 (blz. 24).

[40] A.P. Thijssen en H.A. Vink. *Digitale techniek: van probleemstelling tot realisatie deel 1*. 5de ed. Delft University Press, 1999, p. 273–283. ISBN: 90-407-1793-1 (blz. 28).

[41] Fairchild. *Datasheet 74F00 Quad 2-input NAND gate*. Sep 2000. URL: http://www.farnell.com/datasheets/1498981.pdf (bezocht op 25-07-2018) (blz. 30).

[42] A.P. Thijssen, H.A. Vink en C.H. Eversdijk. *Digitale techniek, van probleemstelling tot realisatie, deel 2*. 3de ed. Delft, Zuid-Holland: Delftse Uitgevers Maatschappij, 1990, p. 11. ISBN: 90-6562-069-9 (blz. 41).

[43] L.M. Surhone, M.T. Timpledon en S.F. Marseken. *Unary Numeral System*. VDM Publishing, 2010. ISBN: 9786131120213 (blz. 41).

[44] L.E. Turner. *Roman Numeral Arithmetic*. 2007. URL: http://turner.faculty.swau.edu/mathematics/materialslibrary/roman/ (bezocht op 18-04-2018) (blz. 41).

[45] R.J. Tocci, N.S. Widmer en G.L. Moss. *Digital Systems, Principles and Applications*. 11de ed. Upper Saddle River, NJ: Pearson Educational, 2011, p. 40. ISBN: 978-0-13-038793-6 (blz. 44).

[46] U. Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Signals and Communication Technology. Springer Berlin Heidelberg, 2007, p. 538. ISBN: 9783540726135 (blz. 44).

[47] I. Matthews. *Commodore 64 – The Best Selling Computer in History*. Nov 2010. URL: http://www.commodore.ca/commodore-products/commodore-64-the-best-selling-computer-in-history/ (bezocht op 26-07-2018) (blz. 50).

[48] ThunderWare Research Center. *World of ZX Spectrum*. 2016. URL: http://www.worldofspectrum.org/ (bezocht op 26-07-2018) (blz. 50).

[49] A.P. Godse en D.A. Godse. *Digital Logic Design & applications*. 1ste ed. Technical Publications, 2008. ISBN: 9788184314717 (blz. 53).

[50] Tien Chi Chen en Irving T. Ho. „Storage-efficient Representation of Decimal Data". In: *Commun. ACM* 18.1 (jan 1975), p. 49–52. URL: http://doi.acm.org/10.1145/360569.360660 (blz. 54).

[51] C.M.L. Burnett. *A 3-bit binary rotary encoder*. Engels. URL: https://commons.wikimedia.org/wiki/File:Encoder_disc_(3-Bit_binary).svg (bezocht op 18-04-2018) (blz. 54).

[52] F. Gray. *Pulse code communication*. US Patent 2,632,058. Mrt 1953 (blz. 55).

[53] Jjbeard. *A Rotary Encoder Disc with a 3-Bit Binary Reflected Gray Code (BRGC)*. Engels. Mei 2006. URL: https://commons.wikimedia.org/wiki/File:Encoder_Disc_(3-Bit).svg (bezocht op 18-04-2018) (blz. 55).

[54] N.B. Dale en J. Lewis. *Computer Science Illuminated*. Jones & Bartlett Learning, 2013, p. 38. ISBN: 9781449665739 (blz. 56).

[55] I. Flores. *Computer design*. Prentice-Hall series in electronic technology. Prentice-Hall, 1967, p. 195 (blz. 56).

[56] H.H. Goldstine. *The Computer from Pascal to Von Neumann*. The Computer from Pascal to Von Neumann. Princeton University Press, 2008, p. 158. ISBN: 9781400820139 (blz. 56).

[57] V. Eijkhout. *An ASCII wall chart*. Nov 2010. URL: https://ctan.org/pkg/ascii-chart (bezocht op 29-07-2018) (blz. 58).

[58] American Standards Association. *ASA standard X3.4-1963*. URL: http://worldpowersystems.com/J/codes/X3.4-1963/ (bezocht op 26-07-2018) (blz. 57).

[59] A.K. Maini. *Digital Electronics: Principles, Devices and Applications*. Wiley, 2007, p. 28. ISBN: 978-0-470-03214-5 (blz. 57).

[60] The Unicode Consortium. *The Unicode Standard Version 7.0 – Core Specification*. Okt 2014. ISBN: 978-1-936213-09-2. URL: http://www.unicode.org/ (blz. 59).

[61] A. Salomaa. *Public-Key Cryptography*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 1996, p. 16. ISBN: 9783540613565 (blz. 59).

[62] Genootschap Onze Taal. *Letterfrequentie in het Nederlands*. 1988. URL: https://onzetaal.nl/taaladvies/advies/letterfrequentie-in-het-nederlands (bezocht op 26-07-2018) (blz. 59).

[63] E. Mahler. *Introductie tot de techniek der digitale rekenaars*. Kluwer, 1967, p. 14–15 (blz. 61).

[64] S. Deering en R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. Engels. Jul 2017. URL: https://www.rfc-editor.org/info/rfc8200 (bezocht op 18-04-2018) (blz. 62).

[65] G. Boole. *An Investigation of the Laws of Thought: On which are Founded the Mathematical Theories of Logic and Probabilities*. Walton en Maberly, 1854 (blz. 65).

[66] E.V. Huntington. „Sets of Independent Postulates for the Algebra of Logic". In: *Transactions of the American Mathematical Society* 5.3 (1904), p. 288–309 (blz. 65).

[67] C.E. Shannon. „A symbolic analysis of relay and switching circuits". In: *American Institute of Electrical Engineers, Transactions of the* 57.12 (dec 1938), p. 713–723 (blz. 65, 109).

[68] H. Zantema en P.W.H. Lemmens. *Beschrijven en Bewijzen*. 1ste ed. Delft University Press, 1999, p. 26, 28. ISBN: 90–407–1942–X (blz. 66).

[69] N.N. Biswas. *Logic design theory*. Prentice Hall, 1993, p. 36. ISBN: 9780135243985 (blz. 74).

[70] M. Karnaugh. „The map method for synthesis of combinational logic circuits". In: *American Institute of Electrical Engineers, Part I: Communication and Electronics, Transactions of the* 72.5 (nov 1953), p. 593–599 (blz. 89).

[71] W.V. Quine. „The Problem of Simplifying Truth Functions". In: *The American Mathematical Monthly* 59.8 (1952), p. 521–531 (blz. 99).

[72] E.J. McCluskey. „Minimization of Boolean functions". In: *The Bell System Technical Journal* 35.6 (nov 1956), p. 1417–1444 (blz. 99).

[73] S.R. Petrick. *A Direct Determination of the Irredundant Forms of a Boolean Function from the Set of Prime Implicants*. Tech. rap. TR-5-110. Bedford MA: Computation Laboratory of the AFCRC, apr 1956 (blz. 103).

[74] D. Lewin en D. Protheroe. *Design of Logic Systems*. 2de ed. Chapman & Hall, 1992, p. 93–101. ISBN: 9780412428906 (blz. 104).

[75] R.K. Brayton e.a. *Logic Minimization Algorithms for VLSI Synthesis*. 8ste ed. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1997. ISBN: 978-1-4612-9784-0 (blz. 104).

[76] R.B. Polansky. „Minimization of multiple-output switching circuits". In: *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics* 80.1 (mrt 1961), p. 67–73 (blz. 105).

[77] M.A. Karim en X. Chen. *Digital Design: Basic Concepts and Principles*. CRC Press, 2008, p. 107–108. ISBN: 978-1-4200-6131-4 (blz. 105).

[78] H.A. Farhat. *Digital Design and Computer Organization*. CRC Press, 2011, p. 106. ISBN: 978-0-203-48669-6 (blz. 106).

[79] P.K. Lala. *Principles of Modern Digital Design*. Wiley & Sons, 2007, p. 404. ISBN: 978-0-470-07296-7 (blz. 107).

[80] W. Dolman. *Digitale Systemen, Ontwerpen met behulp van VHDL*. Free Musketeers, 2011, p. 302–322. ISBN: 978-90-484-2020-9 (blz. 111).

[81] M.D. Ciletti. *Advanced Digital Design with the Verilog HDL*. Prentice Hall, 2003, p. 476–503. ISBN: 0-13-089161-4 (blz. 111).

[82] D.A. Neamen. *Microelectronics Circuit Analysis and Design*. 4de ed. McGraw-Hill Education, 2010, p. 126–141, 1168–1182. ISBN: 978–0–07–338064–3 (blz. 117).

[83] M.H. Rashid. *Microelectronic Circuits Analysis and Design*. 2de ed. Cengage Learning, 2011, p. 1016–1022. ISBN: 978-0-495-66772-8 (blz. 118).

[84] NXP. *I^2C-bus specification and user manual*. Apr 2014. URL: http://www.nxp.com/documents/user_manual/UM10204.pdf (bezocht op 18-08-2016) (blz. 120).

[85] NXP. *2N7002, 60 V, 300 mA N-channel Trench MOSFET*. sep 2011. URL: https://www.nxp.com/documents/data_sheet/2N7002.pdf (bezocht op 05-02-2017) (blz. 120).

[86] Linear Technonlogy. *Design Simulation and Device Modelsl*. 2016. URL: http://www.linear.com/designtools/software/ (bezocht op 21-08-2016) (blz. 120).

[87] B. Holdsworth en C. Woods. *Digital Logic Design*. 4de ed. Elsevier Science, 2002, p. 268–283. ISBN: 9780080477305 (blz. 126).

[88] J. Knight. *Glitches and Hazards in Digital Circuits*. Mrt 2001. URL: http://www.doe.carleton.ca/~shams/ELEC3500/hazards.pdf (bezocht op 28-07-2018) (blz. 126).

[89] W. Kunz en D. Stoffel. *Reasoning in Boolean Networks: Logic Synthesis and Verification Using Testing Techniques*. Frontiers in Electronic Testing. Springer US, 1997, p. 94. ISBN: 978-1-4419-5176-2 (blz. 126).

[90] S. Muroga e.a. „The Transduction Method-Design of Logic Networks Based on Permissible Functions". In: *IEEE Trans. Comput.* 38.10 (okt 1989), p. 1404–1424 (blz. 126).

[91] A.P. Thijssen en H.A. Vink. *Digitale techniek: van probleemstelling tot realisatie deel 1*. 5de ed. Delft University Press, 1999, p. 195. ISBN: 90-407-1793-1 (blz. 126).

[92] J.E. Stine. *Digital Computer Arithmetic Datapath Design Using Verilog HDL*. Springer Science, 2004, p. 56–68. ISBN: 978-1-4613-4725-5 (blz. 147).

[93] R.F. Tinder. *Engineering Digital Design*. Academic Press, 2000, p. 353–357. ISBN: 0-12-691295-5 (blz. 150).

[94] K.C. Chang. *Digital Systems Design with VHDL and Synthesis, An Integrated Approach*. IEEE Computer Society, 1999, p. 445–452. ISBN: 0-7695-0023-4 (blz. 150).

[95] F. Helwig. *CODING for the MIT-IBM 704 COMPUTER*. okt 1957, p. I–9. URL: http://www.textfiles.com/bitsavers/pdf/mit/computer_center/Coding_for_the_MIT-IBM_704_Computer_Oct57.pdf (bezocht op 11-02-2016) (blz. 152).

[96] A.P. Thijssen, H.A. Vink en C.H. Eversdijk. *Digitale techniek, van probleemstelling tot realisatie, deel 2*. 1ste ed. Delft, Zuid-Holland: Delftse Ubiversitaire Pers, 1995, p. 29–32. ISBN: 90-407-1149-6 (blz. 161).

[97] A.P. Thijssen, H.A. Vink en C.H. Eversdijk. *Digitale techniek, van probleemstelling tot realisatie, deel 2*. 3de ed. Delft, Zuid-Holland: Delftse Uitgevers Maatschappij, 1990, p. 91–93. ISBN: 90-6562-069-9 (blz. 171).

[98] A.D. Booth. „A Signed Binary Multiplication Technique". In: *The Quarterly Journal of Mechanics and Applied Mathematics* 4.2 (1951), p. 236–240. URL: http://qjmam.oxfordjournals.org/content/4/2/236.abstract (blz. 171).

[99] O.L. Macsorley. „High-Speed Arithmetic in Binary Computers". In: *Proceedings of the IRE* 49.1 (jan 1961), p. 67–91 (blz. 171).

[100] L.P. Rubinfield. „A Proof of the Modified Booth's Algorithm for Multiplication". In: *IEEE Transactions on Computers* C-24.10 (okt 1975), p. 1014–1015 (blz. 171).

[101] J. Cavanagh. *Computer Arithmetic and Verilog HDL Fundamentals*. CRC Press, 2009, p. 289–304. ISBN: 9781439811276 (blz. 171).

[102] J.Y. Hsu. *Computer Architecture: Software Aspects, Coding, and Hardware*. CRC Press, 2001, p. 53. ISBN: 9781420041101 (blz. 172).

[103] A.P. Thijssen en H.A. Vink. *Digitale techniek: van probleemstelling tot realisatie deel 1*. 5de ed. Delft University Press, 1999, p. 324–326. ISBN: 90-407-1793-1 (blz. 172).

[104] Atmel. *Atmel AVR 8-bit Instruction Set*. Jul 2014. URL: http://www.atmel.com/images/atmel-0856-avr-instruction-set-manual.pdf (bezocht op 11-02-2016) (blz. 172).

[105] Texas Instruments. *4-bit Binary Full Adders with Fast Carry*. Apr 1988. URL: http://www.ti.com/lit/ds/symlink/sn74s283.pdf (bezocht op 30-07-2018) (blz. 173).

[106] A.P. Thijssen en H.A. Vink. *Digitale techniek: van probleemstelling tot realisatie deel 1*. 5de ed. Delft University Press, 1999, p. 366–369. ISBN: 90-407-1793-1 (blz. 173).

[107] W. Padgett en D. Anderson. *Fixed-Point Signal Processing*. Synthesis Lectures on Signal Processing. Morgan & Claypool Publishers, 2009, p. 41. ISBN: 9781598292596 (blz. 178).

[108] A.P. Thijssen en H.A. Vink. *Digitale techniek: van probleemstelling tot realisatie deel 1*. 5de ed. Delft University Press, 1999, p. 57. ISBN: 90-407-1793-1 (blz. 178).

[109] „IEEE Standard for Floating-Point Arithmetic". In: *IEEE Std 754-2008* (aug 2008) (blz. 178).

[110] D.E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison Wesley Longman, 1998, p. 229–242. ISBN: 0-201-89684-2 (blz. 179).

[111] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford series in electrical and computer engineering. Oxford University Press, 2000, p. 279–291. ISBN: 0-19-512583-5 (blz. 179).

[112] D.A. Huffman. „The synthesis of sequential switching circuits". In: *Journal of the Franklin Institute* 257.3 (1954), p. 161–190 (blz. 184).

[113] R.H. Katz en G. Borriello. *Contemporary Logic Design*. 2de ed. Pearson Education, 2005, p. 232–233. ISBN: 9780131278301 (blz. 189).

[114] Z. Navabi. *Digital System Test and Testable Design: Using HDL Models and Architectures*. SpringerLink : Bücher. Springer, 2010, p. 85. ISBN: 978-1-4419-7547-8 (blz. 192).

[115] A.P. Thijssen en H.A. Vink. *Digitale Techniek, van probleemstelling tot realisatie deel 2*. 5de ed. Delft University Press, 2000, p. 44–47. ISBN: 90-407-1838-5 (blz. 199).

[116] M.M. Mano. *Digital logic and computer design*. Prentice-Hall, 1979, p. 208–209. ISBN: 0-13-214510-3 (blz. 199).

[117] A.W. Shaw. *Logic Circuit Design*. International Edition. Saunders, 1993, p. 357. ISBN: 978-0-03-097498-4 (blz. 199).

[118] J.G. Rouland en J.J. Schrage. *Digitale techniek: analyse en synthese van schakelingen*. Educaboek-Stam Technische Boeken, 1983, p. 122–125. ISBN: 90-11-00454X (blz. 205).

[119] NXP. *Datasheet 74F74 Dual D-type flip-flop*. Mrt 1996. URL: http://www.nxp.com/documents/data_sheet/74F74.pdf (bezocht op 13-08-2016) (blz. 205).

[120] A.P. Godse en D.A. Godse. *Logic Design*. Technical Publications, 2009, p. 5-15–5-18. ISBN: 9788184312768 (blz. 205).

[121] W. Dolman. *De taal C en de Xmega*. Free Musketeers, 2015, p. 214–216. ISBN: 978-90-484-3527-2 (blz. 207).

[122] J. Lenk. *Simplified Design of Microprocessor-Supervisory Circuits*. EDN Series for Design Engineers. Elsevier Science, 1998. ISBN: 9780080517193 (blz. 208).

[123] Maxim Integrated. *MAX791 Microprocessor Supervisory Circuit*. Mei 2005. URL: https://datasheets.maximintegrated.com/en/ds/MAX791.pdf (bezocht op 17-10-2015) (blz. 208).

[124] V.G. Oklobdzija e.a. *Digital System Clocking: High-Performance and Low-Power Aspects*. John Wiley & Sons, 2003, p. 112–114. ISBN: 0-471-27447-X (blz. 208).

[125] N. Tripathi. *Reduce Power in Chip Designs with Sequential Clock Gating*. Dec 2012. URL: http://electronicdesign.com/power/reduce-power-chip-designs-sequential-clock-gating (bezocht op 18-10-2015) (blz. 208).

[126] Stackexchange. *Stopping the clock without gating the clock*. URL: http://electronics.stackexchange.com/questions/19966/stopping-the-clock-without-gating-the-clock (bezocht op 18-10-2015) (blz. 208).

[127] H. Kaeslin. *Digital Integrated Circuit Design: From VLSI Architectures to CMOS Fabrication*. Cambridge University Press, 2008, p. 353–357. ISBN: 978-0-521-88267-5 (blz. 208).

[128] A.P. Thijssen en H.A. Vink. *Digitale techniek: van probleemstelling tot realisatie deel 2*. 5de ed. Delft University Press, 2000, p. 170–175. ISBN: 90-407-1838-5 (blz. 208).

[129] V.G. Oklobdzija. *The Computer Engineering Handbook*. Computer Engineering Handbook 2e. Taylor & Francis, 2001, p. 2–10. ISBN: 9780849308857 (blz. 209).

[130] J. Rose e.a. „Architecture of field-programmable gate arrays: the effect of logic block functionality on area efficiency". In: *IEEE Journal of Solid-State Circuits* 25.5 (okt 1990), p. 1217–1225 (blz. 225).

A

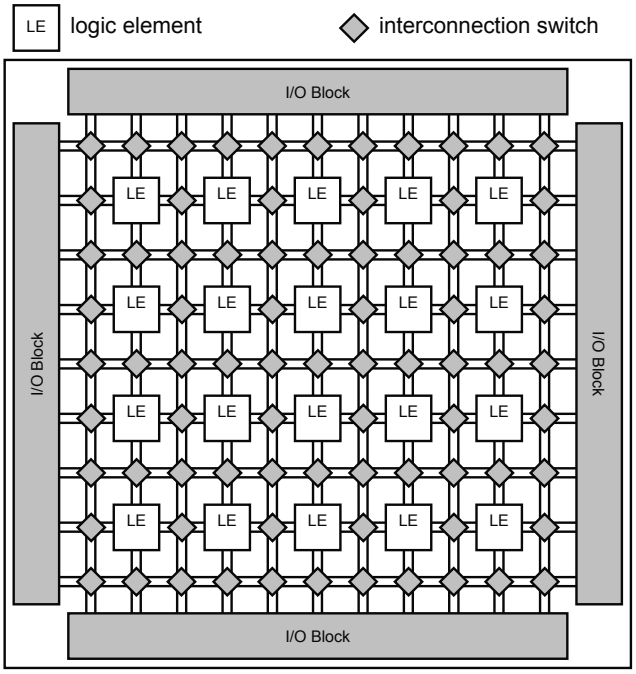
De FPGA

De ontwerper heeft de keuze om een digitaal systeem te realiseren met configureerbare ic zoals een FPGA (Field Programmable Gate Array) of met een ASIC (Application Specific Integrated Circuit). Een FPGA is een configureerbaar ic waarin de de schakeling wordt opgebouwd met behulp van *cellen*. De cellen vervullen elk (een deel van) een logische functie zodat het geheel de juiste werking vertoont. Bij een ASIC worden in de fabriek de transistoren (daar bestaat een ic namelijk uit) direct om te juiste plaats gelegd. De inhoud van de ASIC is dus na productie niet te veranderen terwijl in een FPGA een nieuwe configuratie kan worden geladen als dat nodig mocht zijn.

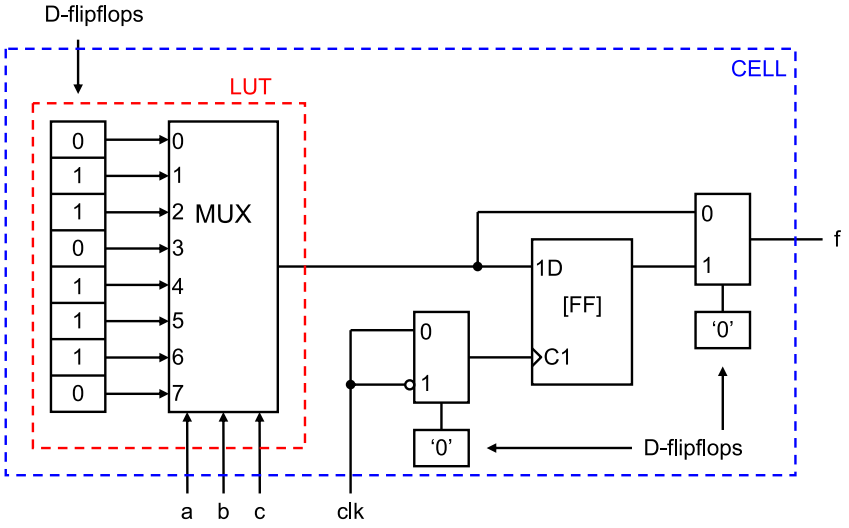
FPGA's hebben een behoorlijk verschillende architectuur ten opzichte van PAL's, PLA's en ROM's omdat FPGA's niet bestaan uit AND- en OR-matrixen maar uit cellen. In figuur A.1 is de globale opbouw van een FPGA te zien. Er zijn drie typen componenten te onderscheiden. Aan de randen van het ic zijn de zogenoemde *I/O Blocks* geplaatst. Deze componenten verzorgen de communicatie met de kern van de FPGA en de buitenwereld. De I/O Blocks bevatten vaak speciale voorzieningen zoals Schmitt-trigger ingangen, flipflops die metastabiel gehard zijn, tri-state en open collecter uitgangen, *line drivers* voor hoge snelheidsbussen en voorzieningen om een keur van spanningsniveaus aan te kunnen (denk hierbij aan TTL en CMOS).

De kern bestaat uit een matrix van programmeerbare kruispunten (*Interconnection Switch*) en logische elementen (*Logic Elements*). Die laatste worden ook wel *cellen* genoemd. Elke cel vervult een logische functie zodanig dat het geheel de juiste werking vertoont. Met behulp van de programmeerbare kruispunten kunnen de diverse cellen via verbindingen (*Interconnect*) met elkaar verbonden worden.

Elke cel bestaat uit een programmeerbare multiplexer en een flipflop. In figuur A.2 in een vereenvoudigd voorbeeld van een cel te zien. De multiplexer realiseert een combinatorische functie en wordt ook wel een Look Up Table (LUT) genoemd. Met behulp van configuratiebits wordt de functie vastgelegd. De stuuringangen van de multiplexer zijn verbonden met de ingangen van de functie.



Figuur A.1: *Globale opbouw FPGA.*



Figuur A.2: *Schematische weergave van een cel in een FPGA.*

Twee andere configuratiebits zorgen ervoor dat de cel te gebruiken is als een stukje combinatorische logica (zoals in de figuur te zien is) of als een stukje sequentiële logica. Daarnaast is de flank waarop de flipflop reageert met een configuratiebit in te stellen.

FPGA's worden gebruikt als een systeem complex is, time-to-market kort is, de prestatie (bijvoorbeeld snelheid) het toelaat om een FPGA te gebruiken en het aantal te bouwen systemen beperkt is. Schakelingen kunnen snel worden ontworpen met een HDL (Hardware Description Language). FPGA leveranciers leveren software tools om de HDL-beschrijving van een schakeling te synthetiseren naar een een schakeling die volledig bestaat uit *primitieven*. Voorbeelden van primitieven zijn poorten, multiplexers,

optellers, aftrekkers, vergelijkers en flipflops. Deze beschrijving van primitieven moeten dan in de cellen geplaatst worden (fitting). Daarna wordt een configuratiebestand gegenereerd die in de FPGA geladen wordt.

Fabrikanten brengen op de FPGA's vaak speciale componenten aan. Te denken van aan vermenigvuldigers, DSP's (Digital Signal Processors), Ethernet-modules en RAM. Met behulp van de ontwikkelsoftware kan een ontwerper deze componenten gebruiken. Het voordeel van de speciale componenten is dat er geen logische cellen gebruikt worden om een component te implementeren en dat de componenten sneller zijn dan wanneer ze met cellen worden gerealiseerd.

In de praktijk komen LUT's met vier en zes (stuur-)ingangen voor. Daarmee zijn logische functies met vier respectievelijk zes variabelen te realiseren. Rose toont in [130] aan dat een aantal in de industrie gebruikte ontwerpen het best kunnen worden gerealiseerd met LUT's van drie of vier variabelen en dat het efficiënt is om een cel met een flipflop uit te rusten.

FPGA's zijn er in alle soorten en maten. De kleinste FPGA's hebben zo'n 10.000 cellen en de grootste zo'n 500.000. Veelal zijn snelle en langzame versies te krijgen (de zogenoemde *speed grade*). Ook qua verpakkingen is er nogal een verschil te vinden. Bij de zogenoemde Ball Grid Array (BGA) zijn aan de onderkant van het ic halve bollen te vinden. Bij de Pin Grid Array (PGA) is de onderkant voorzien van pennen die door de print heen steken. Een andere verpakking is Thin Quad Flat Package (TQFP). Hierbij is het ic voorzien van aansluitingen aan de zijkanten van de behuizing van het ic.

Index