Opgaven

en uitwerkingen bij het boek Digitale Techniek

Jesse op den Brouw

Deel 1

Versie: 0.99pl4
Datum: 9 oktober 2018

DE HAAGSE HOGESCHOOL

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

Inhoudsopgave

1

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.

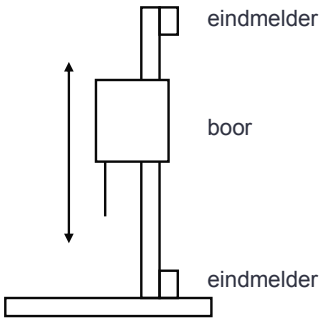
1.3. Ontwerp een NOT-schakeling met een NOR-poort.

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?

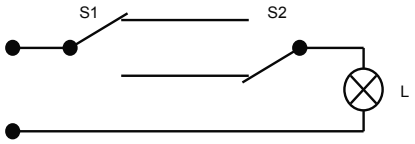
1.7. Een automatische kolomboor (figuur P1.1) heeft twee eindmelders: een aan de bovenkant en een aan de onderkant. Een melder kan open zijn (boor is daar niet) of gesloten zijn (boor is daar wel).



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.

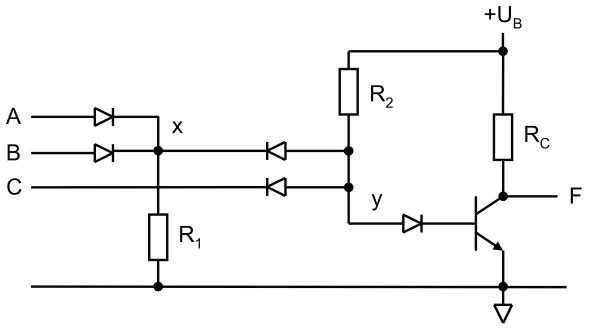


schakelaar in rust: 0
schakelaar geactiveerd: 1
lamp uit: 0
lamp brandt: 1

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

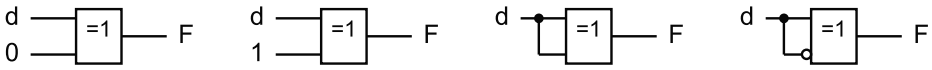
Geef de waarheidstabel van de wisselschakeling.

* **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.

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.

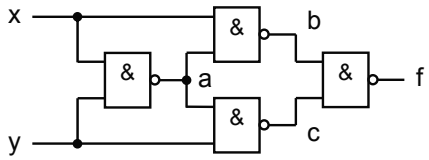
Tabel P1.1: *Drie waarheidstabellen.*

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

1.16. 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.

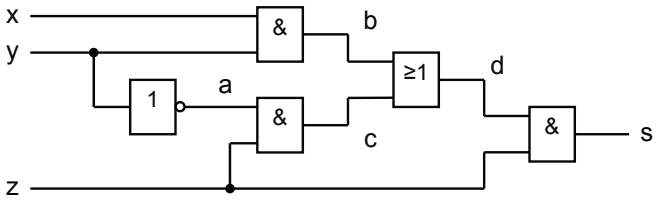
1.18. Gegeven de waarheidstabel in tabel P1.2.

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

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 [1].

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 $\quad$ 1F,3C$_{16} \rightarrow$ dec $\quad$ 3F,EA$_{16} \rightarrow$ bin $\quad$ $110110{,}110111_2 \rightarrow$ 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

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.

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

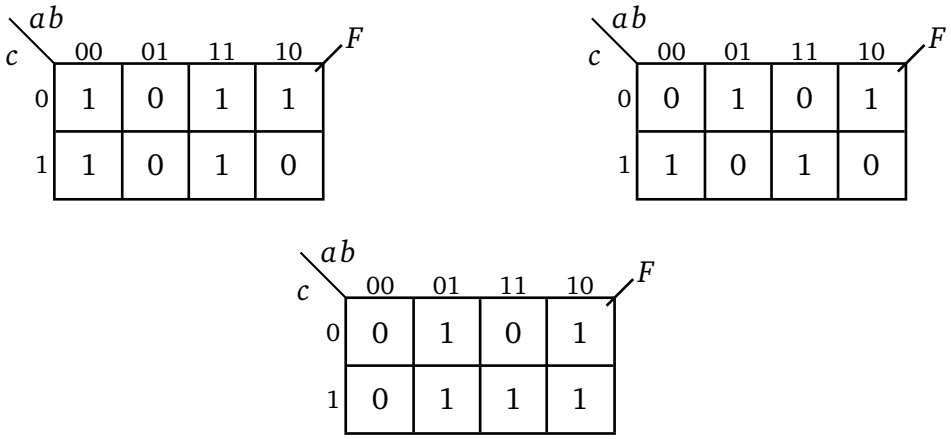
3.11. Gegeven een functie van drie variabelen waarvoor geldt dat de functie voor $xyz = 000$ en $xyz = 111$ don't care is en dat de functie een logische 1 geeft als $x = 0$ terwijl $z = 1$, anders is de functie logisch 0. Geef de waarheidstabel.

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

4

4.1. Minimaliseer de functies gegeven in de Karnaughdiagrammen in figuur P4.1:



Figuur P4.1: *Karnaughdiagrammen met drie variabelen.*

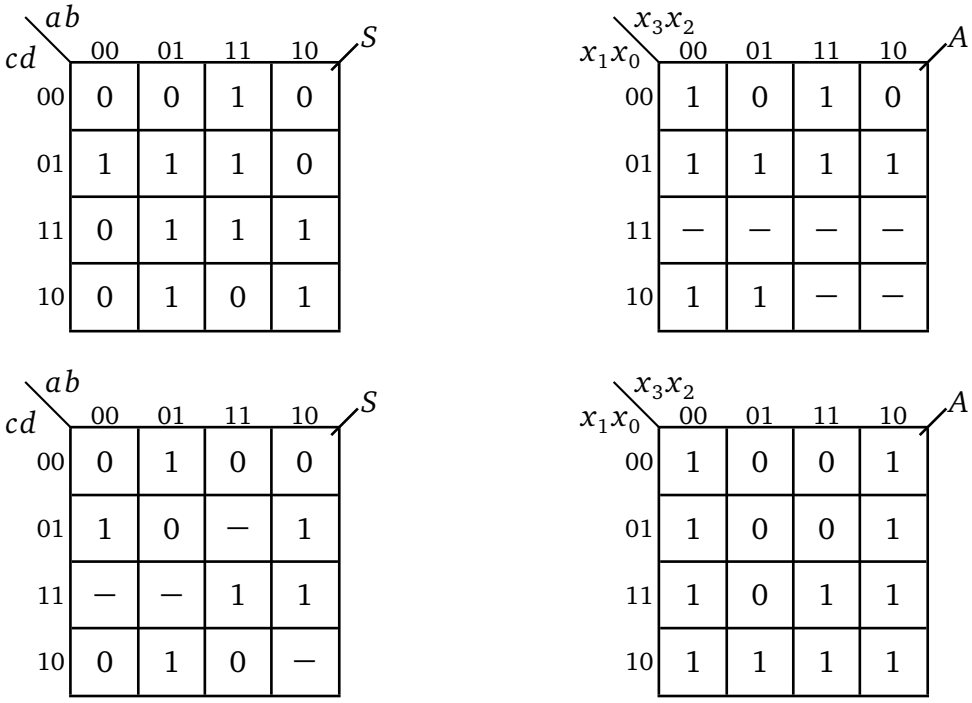
4.2. Gegeven de functies:

$$\begin{aligned} f_{x,y,z} &= \sum m(0,1,2,6) \\ f_{a,b,c} &= \sum m(1,2,5,6,7) \\ f_{s_2,s_1,s_0} &= \sum m(0,3,6) + d(1,2) \end{aligned} \tag{P4.1}$$

Stel de Karnaughdiagrammen op en geef de geminimaliseerde functies. Teken de bijbehorende schakelingen met NOT, AND en OR.

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.*

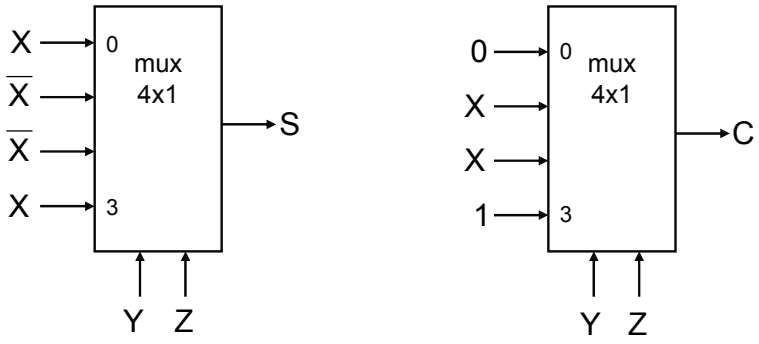
4.5. Gegeven de functies:

$$\begin{aligned} f_{w,x,y,z} &= \sum m(1,2,3,5,7,10,11,12,15) \\ f_{a,b,c,d} &= \sum m(6,7,8,12,14,15) \\ f_{s_3,s_2,s_1,s_0} &= \sum m(4,5,9,10,12) + d(7,8,11,15) \\ f_{x_3,x_2,x_1,x_0} &= \prod M(1,2,3,5,7,10) + D(11,12,15) \end{aligned} \tag{P4.2}$$

Minimaliseer met behulp van Karnaughdiagrammen naar een SOP-vorm.

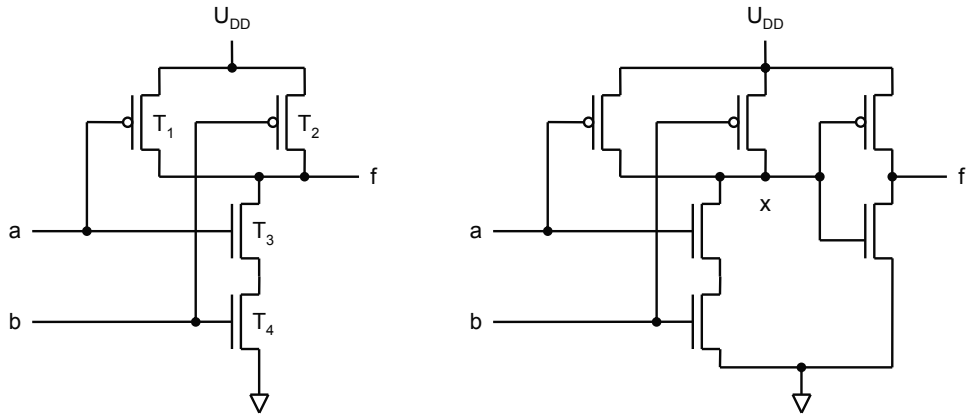
* **4.6.** Gegeven de functies in opgave 4.5. Minimaliseer met behulp van de Quine-McCluskey-methode naar een SOP-vorm.

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

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:

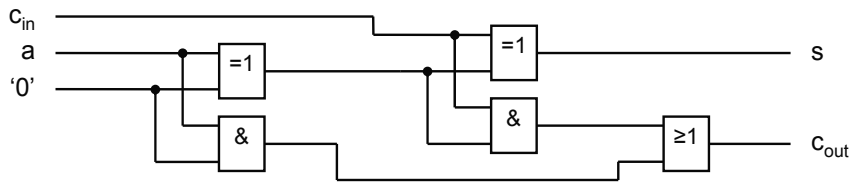
$1011001 - 0111011 \qquad 110000 - 010011 \qquad 010110 - 100101$

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.



Figuur P5.1: *Full adder met ingang b aangesloten op een logische 0.*

5.7. Ontwerp een 2x2-bits unsigned vermenigvuldiger. Stel de waarheidstabel op en leid de schakelfuncties af.

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:

$10011001 + 00111001 \quad 0011 + 0111 \quad 010110000110 + 000001110001$

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

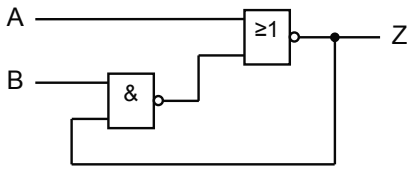
op 0. De auto rijdt 123 kilometer achteruit. Wat is de kilometerstand als de kilometerteller vijf decimale cijfers heeft? Ga ervan uit dat de kilometerteller terug telt.

* **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

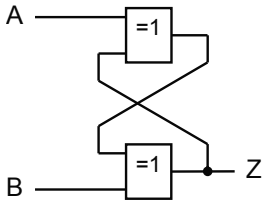
6.1. De SR-latches met overheersende set en reset zijn reeds behandeld. De don't cares waren dan ingevuld als respectievelijk twee enen en twee nullen. Het zijn twee don't cares, dus er zijn vier combinaties mogelijk. Ontwerp SR-latches voor de twee overgebleven combinaties. Zijn de ontwerpen zinvol?

6.2. Gegeven is de schakeling in figuur P6.1. Realiseert deze schakeling een goedwerkend geheugenelement? Motiveer het antwoord.



Figuur P6.1: *Een poging om een latch te realiseren.*

6.3. Gegeven is de schakeling in figuur P6.2. Realiseert deze schakeling een goedwerkend geheugenelement? Motiveer het antwoord.



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.

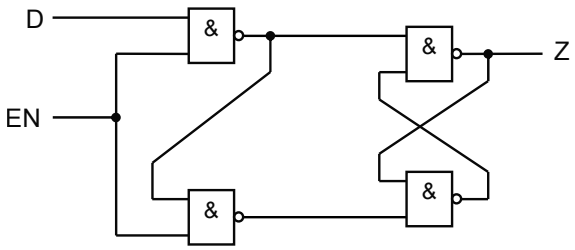
$SR = 11$ onthouden
$SR = 10$ niet gebruikt

$SR = 01$ 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.*

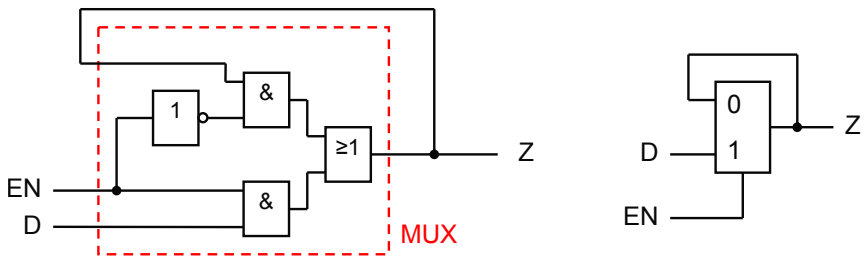
6.8. Een *tweedeler* is een schakeling die op commando van een enable-puls geacht wordt eenmaal van uitgangswaarde te veranderen. Als de uitgang 1 is wordt deze 0 en omgekeerd. Is een tweedeler te maken met één latch en poorten? En met twee latches en poorten? Motiveer het antwoord.

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).

A

Uitwerkingen

Uitwerking opgave 1.1.
Eerst maar eens een inventarisatie van de signalen. Een ingangssignaal moet óf onveranderd worden doorgelaten óf geïnverteerd worden. Er is een signaal nodig dat de data kan aanleveren. De keuze tussen doorlaten of inverteren moet met een tweede signaal gedaan worden, een zogenaamd *stuur*signaal. Natuurlijk is er één uitgang. Het datasignaal noemen we a, het stuursignaal noemen we m (*mode*). De uitgang noemen we f.

We maken een keuze in de besturing: signaal a wordt onveranderd doorgelaten als $m = 0$ en geïnverteerd als $m = 1$. De functie is als volgt te schrijven:

$$f = \begin{cases} a & \text{als } m = 0 \\ \overline{a} & \text{als } m = 1 \end{cases} \tag{A.1}$$

De waarheidstabel en de functie zijn eenvoudig op te stellen en af te leiden.

Tabel A.1: *Waarheidtabel voor functie f*

m	a	f	
0	0	0	a doorgeven
0	1	1	”
1	0	1	a inverteren
1	1	0	”

De functie levert twee enen die niet te combineren zijn:

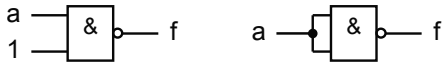
$$\begin{aligned} s_{m,a} &= \overline{a} \cdot m + a \cdot \overline{m} \\ &= a \oplus m \end{aligned} \tag{A.2}$$

De omschakelbare buffer/NOT is de bekende EXOR-functie!

Uitwerking opgave 1.2.

Een NAND kan je op twee manieren schakelen als een NOT (of inverter). De functie van de NAND is $f = \overline{x \cdot y}$. Een mogelijkheid is de signalen x en y met elkaar te verbinden (we noemen het signaal nu a). Een andere mogelijkheid is om één ingang aan een logische 1 te verbinden (de andere ingang noemen we nu ook a). Aldus:

$$x = a, y = a \quad \rightarrow \quad f = \overline{a \cdot a} = \overline{a}$$
$$x = a, y = 1 \quad \rightarrow \quad f = \overline{a \cdot 1} = \overline{a}$$

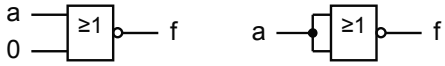


Figuur A.1: *Functies en aansluitingen van een NAND geschakeld als NOT.*

Uitwerking opgave 1.3.

Een NOR kan je op twee manieren schakelen als een NOT (of inverter). De functie van de NOR is $f = \overline{x + y}$. Een mogelijkheid is de signalen x en y met elkaar te verbinden (we noemen het signaal nu a). Een andere mogelijkheid is om één ingang aan een logische 0 te verbinden (de andere ingang noemen we nu ook a). Aldus:

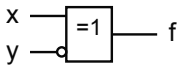
$$x = a, y = a \quad \rightarrow \quad f = \overline{a + a} = \overline{a}$$
$$x = a, y = 0 \quad \rightarrow \quad f = \overline{a + 0} = \overline{a}$$



Figuur A.2: *Functies en aansluitingen van een NOR geschakeld als NOT.*

Uitwerking opgave 1.4.

In onderstaande waarheidtabel is de functie gegeven van een EXOR waarvan één ingang (y) geïnverteerd wordt aangeboden. De verkregen functie is die van de EXNOR.



x	y	$\overline{y}$	$f = x \oplus \overline{y}$
0	0	1	1
0	1	0	0
1	0	1	0
1	1	0	1

Figuur A.3: *Aansluitingen en waarheidstabel van een EXOR met één geïnverteerde ingang.*

Uitwerking opgave 1.5.

De AND is als volgt met nullen te beschrijven:

De uitgang is logisch 0 als één of meer ingangen logisch 0 zijn.

Een andere is:

De uitgang is logisch 0 als de ene ingang logisch 0 is óf de andere ingang logisch 0 is óf beide ingangen logisch 0 zijn.

In beide gevallen is duidelijk de term “of” te lezen: vanuit de 0-en gezien is het een OR-functie.

Uitwerking opgave 1.6.

Met twee variabelen kunnen vier combinaties gemaakt worden. Dat levert o.a. de bekende AND- en OR-functies op. Maar er zijn meer functies te maken: NAND, NOR, EXOR en NOT. In feite kunnen de vier functiewaarden de combinaties 0000 t/m 1111 aannemen. Met twee variabelen zijn dus zestien functies te maken.

Uitwerking opgave 1.7.

Er zijn twee eindmelders, we noemen ze E_T (top) en E_B (bottom). Elk van deze melders kan óf open óf gesloten zijn, per melder dus twee mogelijkheiden. In totaal zijn er vier mogelijkheden of *combinaties*. Eén combinatie komt in de praktijk (hopelijk) niet voor. We stellen een tabel op om wat meer duidelijkheid te krijgen.

Tabel A.2: *Functietabel van de eindmelders van de kolomboor.*

E_T	E_B	positie boor
open	open	boor is tussen de melders
open	gesloten	boor is beneden
gesloten	open	boor is boven
gesloten	gesloten	kan niet voorkomen

Het mag duidelijk zijn dat de situatie van twee gesloten eindmelders niet voor kan komen. Dit duidt op een systeemfout.

Uitwerking opgave 1.8.

We stellen een tabel op voor open en gesloten schakelaars en voor gedoofde of brandende lamp. We stellen een identieke tabel op maar nu vervangen we de termen “open” en “gedoofd” met een 0 en de termen “gesloten” en “brandt” met 1.

Tabel A.3: *Functietabel en waarheidstabel van de schakeling.*

S_1	S_2	L
open	open	gedoofd
open	gesloten	brandt
gesloten	open	brandt
gesloten	gesloten	gedoofd

S_1	S_2	L
0	0	0
0	1	1
1	0	1
1	1	0

Te zien is dat de schakeling de EXOR-functie realiseert.

Uitwerking opgave 1.9.

Eerst maar eens de schakeling goed bekijken. Links is een OR-schakeling van de signalen A en B te zien. De uitgang van de OR (hulpsignaal x) is samen met signaal C verwerkt tot

Tabel A.4: *Waarheidstabel OR-AND-schakeling met diodes.*

A	B	C	x	y	F	geleiden/sperren diodes
0	0	0	0	0	1	A en B sperren, C geleidt
0	0	1	0	0	1	A, B en C sperren
0	1	0	1	0	1	A spert, B en C geleiden
0	1	1	1	1	0	A en C sperren, B geleidt
1	0	0	1	0	1	A en C geleiden, B spert
1	0	1	1	1	0	A geleidt, B en C sperren
1	1	0	1	0	1	A, B en C geleiden
1	1	1	1	1	0	A en B geleiden, C spert

een AND-schakeling. De laatste stap is een NOT-schakeling van hulpsignaal y. Hieronder is de waarheidstabel gegeven.

Uitwerking opgave 1.10.
De functie van de EXOR is $f = x \oplus y$. Aan ingang x wordt de variabele d gekoppeld en aan ingang y worden achtereenvolgens de waarden 0, 1, d en $\overline{d}$ gekoppeld. Hierdoor vereenvoudigd de functie f. Zie onderstaande tabel.

$$\begin{aligned} x = d, y = 0 &\rightarrow f = d \oplus 0 = d \\ x = d, y = 1 &\rightarrow f = d \oplus 1 = \overline{d} \\ x = d, y = d &\rightarrow f = d \oplus d = 0 \\ x = d, y = \overline{d} &\rightarrow f = d \oplus \overline{d} = 1 \end{aligned} \tag{A.3}$$

Uitwerking opgave 1.11.
De functie van de EXNOR is $f = \overline{x \oplus y}$. Aan ingang x wordt de variabele d gekoppeld en aan ingang y worden achtereenvolgens de waarden 0, 1, d en $\overline{d}$ gekoppeld. Hierdoor vereenvoudigd de functie f. Zie onderstaande tabel.

$$\begin{aligned} x = d, y = 0 &\rightarrow f = \overline{d \oplus 0} = \overline{d} \\ x = d, y = 1 &\rightarrow f = \overline{d \oplus 1} = d \\ x = d, y = d &\rightarrow f = \overline{d \oplus d} = 1 \\ x = d, y = \overline{d} &\rightarrow f = \overline{d \oplus \overline{d}} = 0 \end{aligned} \tag{A.4}$$

(Zoooooo, deze uitwerking lijkt wel heel erg op die van opgave 1.10.)

Uitwerking opgave 1.12.
Een EXOR omzetten naar een EXNOR kan op twee manieren: inverteren van de uitgang van een EXOR of inverteren van één ingang van een EXOR (maakt niet uit welke ingang).

Het kan ook met behulp van de schakelalgebra bewezen worden.

$$
\begin{aligned}
f &= x \oplus \overline{y} && \text{de functie met één geïnverteerde ingang} \\
&= x\overline{\overline{y}} + \overline{x}\,\overline{y} && \text{uitschrijven in AND, OR en NOT} \\
&= xy + \overline{x}\,\overline{y} && \text{dubbele inverse verwijderen} \\
&= \overline{\overline{xy} \cdot \overline{\overline{x}\,\overline{y}}} && \text{DeMorgan toepassen} \\
&= \overline{(\overline{x} + \overline{y}) \cdot (x + y)} && \text{DeMorgan toepassen} \\
&= \overline{\overline{x}x + \overline{x}y + x\overline{y} + y\overline{y}} && \text{uitvermenigvuldigen} \\
&= \overline{0 + \overline{x}y + x\overline{y} + 0} && \text{reductie inversen} \\
&= \overline{\overline{x}y + x\overline{y}} && \text{de EXNOR-functie} \\
&= \overline{x \oplus y} && \text{geschreven als een NOT-EXOR-functie}
\end{aligned}
\tag{A.5}
$$

De eerste en de laatste functie zijn resp. een EXOR met één geïnverteerde ingang en een EXOR met geïnverteerde uitgang.

Uitwerking opgave 1.13.

Bij de linker waarheidstabel valt op dat de uitgang y 1 is als ingang a 0 is. De uitgang is ook 1 als ingang b 0 is. De functie voor deze tabel is $y = \overline{a} + \overline{b}$. Merk op dat dit de tabel is van een NAND-poort.

Bij de middelste waarheidstabel valt op dat de uitgang y 1 is als ingang a 1 is. De uitgang is ook 1 als ingang b 0 is. De functie voor deze tabel is $y = a + \overline{b}$.

De rechter tabel levert een 1 als a 0 en b 1 is. Daarnaast levert de tabel een 1 als a 1 en b 0 is. Deze twee (uitgangs-)enen zijn niet te combineren tot een kleinere functie. Merk op dat dit de tabel van een EXOR-poort is.

Uitwerking opgave 1.14.

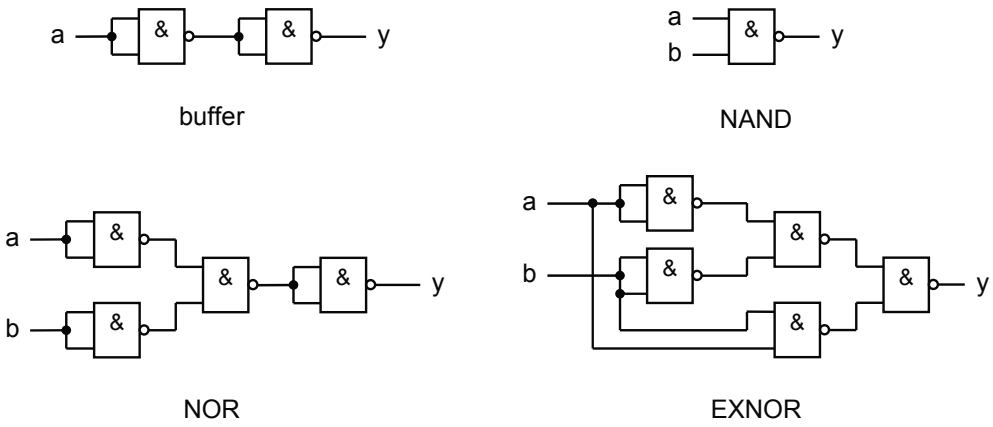
In deze opgave moet een aantal functies worden gerealiseerd met alleen NANDS. De buffer is natuurlijk te realiseren met twee NANDs geschakeld als inverters. De NAND is te maken met één NAND (echt waar!). De NOR en de EXNOR verdienen wat meer aandacht.

De NOR is te schrijven als $y = \overline{a + b} = \overline{\overline{a} \cdot \overline{b}}$ waarbij diverse NANDs geschakeld zijn als inverter.

De EXNOR is wat lastiger:

$$
y = \overline{a \oplus b} = a \oplus \overline{b} = \overline{a} \cdot \overline{b} + a \cdot b = \overline{\overline{\overline{a} \cdot \overline{b}} \cdot \overline{a \cdot b}}
\tag{A.6}
$$

De schema's zijn in figuur A.4 getekend.



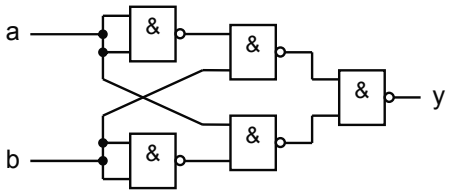
Figuur A.4: *Elementaire poorten opgebouwd uit NAND-poorten.*

Uitwerking opgave 1.15.

De functie is met behulp van De Morgen eenvoudig om te zetten in NANDs:

$$y = \overline{a} \cdot b + a \cdot \overline{b} = \overline{\overline{\overline{a} \cdot b} \cdot \overline{a \cdot \overline{b}}} \tag{A.7}$$

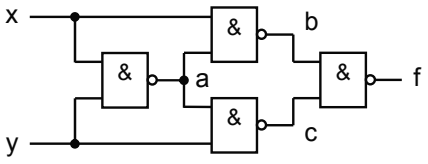
De termen $\overline{a}$ en $\overline{b}$ kunnen gemaakt worden met een NAND geschakeld als inverter. Er zijn in totaal dan vijf NAND-poorten nodig. Zie het schema hieronder.



Figuur A.5: *Schema van een EXOR-poort opgebouwd uit NAND-poorten*

Uitwerking opgave 1.16.

Hieronder is nogmaals de schakeling opgebouwd met vier NAND-poorten gegeven. Bij de interne knooppunten zijn de variabelen a, b en c geplaatst. D.m.v. een waarheidstabel met de functiewaarden van alle uitgangen kan de functie van f bepaald worden.



x	y	a	b	c	f
0	0	1	1	1	0
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	1	1	0

Figuur A.6: *Schema en waarheidstabel van de schakeling met vier NAND-poorten.*

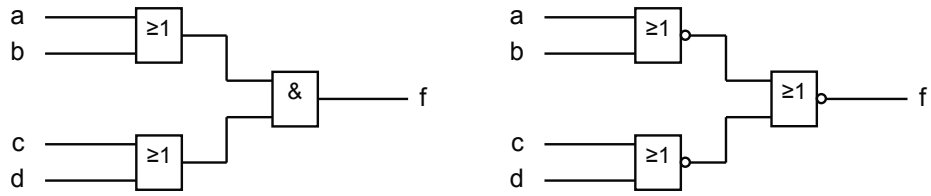
Dit is de functie van een EXOR. Deze schakeling werd in de jaren '70 van de vorige eeuw vaak gebruikt want het was te bouwen met één 7400 TTL-IC.

Uitwerking opgave 1.17.

Een AND-OR-constructie is eenvoudig naar een NOR-NOR-constructie om te zetten door één keer De Morgan toe te passen:

$$f = (a+b)\cdot(c+d) = \overline{\overline{a+b} + \overline{c+d}} \tag{A.8}$$

Het OR-AND-schema en het NOR-NOR-schema zijn hieronder weergegeven.



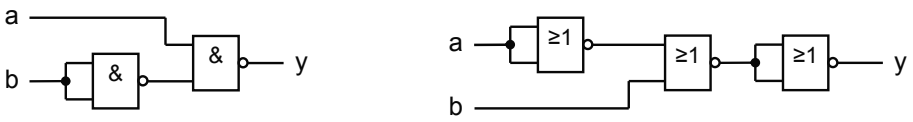
Figuur A.7: *Schema's van een OR-AND-schema en NOR-NOR-schema.*

Uitwerking opgave 1.18.

Snel is te zien dat de functie logisch 1 is als $a = 0$ en/of $b = 1$. In formulevorm $y = \overline{a} + b$. De formule is om te schrijven naar NANDs en NORs.

$$\begin{aligned} y &= \overline{a} + b = \overline{a \cdot \overline{b}} && \text{realisatie met NANDs} \\ y &= \overline{a} + b = \overline{\overline{a} + b} && \text{realisatie met NORs} \end{aligned} \tag{A.9}$$

De termen $\overline{a}$ en $\overline{b}$ kunnen gemaakt worden met een NAND of een NOR geschakeld als inverter. Zie de schema's hieronder.



Figuur A.8: *Realisatie van de functie met NAND- en NOR-poorten.*

Uitwerking opgave 1.19.

We moeten zoeken naar het kortste en langste pad (in tijd). Het kortste pad is snel te vinden. Via ingang z naar uitgang s wordt één AND-poort gepasseerd. Dus:

$$t_{P(min)}(\text{schakeling}) = t_{P(min)}(\text{AND}) = 2{,}4 \text{ ns} \tag{A.10}$$

Het langste pad is via ingang y. Daarbij wordt een NOT-poort, twee AND-poorten en een OR-poort gepasseerd. Dus

$$\begin{aligned} t_{P(max)}(\text{schakeling}) &= t_{P(max)}(\text{NOT}) + 2\cdot t_{P(min)}(\text{AND}) + t_{P(max)}(\text{OR}) \\ &= 4{,}3 + 2\cdot 6{,}9 + 7{,}2 \\ &= 25{,}3 \text{ ns} \end{aligned} \tag{A.11}$$

Uitwerking opgave 1.20.
De periodetijd van een signaal met de frequentie van 33,3 MHz is 30,03 ns (nanoseconden, 1 ns is 10^{-9} s. Dit is de frequentie van een 486DX-33 processor.

De periodetijd van een signaal met de frequentie van 3,4 GHz is 294,12 ps (picoseconden, 1 ps is 10^{-12} s. Dit is de frequentie van o.a. een Core-i7 processor.

De frequentie van een signaal met een periodetijd van 104,167 μs is 9,6 kHz (kilohertz). Dit is een standaard seinsnelheid van een RS-232 seriële verbinding.

Uitwerking opgave 2.1.
De getallen zijn om te zetten naar decimale equivalenten door alle 2-machten te noteren waarvan het corresponderende bit een 1 is en de 0-en te negeren:

$$\begin{aligned} 1111_2 &= 2^3 + 2^2 + 2^1 + 2^0 \\ &= 8 + 4 + 2 + 1 \\ &= 15 \end{aligned} \qquad \begin{aligned} 110011_2 &= 2^5 + 2^4 + 2^1 + 2^0 \\ &= 32 + 16 + 2 + 1 \\ &= 51 \end{aligned}$$

$$\begin{aligned} 01111111_2 &= 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 \\ &= 64 + 32 + 16 + 8 + 4 + 2 + 1 \\ &= 127 \end{aligned}$$

$$\begin{aligned} 1010.0101.1010.0101_2 &= 2^{15} + 2^{13} + 2^{10} + 2^8 + 2^7 + 2^5 + 2^2 + 2^0 \\ &= 32768 + 8192 + 1024 + 256 + 128 + 32 + 4 + 1 \\ &= 42405 \end{aligned}$$

Figuur A.9: *Decimale equivalent van enkele binaire getallen.*

Uitwerking opgave 2.2.
De getallen worden van decimaal naar binair omgezet door herhaald de delen door 2 en rest na deling te noteren. Dit gaat door tot het gehele deel 0 geworden is. Daarna kan het binaire equivalent uitgelezen worden door de resten na deling van onder naar boven te noteren.

$$\begin{array}{rrcrcl} 25 & \div 2 & = & 12 & \rightarrow & 1 \\ 12 & \div 2 & = & 6 & \rightarrow & 0 \\ 6 & \div 2 & = & 3 & \rightarrow & 0 \\ 3 & \div 2 & = & 1 & \rightarrow & 1 \\ 1 & \div 2 & = & 0 & \rightarrow & 1 \\ \underline{0} & & & & & \end{array} \qquad \begin{array}{rrcrcl} 10 & \div 2 & = & 5 & \rightarrow & 0 \\ 5 & \div 2 & = & 2 & \rightarrow & 1 \\ 2 & \div 2 & = & 1 & \rightarrow & 0 \\ 1 & \div 2 & = & 0 & \rightarrow & 1 \\ \underline{0} & & & & & \end{array}$$

Figuur A.10: *Binaire equivalent van enkele decimale getallen.*

Dus 25_{10} is gelijk aan 11001_2 en 10_{10} is gelijk aan 1010_2.

128	$\div 2$	=	64	→	0
64	$\div 2$	=	32	→	0
32	$\div 2$	=	16	→	0
16	$\div 2$	=	8	→	0
8	$\div 2$	=	4	→	0
4	$\div 2$	=	2	→	0
2	$\div 2$	=	1	→	0
1	$\div 2$	=	0	→	1
0					

27543	$\div 2$	=	13771	→	1
13771	$\div 2$	=	6885	→	1
6885	$\div 2$	=	3442	→	1
3442	$\div 2$	=	1721	→	0
1721	$\div 2$	=	860	→	1
860	$\div 2$	=	430	→	0
430	$\div 2$	=	215	→	0
215	$\div 2$	=	107	→	1
107	$\div 2$	=	53	→	1
53	$\div 2$	=	26	→	1
26	$\div 2$	=	13	→	0
13	$\div 2$	=	6	→	1
6	$\div 2$	=	3	→	0
3	$\div 2$	=	1	→	1
1	$\div 2$	=	0	→	1
0					

Figuur A.11: *Decimale equivalent van enkele binaire getallen.*

Dus 128_{10} is gelijk aan 10000000_2 en 27543_{10} is gelijk aan 110101110010111_2.

Uitwerking opgave 2.3.
Deze leggen we niet uit....

Uitwerking opgave 2.4.
Het uitrekenen geschiedt door de cijfers uit de hexadecimale getallen te schrijven als een 16-macht met in de exponent de positie van het cijfer (beginnend bij 0 voor het meest rechtse hexadecimale cijfer) vermenigvuldigd met het corresponderende cijfer uit het hexadecimale getal. Let er op dat de hexadecimale cijfers A t/m F omgezet moeten worden naar de getallen 10 t/m 15.

$$\begin{aligned} 3FF_{16} &= 3\cdot 16^2 + 15\cdot 16^1 + 15\cdot 16^0 \\ &= 3\cdot 256 + 15\cdot 16 + 15\cdot 1 \\ &= 768 + 240 + 25 \\ &= 1023 \end{aligned}$$

$$\begin{aligned} 4D52_{16} &= 4\cdot 16^3 + 13\cdot 16^2 + 5\cdot 16^1 + 2\cdot 16^0 \\ &= 4\cdot 4096 + 13\cdot 256 + 5\cdot 16 + 2\cdot 1 \\ &= 16384 + 3328 + 80 + 2 \\ &= 19794 \end{aligned}$$

$$\begin{aligned} CAFE_{16} &= 12\cdot 16^3 + 10\cdot 16^2 + 15\cdot 16^1 + 14\cdot 16^0 \\ &= 12\cdot 4096 + 10\cdot 256 + 15\cdot 16 + 14\cdot 1 \\ &= 51966 \end{aligned}$$

Figuur A.12: *Decimale equivalent van enkele hexadecimale getallen.*

Uitwerking opgave 2.5.

255	÷ 16	=	15	→	15 (F)		57005	÷ 16	=	3562	→	13 (D)
15	÷ 16	=	0	→	15 (F)		3562	÷ 16	=	222	→	10 (A)
0							222	÷ 16	=	13	→	14 (E)
							13	÷ 16	=	0	→	13 (D)
							0					

Figuur A.13: *Hexadecimake equivalent van enkele decimale getallen.*

Dus 255_{10} is gelijk aan FF_{16} en 57005_{10} is gelijk aan $DEAD_{16}$. Het hexadecimale equivalent van 32768_{10} is 8000_{16}. Dit is een macht van 2 (2^{15}).

Uitwerking opgave 2.6.

IPv6-adressen zijn 128 bits groot. Aangezien elk bit een 0 of een 1 kan zijn, is het aantal adressen 2^{128}. Dat zijn 340.282.366.920.938.463.463.374.607.431.768.211.455 verschillende adressen. Zie http://en.wikipedia.org/wiki/IPv6_address.

Leuk om te weten: het hele IPv4-adressenbereik kan 2^{96} keer in het IPv6-adressenbereik. Dat is 79.228.162.514.264.337.593.543.950.336 keer.

Uitwerking opgave 2.7.

Voor het gemak gaan we ervan uit dat de aarde een bol is (dat is het namelijk niet). Onze aarde heeft een diameter van 6.371 km, dat is 6.371.000 m. De oppervlakte van een bol is $O = 4\pi r^2$. Invullen en uitrekenen levert een opppervlakte van 510.064.471.909.788 m^2. Nu het aantal adressen delen door de oppervlakte, dat levert 667.135.990.959.828.999.515.555 IPv6-adressen per vierkante meter. We schrijven het even in 10-machten om de grootte te overzien: $6{,}67 \cdot 10^{23}$. Dat is waarschijnlijk wel genoeg voor de komende tijd. Op de website https://rednectar.net/2012/05/24/just-how-many-ipv6-addresses-are-there-really/ zijn nog meer interessante gegevens te vinden.

Uitwerking opgave 2.8.

Binaire breuken omzetten naar het decimale equivalent gaat op dezelfde wijze als het omzetten van gehele getallen, alleen zijn de machten nu negatief. Het gewicht van het eerste cijfer rechts van de komma is $2^{-1} = \frac{1}{2}$ en het tweede cijfer is $2^{-2} = \frac{1}{4}$ etc. Uiteraard worden alleen die machten genoteerd waarvoor het cijfer 1 in het getal staat.

$$
\begin{aligned}
0{,}1001_2 &= 2^{-1} + 2^{-4} \\
&= \frac{1}{2} + \frac{1}{16} \\
&= 0{,}5 + 0{,}0625 \\
&= 0{,}5625
\end{aligned}
\qquad
\begin{aligned}
0{,}11111111 &= 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-6} + 2^{-7} + 2^{-8} \\
&= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} + \frac{1}{256} \\
&= 0{,}5 + 0{,}25 + \cdots + 0{,}0078125 + 0{,}00390625 \\
&= 0{,}99609375
\end{aligned}
$$

Figuur A.14: *Decimale equivalent van enkele binaire fracties.*

Overigens het het slimmer om gebruik te maken van de gelijkheid $0{,}11111111_2 = 1_2 - 0{,}00000001_2$ zodat het antwoord berekend kan worden met $1 - 1^{-8}$ of $1 - \frac{1}{256}$.

Uitwerking opgave 2.9.

Het omzetten van een decimale breuk naar het binaire equivalent wordt gerealiseerd door de decimale breuk te vermenigvuldigen met 2, het gehele getal te noteren (0 of 1) en de procedure te herhalen met de fractie totdat het resultaat van een vermenigvuldiging 0,0 oplevert. Zie hieronder.

0,75	× 2	=	1,5	→	1	0,3	× 2	=	0,6	→	0
0,5	× 2	=	1,0	→	1	0,6	× 2	=	1,2	→	1
0,0						0,2	× 2	=	0,4	→	0
						0,4	× 2	=	0,8	→	0
						0,8	× 2	=	1,6	→	1
						0,6	...				

Figuur A.15: *Binaire equivalent van enkele decimale fracties.*

Het getal $0{,}75_{10}$ is gelijk aan $0{,}11_2$. Het getal $0{,}6_{10}$ leidt tot de repeterende binaire breuk $0{,}010011001\ldots_2$ wat inhoudt *dat 0,6 niet exact kan worden weergegeven in het binaire stelsel!* Dit geldt trouwens voor veel eindige decimale breuken. In de rechter berekening zijn de getallen $0{,}3_{10}$, $0{,}2_{10}$, $0{,}3_{10}$ en $0{,}8_{10}$ te zien, die zijn uiteraard ook niet exact weer te geven.

Op eenzelfde wijze levert de omzetting van $0{,}8_{10}$ het binaire getal $0{,}11001100\ldots_2$ op.

Uitwerking opgave 2.10.

Het getal 11 is zo basaal, dat schrijven we direct op: B_{16}.

De decimale breuk kan eerst omgezet worden naar een binaire breuk en dan omgezet worden naar een hexadecimaal equivalent. Maar het kan sneller door het direct uit te rekenen als hexadecimale breuk:

0,7	× 16	=	11,2	→	B
0,2	× 16	=	3,2	→	3
0,2					

Figuur A.16: *Hexadecimaal equivalent van een decimale fractie.*

Het getal $11{,}7_{10}$ is equivalent met $B{,}B3333\ldots_{16}$. Merk op dat het getal niet exact is weer te geven in het hexadecimale talstelsel.

Het getal $1F,3C_{16}$ naar decimaal omzetten gaat op de gebruikelijke manier.

$$\begin{aligned} 1F,3C_{16} &= 1\cdot16^1 + 15\cdot16^0 + 3\cdot16^{-1} + 12\cdot16^{-2} \\ &= 16 + 15 + \frac{3}{16} + \frac{12}{256} \\ &= 31 + 0{,}1875 + 0{,}046875 \\ &= 31{,}234375 \end{aligned}$$

Uitschrijven van een hexadecimaal getal als binair getal is eenvoudig weg alle hexadecimale cijfers vervangen door de bijbehorende 4-bits binaire code:

$$3F,EA_{16} = 0011.1111,1110.1010_2$$

Let op de plaats van de punten en de komma. Leidende en volgende nullen zouden kunnen worden weggelaten.

Bij het omzetten van het binaire getal $110110,110111_2$ moeten het gehele deel en de fractie eerst geschreven worden in veelvouden van vier bits, een hexadecimaal cijfer moet immers geschreven worden met precies vier bits.

We breiden beide delen uit: het gehele deel krijgt voorlopende nullen, de fractie krijgt volgende nullen: $00110110,11011100_2$. Het getal is nu op te splitsen in delen van precies vier bits: $0011.0110,11011100_2$ en is direct te schrijven als het hexadecimale getal $36,DC_{16}$.

Uitwerking opgave 2.11.

In het octale (achttallig) talstelsel worden de cijfers 0 t/m 7 gebruikt. Het omzetten van een octaal getal naar een decimaal getal geschiedt op de zelfde wijze als bij binaire en hexadecimale getallen.

$$\begin{aligned} 377_8 &= 3\cdot8^2 + 7\cdot8^2 + 7\cdot8^0 \\ &= 3\cdot64 + 7\cdot8 + 7\cdot1 \\ &= 192 + 56 + 7 \\ &= 255 \end{aligned} \qquad \begin{aligned} 127_8 &= 1\cdot8^2 + 2\cdot8^2 + 7\cdot8^0 \\ &= 1\cdot64 + 2\cdot8 + 7\cdot1 \\ &= 64 + 16 + 7 \\ &= 87 \end{aligned}$$

Octale getallen bestaan uit acht verschillende cijfers. Acht is een macht van 2 ($2^3 = 8$) en dat houdt in dat een octaal getal eenvoudig is om te zetten in een binair getal en dus ook in een hexadecimaal getal.

In de programmeertaal C begint een octaal getal met een 0, dus `0377` is 377_8 (255_{10}).

Het toegangsrechtensysteem bij bestandssystemen onder Linux kan worden aangepast en weergegeven d.m.v. octale getallen. Zie http://linuxcommand.org/lts0070.php. Onder Linux is het commando `od` (octal dump) beschikbaar om bestanden octaal (en hexadecimaal) weer te geven. Zie http://en.wikipedia.org/wiki/Od_(Unix).

In de datacommunicatiewereld wordt niet gesproken van een byte maar een *octet*.

Uitwerking opgave 2.12.
Het getal 41 in onbekende talstelsel is gelijk aan het kwadraat van 5 in dit stelsel. Dus $41_x = (5_x)^2$. Daaruit volgt

$$4 \cdot x^1 + 1 \cdot x^0 = (5 \cdot x^0) \cdot (5 \cdot x^0) \tag{A.12}$$

Nu het een en ander herschrijven en vervolgens uitrekenen:

$$\begin{aligned} 4x + 1 &= 25 \\ 4x &= 24 \\ x &= 6 \end{aligned} \tag{A.13}$$

Het getal 41 staat in het zestallig (hexaal?) talstelsel.

Uitwerking opgave 2.13.
Het marsiaanse getal 234 is geschreven in het x-tallig stelsel en dit getal is gelijk aan 123_{10}. Dan geldt $234_x = 123_{10}$. We schrijven het marsiaanse getal uit in machten van x en stellen dit gelijk aan het decimale getal 123.

$$\begin{aligned} 234_x &= 2 \cdot x^2 + 3 \cdot x^1 + 4 \cdot x^0 \\ &= 2 \cdot x^2 + 3 \cdot x + 4 \cdot 1 \\ &= 123 \end{aligned} \tag{A.14}$$

Deze vergelijking kan worden omgewerkt naar een tweedegraads vergelijking

$$2x^2 + 3x - 119 = 0 \tag{A.15}$$

Oplossen met de ABC-formule[1] levert twee antwoorden op: $x_1 = 7$ en $x_2 = -8{,}5$. Een grondtal moet een geheel getal groter dan 1[2] zijn waardoor x_2 afvalt. De marsiaanse getallen zijn in het 7-tallig stelsel geschreven.

Uitwerking opgave 2.14.
Het getal 653 moet worden omgeslagen in acht bits. De vraag is natuurlijk hoe je zo'n getal kan invoeren als het in acht bits moet worden opgeslagen, maar de C-compiler doet dat. De bewerking is

$$653 \bmod 2^8 = 141 \qquad \text{of} \qquad 141 \equiv 653 \pmod{2^8} \tag{A.16}$$

Onderstaande code is een implementatie in C (listing A.1) die continue tekens inleest totdat een niet-cijfer wordt ingevoerd. Met deze code is het goed mogelijk een getal in te voeren dat te groot is om in het resultaat opgeslagen te worden.

Uitwerking opgave 2.15.
Onderstaande uitdrukkingen geven het verband aan.

$$x \equiv M \pmod{2^n} \quad \rightarrow \quad x = k \cdot 2^n + M \quad \text{met } k \in \mathrm{N} \tag{A.17}$$

[1] Zie http://www.wiskundeonline.nl/lessen/kw_bewijs_abc_formule.htm
[2] Strikt genomen hoeft het grondtal geen geheel getal en/of positief te zijn. Je kan ook getallen weergeven met het grondtal $\sqrt{2}$ or π. Zie http://en.wikipedia.org/wiki/Non-integer_representation

```c
#include <stdio.h>              /* for getchar et al. */

int main(void) {

    unsigned char res;          /* 8-bit unsigned */
    int x;                      /* 16-bit signed */

    res = 0;                    /* clear intermediate result */
    x = getchar();              /* get a character */
    while (isdigit(x)) {        /* if it is a digit ... */
        res = res*10 + (x-'0'); /* mult. prev. by 10, add new digit */
        x = getchar();          /* get next character */
    }
    printf("%d\n", res);
    return 0;
}
```

Listing A.1: *Het inlezen van een integer in 'C'.*

Uitwerking opgave 2.16.
Het aantal bits dat nodig is om een 8-cijferig decimaal getal op te slaan, is eenvoudig te bepalen door de 2-log van 10^8 te nemen en het antwoord af te ronden naar het eerst hogere gehele getal (er bestaan immers alleen hele bits). Dus

$$n = \left\lceil \frac{8}{\log 2} \right\rceil = \lceil 26{,}575\cdots \rceil = 27 \tag{A.18}$$

Natuurlijk kunnen er met 27 bits meer dan 10^8 getallen gemaakt worden.

Uitwerking opgave 2.17.
Om het aantal bits dat nodig is om een specifiek decimaal getal op te slaan uit te rekenen moet de 2-log van het getal vermeerderd met 1 uitgerekend worden. Natuurlijk moet weer worden afgerond naar het eerst hogere gehele getal.

$$\begin{aligned}
n_{365} &= \left\lceil \frac{\log(365+1)}{\log 2} \right\rceil = \lceil 8{,}51\cdots \rceil = 9 \\
n_{1024} &= \left\lceil \frac{\log(1024+1)}{\log 2} \right\rceil = \lceil 10{,}001\cdots \rceil = 11 \\
n_{7987} &= \left\lceil \frac{\log(7987+1)}{\log 2} \right\rceil = \lceil 12{,}96\cdots \rceil = 13 \\
n_{176890} &= \left\lceil \frac{\log(176890+1)}{\log 2} \right\rceil = \lceil 17{,}43\cdots \rceil = 18
\end{aligned} \tag{A.19}$$

Op http://www.exploringbinary.com/number-of-bits-in-a-decimal-integer/ is een uitgebreid verhaal te vinden.

Uitwerking opgave 2.18.
Een twee-cijferig BCD-getal gebruikt 100 combinaties (00 t/m 99). Hiervoor zijn acht

bits nodig, vier bits per BCD-cijfer. Het aantal nuttige combinaties is $\frac{100}{256} \cdot 100\%$. Dat is ongeveer 39%.

Een drie-cijferig BCD-getal heeft $\frac{1000}{4096} \cdot 100\%$ codecombinaties, dat is ongeveer 24%.

Uitwerking opgave 2.19.

Eerst maar eens een waarheidstabel opstellen voor de omzetter. De ingangen staan als normaal binaire telcode gerangschikt, de uitgangen stellen de bijbehorende Gray-code voor. Daar kunnen de functies voor de Gray-bits worden bepaald.

Tabel A.5: *Waarheidstabel omzetting binaire telcode naar Gray-code.*

b_2	b_1	b_0	g_2	g_1	g_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

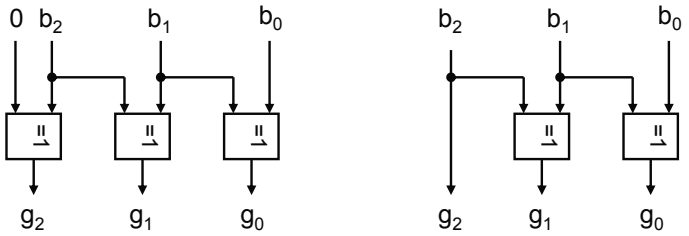
Na enig rekenwerk volgt:

$$\begin{aligned} g_0 &= b_1 \oplus b_0 \\ g_1 &= b_2 \oplus b_1 \\ g_2 &= 0 \oplus b_2 = b_2 \end{aligned} \tag{A.20}$$

Het mag duidelijk zijn dat de functies uit EXOR-poorten bestaan. De functie voor g_2 is als EXOR geschreven waarbij één ingang aan een logische 0 gekoppeld is. Uitbreiding naar meer bits is dus eenvoudig: voor bit g_i geldt:

$$g_i = b_{i+1} \oplus b_i \tag{A.21}$$

De vertragingstijd is slechts één poortvertraging, onafhankelijk van het aantal bits. Zie onderstaande figuur.



Figuur A.17: *Omzetter binaire telcode naar Gray-code.*

Uitwerking opgave 2.20.

Op vergelijkbare wijze als in opgave 2.19 kan ook deze opgave worden opgelost. De ingangen stellen nu de Gray-code voor, de uitgangen stellen de bijbehorende normale binaire code voor. Zie onderstaande tabel.

Tabel A.6: *Waarheidstabel omzetting Gray-code naar binaire telcode.*

g_2	g_1	g_0	b_2	b_1	b_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	1
1	0	1	1	1	0
1	1	0	1	0	0
1	1	1	1	0	1

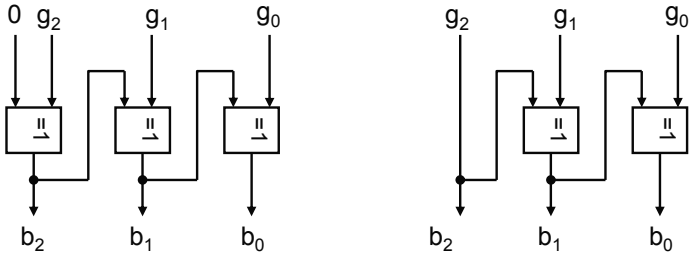
Na enig rekenwerk volgt:

$$\begin{aligned} b_0 &= g_2 \oplus g_1 \oplus g_0 &&= b_1 \oplus g_0 \\ b_1 &= g_2 \oplus g_1 &&= b_2 \oplus g_1 \\ b_2 &= g_2 \oplus 0 &&= 0 \oplus g_2 \end{aligned} \tag{A.22}$$

Het mag duidelijk zijn dat de functies uit EXOR-poorten bestaan. De functie voor g_2 is als EXOR geschreven waarbij één ingang aan een logische 0 gekoppeld is. Uitbreiding naar meer bits is dus eenvoudig: voor bit g_i geldt:

$$b_i = b_{i+1} \oplus g_i \tag{A.23}$$

Let er op dat dit een cascade- of serieschakeling van EXOR-poorten oplevert omdat bij de functie voor b_i de waarde van b_{i+1} nodig is, die moet dan wel bekend zijn. Bij een groot aantal bits levert dit een aanzienlijke vertragingstijd op. Zie onderstaande figuur.



Figuur A.18: *Omzetter van Gray-code naar binaire code.*

Uitwerking opgave 2.21.

Uit de ASCII-tabel blijkt dat de hoofdletters oplopend worden gecodeerd vanaf code 41_{16}. De 'A' is 41_{16}, 'B' is 42_{16} etc. Voor de kleine letters geldt dat deze ook oplopend gecodeerd zijn maar dan vanaf 61_{16}, dus 'a' is 61_{16}, 'b' is 62_{16} etc. Het verschil tussen een hoofdletters en een bijbehorende kleine letter is 20_{16}.

Om een hoofdletter te veranderen in een kleine letter moet bij de code van de hoofdletter 20_{16} worden opgeteld, bij het veranderen van een kleine letter in een hoofdletter moet er 20_{16} worden afgetrokken.

Nu is 20_{16} hetzelfde als 100000_2 of 2^5. Je kan dus ook het 6^e bit van de ASCII-code aanpassen: een 0 levert een hoofdletter op, een 1 levert een kleine letter op. Dat alles mag natuurlijk alleen in het ASCII-bereik van de letters.

In de programmeertaal C kan je eenvoudig bewerkingen doen op *characters* (tekens). Zie het voorbeeld hieronder.

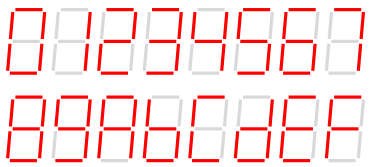
```c
#include <stdio.h>

int main(void) {

    char A, a, B, b, Negen, negen;

    A = 'A'; b = 'b'; negen = '9';

    a = A + 0x20;           /* a = 'a' */
    B = b - 0x20;           /* B = 'B' */
    Negen = negen + 0x20;   /* Negen = 'Y' */

    printf("%c -- %c,%c -- %c,%c -- %c\n", A, a, B, b, Negen, negen);

    return 0;
}
```

Listing A.2: *Het omzetten van hoofdletters naar kleine letters in 'C'.*

In C zijn de functies `toupper()` en `tolower()` beschikbaar voor de conversie tussen kleine letters en hoofdletters.

Uitwerking opgave 2.22.

Hieronder is een afbeelding te zien van alle 16 hexadecimale cijfers. Merk op dat de 'A'



Figuur A.19: *Alle hexadecimale cijfers op een 7-segment display.*

als hoofdletter is geschreven, het is niet mogelijk om een kleine 'a' te maken. De 'b' is als kleine letter te zien, want een hoofdletter 'B' is niet te onderscheiden van een '8'. De 'C' zou ook als kleine letter kunnen worden uitgevoerd, maar meestal wordt gekozen voor de hoofdletter. De 'd' is weer als kleine letter geschreven, een hoofdletter 'D' is niet te onderscheiden van een '0'. De 'E' wordt als hoofdletter geschreven, een kleine letter 'e' kan wel maar is te 'hoog'. De 'F' kan alleen als hoofdletter geschreven worden. Merk trouwens het subtiele verschil tussen de '6' en de 'b' op. (De figuur is realiseert met een aangepaste versie op http://www.texample.net/tikz/examples/segment-display/.)

Uitwerking opgave 3.1.

Gegeven dat $a = 1$, $b = 0$ en $c = 0$. De waarde van d is niet gegeven.

$$
\begin{aligned}
s &= (a+b)\cdot(\overline{b}+c) \\
&= (1+0)\cdot(\overline{0}+0) \\
&= (1)\cdot(1+0) \\
&= 1\cdot 1 \\
&= 1
\end{aligned}
\qquad
\begin{aligned}
s &= \overline{(\overline{a+b})\cdot(\overline{c+d})} \\
&= \overline{(\overline{1+0})\cdot(\overline{0+d})} \\
&= \overline{(\overline{1})\cdot(\overline{d})} \\
&= \overline{0\cdot\overline{d}} \\
&= \overline{0} \\
&= 1
\end{aligned}
\qquad
\begin{aligned}
s &= \overline{b}\cdot(c+d\cdot(c+a)) \\
&= \overline{0}\cdot(0+d\cdot(0+1)) \\
&= 1\cdot(0+d\cdot(1)) \\
&= 1\cdot(0+d\cdot 1) \\
&= 1\cdot d \\
&= d
\end{aligned}
\tag{A.24}
$$

Natuurlijk kunnen sommige stappen samengenomen worden. Zo kan $(0+1)$ direct geschreven worden als 1.

Uitwerking opgave 3.2.
Onderstaande is het bewijs voor de stelling $a + a\cdot b = a$.

$$
\begin{aligned}
a + a\cdot b &= a\cdot 1 + a\cdot b \\
&= a\cdot(1+b) \\
&= a\cdot 1 \\
&= a
\end{aligned}
\tag{A.25}
$$

Uitwerking opgave 3.3.
Laten we de functie even S noemen:

$$
\begin{aligned}
S &= (a+b)\cdot(\overline{a}+c)\cdot(b+c) && \text{de functie} \\
&= (a+b)\cdot(\overline{a}+c)\cdot(0+b+c) && \text{middels } x = 0+x \\
&= (a+b)\cdot(\overline{a}+c)\cdot(\overline{a}\cdot a+b+c) && \text{middels } \overline{x}\cdot x = 0 \\
&= (a+b)\cdot(\overline{a}+c)\cdot(\overline{a}+b+c)\cdot(a+b+c) && \text{middels } \overline{x}\cdot x + y = (\overline{x}+y)\cdot(x+y) \\
&= (a+b)\cdot(a+b+c)\cdot(\overline{a}+c)\cdot(\overline{a}+b+c) && \text{termen herschikken} \\
&= (a+b)\cdot(\overline{a}+c) && \text{middels } (x)\cdot(x+y) = (x)
\end{aligned}
\tag{A.26}
$$

Uitwerking opgave 3.4.
De functie omwerken naar een som van mintermen gaat als volgt:

$$
\begin{aligned}
s_{x,y,z} &= x\cdot(y+z\cdot\overline{y})+z\cdot y \\
&= x\cdot y + x\cdot\overline{y}\cdot z + y\cdot z \\
&= x\cdot y\cdot(\overline{z}+z) + x\cdot\overline{y}\cdot z + (\overline{x}+x)\cdot y\cdot z \\
&= x\cdot y\cdot\overline{z} + x\cdot y\cdot z + x\cdot\overline{y}\cdot z + \overline{x}\cdot y\cdot z + x\cdot y\cdot z \\
&= \overline{x}\cdot y\cdot z + x\cdot\overline{y}\cdot z + x\cdot y\cdot\overline{z} + x\cdot y\cdot z
\end{aligned}
\tag{A.27}
$$

In deze functie is x de meest significante en z de minst significante variabele. De som van mintermen en de waarheidstabel zijn als volgt (waarheidstabel en functie):

Tabel A.7: *Waarheidstabel.*

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

$$\begin{aligned} s_{x,y,z} &= \overline{x}\cdot y\cdot z + x\cdot \overline{y}\cdot z + x\cdot y\cdot \overline{z} + x\cdot y\cdot z \\ &= m_3 + m_5 + m_6 + m_7 \\ &= \sum m(3,5,6,7) \end{aligned} \tag{A.28}$$

Uitwerking opgave 3.5.
De functie moet in mintermen uitgeschreven worden:

$$\begin{aligned} s_{x,y,z} &= m_1 + m_3 + m_4 + m_7 \\ &= \overline{x}\cdot\overline{y}\cdot z + \overline{x}\cdot y\cdot z + x\cdot\overline{y}\cdot\overline{z} + x\cdot y\cdot z \end{aligned} \tag{A.29}$$

Uitwerking opgave 3.6.
De functie moet in maxtermen worden uitgeschreven:

$$\begin{aligned} s_{x,y,z} &= M_2\cdot M_6\cdot M_7 \\ &= (x+\overline{y}+z)\cdot(\overline{x}+\overline{y}+z)\cdot(\overline{x}+\overline{y}+\overline{z}) \end{aligned} \tag{A.30}$$

Uitwerking opgave 3.7.
We kunnen de som-van-producten-vorm omzetten in een product-van-sommen-vorm door de nullen uit de functie $s_{x,y,z}$ te nemen:

$$s_{x,y,z} = \sum m(1,2,3,6) = \prod M(0,4,5,7) \tag{A.31}$$

zodat de product-van-sommen-vorm als volgt wordt:

$$s_{x,y,z} = \prod M(0,4,5,7) = (x+y+z)\cdot(\overline{x}+y+z)\cdot(\overline{x}+y+\overline{z})\cdot(\overline{x}+\overline{y}+\overline{z}) \tag{A.32}$$

Uitwerking opgave 3.8.
We kunnen de product-van-sommen-vorm omzetten in een som-van-producten-vorm door de enen uit de functie $s_{x,y,z}$ te nemen:

$$s_{x,y,z} = \prod M(0,1,3,5) = \sum m(2,4,6,7) \tag{A.33}$$

zodat de product-van-sommen-vorm als volgt wordt:

$$s_{x,y,z} = \sum m(2,4,6,7) = \overline{x}\cdot y\cdot\overline{z} + x\cdot\overline{y}\cdot\overline{z} + x\cdot y\cdot\overline{z} + x\cdot y\cdot z \tag{A.34}$$

Uitwerking opgave 3.9.

$$
\begin{aligned}
S_1 &= \sum m(1,3,5,6,7) = \overline{x}\cdot\overline{y}\cdot z + \overline{x}\cdot y\cdot z + x\cdot\overline{y}\cdot z + x\cdot y\cdot\overline{z} + x\cdot y\cdot z \\
S_2 &= \sum m(0,1,2,4,7) = \overline{x}\cdot\overline{y}\cdot\overline{z} + \overline{x}\cdot\overline{y}\cdot z + \overline{x}\cdot y\cdot\overline{z} + x\cdot\overline{y}\cdot\overline{z} + x\cdot y\cdot z \\
S_3 &= \sum m(7) = x\cdot y\cdot z \\
S_4 &= \sum m(1,2,3,5,6,7) = \overline{x}\cdot\overline{y}\cdot z + \overline{x}\cdot y\cdot\overline{z} + \overline{x}\cdot y\cdot z + x\cdot\overline{y}\cdot z + x\cdot y\cdot\overline{z} + x\cdot y\cdot z \\
S_5 &= \sum m(1,2,3,4,5,6,7) = \overline{x}\cdot\overline{y}\cdot z + \overline{x}\cdot y\cdot\overline{z} + \overline{x}\cdot y\cdot z + x\cdot\overline{y}\cdot\overline{z} + x\cdot\overline{y}\cdot z + x\cdot y\cdot\overline{z} + x\cdot y\cdot z \\
S_6 &= \sum m(0,2,4,6) = \overline{x}\cdot\overline{y}\cdot\overline{z} + \overline{x}\cdot y\cdot\overline{z} + x\cdot\overline{y}\cdot\overline{z} + x\cdot y\cdot\overline{z} \\
S_7 &= \sum m(0,2,4) + d(1,3,6)
\end{aligned}
\tag{A.35}
$$

De functie voor S_7 is alleen in de verkorte notatie (of somnotatie) te schrijven.

Uitwerking opgave 3.10.
Om de product-van-maxterm-vorm te krijgen moeten we de nullen uit de functies nemen:

$$
\begin{aligned}
S_1 &= \prod M(0,2,4) = (x+y+z)\cdot(x+\overline{y}+z)\cdot(\overline{x}+y+z) \\
S_2 &= \prod M(3,5,6) = (x+\overline{y}+\overline{z})\cdot(\overline{x}+y+\overline{z})\cdot(\overline{x}+\overline{y}+z) \\
S_3 &= \prod M(0,1,2,3,4,5,6) = (x+y+z)\cdot(x+y+\overline{z})\cdot(x+\overline{y}+z)\cdot(x+\overline{y}+\overline{z})\cdot \\
&\qquad (\overline{x}+y+z)\cdot(\overline{x}+y+\overline{z})\cdot(\overline{x}+\overline{y}+z) \\
S_4 &= \prod M(0,4) = (x+y+z)\cdot(\overline{x}+y+z) \\
S_5 &= \prod M(0) = (x+y+z) \\
S_6 &= \prod M(1,3,5,7) = (x+y+\overline{z})\cdot(x+\overline{y}+\overline{z})\cdot(\overline{x}+y+\overline{z})\cdot(\overline{x}+\overline{y}+\overline{z}) \\
S_7 &= \prod M(1,3,5,6,7)\cdot D(1,3,6)
\end{aligned}
\tag{A.36}
$$

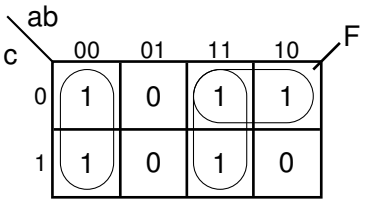
De functie voor S_7 is alleen in de verkorte notatie (of productnotatie) te schrijven.

Uitwerking opgave 3.11.
Hieronder is de waarheidstabel gegeven, zie tabel A.8

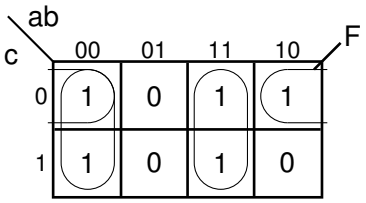
Uitwerking opgave 4.1.
Hieronder zijn de Karnaughdiagrammen met omrandingen te zien. Zie figuur A.20. Bij twee diagrammen zijn meerdere oplossingen mogelijk die leiden tot een minimale functie. Eén diagram kan niet vereenvoudigd worden.

Tabel A.8: *Waarheidstabel.*

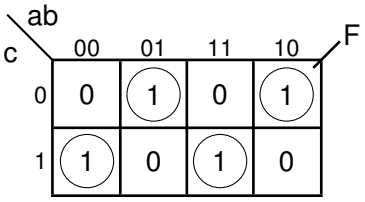
x	y	z	s	
0	0	0	-	don't care
0	0	1	1	$x = 0$ en $z = 1$
0	1	0	0	
0	1	1	1	$x = 0$ en $z = 1$
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	-	don't care



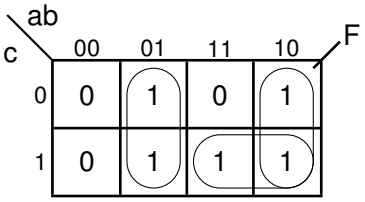
$F = \overline{a} \cdot \overline{b} + a \cdot b + a \cdot \overline{c}$



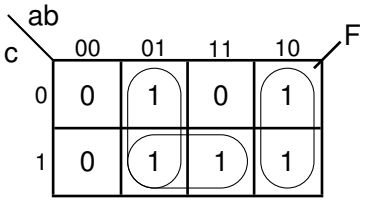
$F = \overline{a} \cdot \overline{b} + a \cdot b + \overline{b} \cdot \overline{c}$



$F = \overline{a} \cdot \overline{b} \cdot c + \overline{a} \cdot b \cdot \overline{c} + a \cdot \overline{b} \cdot \overline{c} + a \cdot b \cdot c$



$F = \overline{a} \cdot b + a \cdot \overline{b} + a \cdot c$



$F = \overline{a} \cdot b + a \cdot \overline{b} + b \cdot c$

Figuur A.20: *Uitwerkingen van de Karnaughdiagrammen.*

Uitwerking opgave 4.2.

We vullen de functie $f_{x,y,z} = \sum m(0,1,2,6)$ in een Karnaughdiagram in. Op de plaatsen van de mintermen 0, 1, 2 en 6 wordt een 1 ingevuld, de overige plaasten zijn logisch 0. Zie het Karnaughdiagram linksonder. In dit diagram zijn ook de plaatsen van de mintermen aangegeven met een indexcijfer. Het Karnaughdiagram is rechtsonder nog een keer getekend maar nu zijn de omrandingen zichtbaar. De functie kan vereenvoudigd worden tot $f_{x,y,z} = \overline{x} \cdot \overline{y} + y \cdot \overline{z}$.

Figuur A.21: *Karnaughdiagrammen.*

De tweede functie is $f_{a,b,c} = \sum m(1,2,5,6,7)$ en wordt ingevuld in een Karnaughdiagram, zie linksonder. Rechtsonder is het diagram omrand. De vereenvoudigde functie is $f_{a,b,c} = \overline{b}\cdot c + b\cdot\overline{c} + a\cdot c$. In deze uitwerking is gekozen om minterm 7 samen te nemen met minterm 5, maar is ook mogelijk om minterm 7 samen te nemen met minterm 6. In dat geval wordt de term $a\cdot c$ vervangen door $a\cdot b$.

Figuur A.22: *Karnaughdiagrammen.*

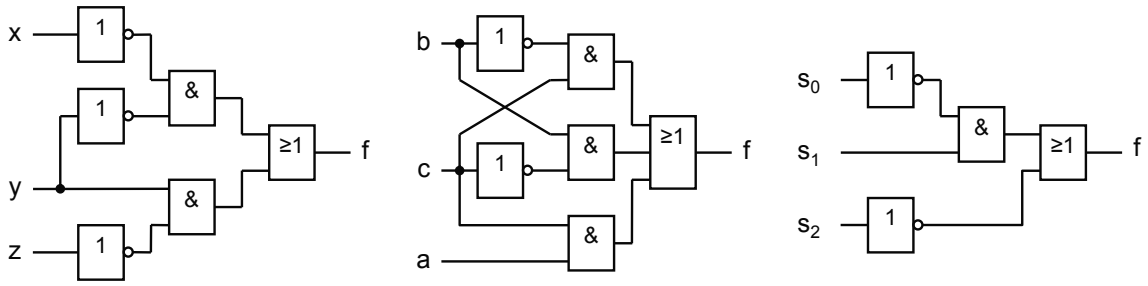
We vullen de functie $f_{s_2,s_1,s_0} = \sum m(0,3,6) + d(1,2)$ in een Karnaughdiagram in. Merk op dat twee functiewaarden als don't care zijn gespecificeerd. We vullen hiervoor een minteken ("dash") in. Zie het Karnaughdiagram linksonder. Bij het uitwerken van het Karnaugh-diagram worden de don't cares meegenomen als logische enen. Hierdoor wordt de functie het meest eenvoudig. De functie is $f_{s_2,s_1,s_0} = \overline{s_2} + s_1\cdot\overline{s_0}$.

Figuur A.23: *Karnaughdiagrammen.*

Hieronder zijn de schema's gegeven van de functies in AND-, OR- en NOT-poorten. De schema's zijn direct uit de functies getekend. Zie figuur A.24.

Uitwerking opgave 4.3.

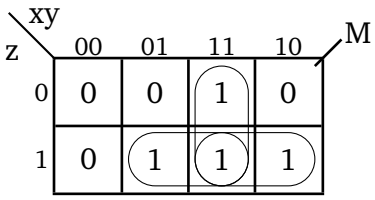
Een *majority gate* is een schakeling waarbij de uitgang logisch 1 is als de meerderheid van de ingangen logisch 1 is, anders is de uitgang logisch 0. Dat houdt in dat een majority gate altijd een *oneven* aantal ingangen heeft, in dit geval drie. De uitgang wordt logisch 1 als er twee of drie ingangen logisch 1 zijn, anders wordt de uitgang logisch 0. We noemen de ingangen x, y en z en de uitgang M. De tabel is als volgt te construeren waarna het Karnaughdiagram getekend kan worden en de functie kan worden afgeleid.



Figuur A.24: *Schema's van de functies.*

Tabel A.9: *Waarheidstabel majority gate.*

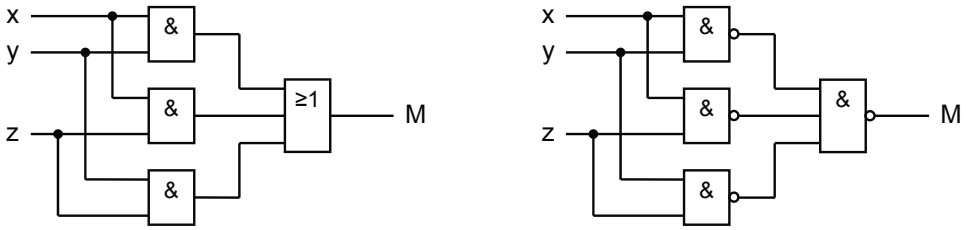
x	y	z	M	
0	0	0	0	0 enen
0	0	1	0	1 een
0	1	0	0	1 een
0	1	1	1	2 enen
1	0	0	0	1 een
1	0	1	1	2 enen
1	1	0	1	2 enen
1	1	1	1	3 enen



$$M_{x,y,z} = x \cdot y + x \cdot z + y \cdot z$$

Figuur A.25: *Karnaughdiagram en functie majority gate.*

De functie is ook wel te verklaren: de functie moet een logische 1 geven als twee uitgangen logisch 1 zijn. Elk van de producttermen wordt logisch 1 als de twee bijbehorende variabelen logisch 1 zijn. Als drie variabelen logisch 1 zijn, zijn alle producttermen logisch 1. In alle andere gevallen is de uitgang logisch 0. Een geoefende lezer ziet in deze functie de *carry out*-functie van de full adder. Hieronder zijn twee schema's weergegeven. Links is het schema gebouwd in de AND-OR-vorm, rechts het schema in de NAND-NAND-vorm. De rechter variant is eenvoudig te vinden door één keer De Morgan toe te passen op de bovenstaande functie.

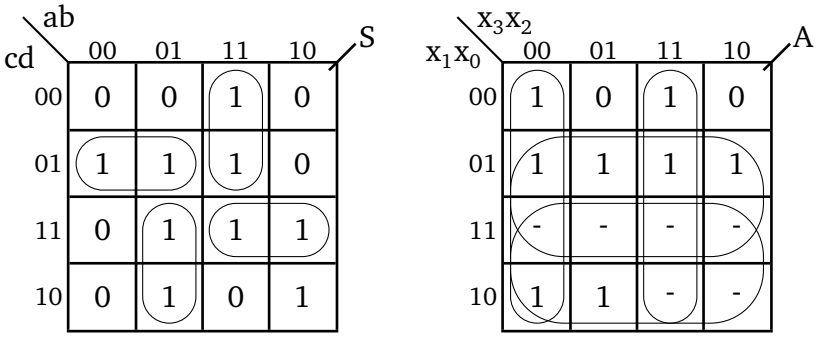


Figuur A.26: *Twee schema's voor een 3-ingangen majority gate.*

Uitwerking opgave 4.4.

Hieronder zijn de Karnaughdiagrammen nogmaals getekend, maar nu met omrandingen.

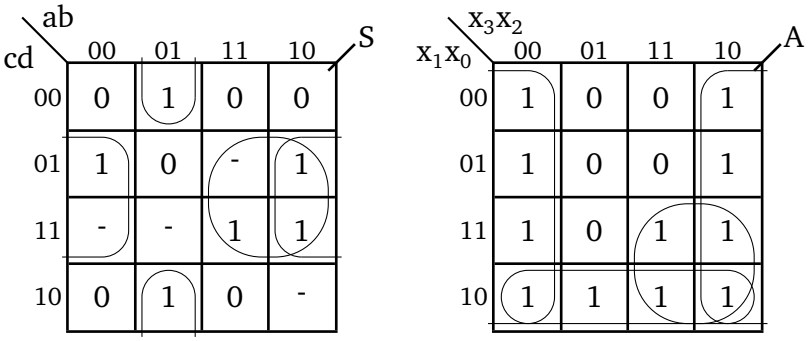
De functie voor het linker Karnaughdiagram is $S = \overline{a}\,\overline{c}d + acd + ab\overline{c} + \overline{a}bc$. Het is trouwens



Figuur A.27: *Karnaughdiagrammen.*

verleidelijk om de vier enen in het midden van het diagram samen te nemen tot de term bd. Maar zoals te zien is, worden deze enen afgedekt door de overige vier omrandingen.

De functie voor het rechter Karnaughdiagram is $A = x_0 + x_1 + \overline{x_3}\,\overline{x_2} + x_3x_2$. Hier zijn twee omrandingen van acht mintermen te zien. Die leveren per omranding een term van slechts één variabele op.



Figuur A.28: *Karnaughdiagrammen.*

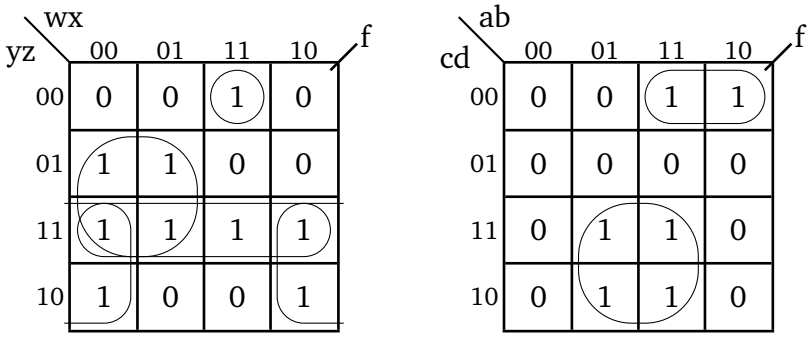
De functie voor het linker Karnaughdiagram is $S = ad + \overline{b}d + \overline{a}b\overline{d}$. Merk op dat de variabele c in deze functie niet voorkomt.

De functie van het rechter Karnaughdiagram is $A = \overline{x_2} + x_3x_1 + x_1\overline{x_0}$.

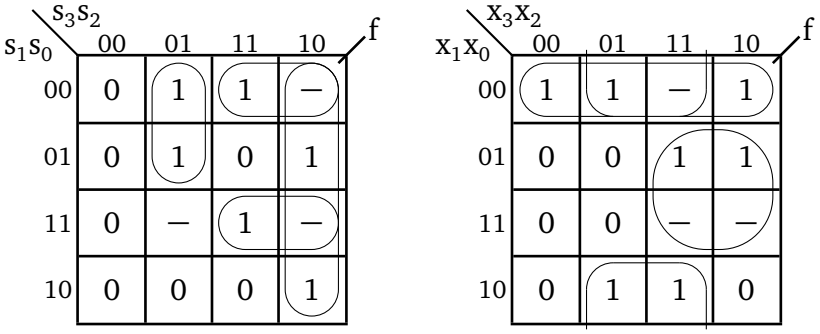
Uitwerking opgave 4.5.

De functie voor het linker Karnaughdiagram is $f_{w,x,y,z} = \overline{w}\cdot z + \overline{x}\cdot y + y\cdot z + w\cdot x\cdot\overline{y}\cdot\overline{z}$. De functie van het rechter Karnaughdiagram is $f_{a,b,c,d} = b\cdot c + a\cdot\overline{c}\cdot\overline{d}$.

Voor wat beteft het linker Karnaughdiagram zijn er meerdere functies mogelijk. Een mogelijke functie is $f_{s_3,s_2,s_1,s_0} = s_3\cdot\overline{s_2} + \overline{s_3}\cdot s_2\cdot\overline{s_0} + s_3\cdot\overline{s_1}\cdot\overline{s_0} + s_3\cdot s_1\cdot s_0$. De functie f_{x_3,x_2,x_1,x_0} is in productnotatie gegeven. Dat zijn de nullen in een waarheidstabel of Karnaughdiagram. De don't cares moeten gewoon worden ingevuld. De functie van het rechter Karnaughdiagram is $f_{x_3,x_2,x_1,x_0} = \overline{x_1}\cdot\overline{x_0} + x_2\cdot\overline{x_0} + x_3\cdot x_1$. I.p.v. productterm $x_3\cdot x_1$ kan ook productterm $x_3\cdot\overline{x_1}$ genomen worden.



Figuur A.29: *Karnaughdiagrammen.*



Figuur A.30: *Karnaughdiagrammen.*

Uitwerking opgave 4.6.

De combinaties van de functie $f_{w,x,y,z} = \sum m(1,2,3,5,7,10,11,12,15)$ zijn in tabel A.10 te zien. Uit kolom I volgt direct dat minterm 12 niet met andere te combineren is. Dit is dus een priemimplicant.

Tabel A.10: *Uitwerking van de tabellen.*

Kolom I.

	w	x	y	z	m.t.	
Groep 1	0	0	0	1	1	✓
	0	0	1	0	2	✓
Groep 2	0	0	1	1	3	✓
	0	1	0	1	5	✓
	1	0	1	0	10	✓
	1	1	0	0	12	*A*
Groep 3	0	1	1	1	7	✓
	1	0	1	1	11	✓
Groep 4	1	1	1	1	15	✓

Kolom II.

w	x	y	z	m.t.	
0	0	–	1	1,2	✓
0	–	0	1	1,5	✓
0	0	1	–	2,3	✓
–	0	1	0	2,10	✓
0	–	1	1	3,7	✓
–	0	1	1	3,11	✓
0	1	–	1	5,7	✓
1	0	1	–	10,11	✓
–	1	1	1	7,15	✓
1	–	1	1	14,15	✓

Kolom III.

w	x	y	z	m.t.	
0	–	–	1	1,3,5,7	*B*
0	–	–	1	1,5,3,7	
–	0	1	–	2,3,10,11	*C*
–	0	1	–	2,10,3,11	
–	–	1	1	3,7,11,15	*D*
–	–	1	1	3,11,7,15	

We hebben nu de volgende priemimplicanten gevonden:

$$1110 = A \qquad -010 = C$$
$$0--1 = B \qquad --11 = D$$

We zetten nu de dekkingstabel op voor de priemimplicanten, zie tabel A.11. We geven met een omcirkelde x aan of de priemimplicant essentieel is. Een priemimplicant is essentieel als er in een kolom van een minterm maar één x voorkomt. Deze minterm is namelijk alleen te combineren met de overige mintermen van de priemimplicant.

Tabel A.11: *Dekkingstabel van de priemplicanten.*

	m_1	m_2	m_3	m_5	m_7	m_{10}	m_{11}	m_{12}	m_{15}
A								(x)	
B	(x)		x	(x)	x				
C		(x)	x			(x)	x		
D			x		x		x		(x)

Uit de tabel volgt dat alle priemimplicanten essentieel zijn. We kunnen nog de Petrick-functie opstellen:

$$\begin{aligned} P_{f_{w,x,y,z}} &= B\cdot C\cdot(B+C+D)\cdot B\cdot(B+D)\cdot C\cdot(C+D)\cdot A\cdot D \\ &= B\cdot C\cdot B\cdot C\cdot A\cdot D \\ &= A\cdot B\cdot C\cdot D \end{aligned} \tag{A.37}$$

Zoals te zien is, zijn alle priemimplicanten nodig voor de dekking van de functie. Dus:

$$f_{w,x,y,z} = w\cdot x\cdot \overline{y}\cdot \overline{z} + \overline{w}\cdot z + \overline{x}\cdot y + y\cdot z \tag{A.38}$$

De combinaties van de functie $f_{a,b,c,d} = \sum m(6,7,8,12,14,15)$ zijn in tabel A.12 te zien.

Tabel A.12: *Uitwerking van de tabellen.*

Kolom I.

	a	b	c	d	m.t.	
Groep 1	1	0	0	0	8	✓
	0	1	1	0	6	✓
Groep 2	1	1	0	0	12	✓
	0	1	1	1	7	✓
Groep 3	1	1	1	0	14	✓
	1	1	1	1	15	✓
Groep 4						

Kolom II.

a	b	c	d	m.t.	
1	–	0	0	8,12	A
0	1	1	–	6,7	✓
–	1	1	0	6,14	✓
1	1	–	0	12,14	B
–	1	1	1	7,15	✓
1	1	1	–	14,15	✓

Kolom III.

a	b	c	d	m.t.	
–	1	1	–	6,7,14,15	C
–	1	1	–	6,14,7,15	

We hebben nu de volgende priemtimplicanten gevonden:

$$1-00 = A \qquad 11-0 = B \qquad -11- = C$$

Tabel A.13: *Dekkingstabel van de priemplicanten.*

	m_6	m_7	m_7	m_{12}	m_{14}	m_{15}
A			Ⓧ	x		
B				x	x	
C	Ⓧ	Ⓧ			x	Ⓧ

We zetten nu de dekkingstabel op voor de priemimplicanten, zie tabel A.13. We geven met een omcirkelde x aan of de priemimplicant essentieel is. Een priemimplicant is essentieel als er in een kolom van een minterm maar één x voorkomt. Deze minterm is namelijk alleen te combineren met de overige mintermen van de priemimplicant.

Uit de tabel volgt dat de priemimplicanten A en C essentieel zijn. Te zien is dat de mintermen gedekt door priemimplicant B gedekt worden door A en C. We kunnen nog de Petrick-functie opstellen:

$$\begin{aligned} P_{f_{a,b,c,d}} &= C \cdot C \cdot A \cdot (A+B) \cdot (B+C) \cdot C \\ &= C \cdot C \cdot A \cdot C \\ &= A \cdot C \end{aligned} \tag{A.39}$$

Voor de functie vinden we dus:

$$f_{a,b,c,d} = a \cdot \bar{c} \cdot \bar{d} + b \cdot c \tag{A.40}$$

De combinaties van de functie $f_{s_3,s_2,s_1,s_0} = \sum m(4,5,9,10,12) + d(7,8,11,15)$ zijn in tabel A.14 te zien.

Tabel A.14: *Uitwerking van de tabellen.*

Kolom I.

	s_3	s_2	s_1	s_0	m.t.	
Groep 1	0	1	0	0	4	✓
	1	0	0	0	8	✓
Groep 2	0	1	0	1	5	✓
	1	0	0	1	9	✓
	1	0	1	0	10	✓
	1	1	0	0	12	✓
Groep 3	0	1	1	1	7	✓
	1	1	0	1	11	✓
Groep 4	1	1	1	1	15	✓

Kolom II.

s_3	s_2	s_1	s_0	m.t.	
0	1	0	–	4,5	A
–	1	0	0	4,12	B
1	0	0	–	8,9	✓
1	0	–	0	8,10	✓
1	–	0	0	8,12	C
0	1	–	1	5,7	D
1	0	–	1	9,11	✓
1	0	1	–	10,11	✓
–	1	1	1	7,15	E
1	–	1	1	11,15	F

Kolom III.

s_3	s_2	s_1	s_0	m.t.	
1	0	–	–	8,9,10,11	G
1	0	–	–	8,10,9,11	

We hebben nu de volgende priemimplicanten gevonden:

$$\begin{aligned} 010- &= A & -111 &= E \\ -100 &= B & 1-11 &= F \\ 1-00 &= C & 10-- &= G \\ 01-1 &= D \end{aligned}$$

We stellen een dekkingstabel op van alle mintermen gecombineerd met de priemimplicanten maar laten de mintermen die als don't care te boek staan weg. Zie tabel A.15.

Tabel A.15: *Dekkingstabel van de priemplicanten.*

	m_4	m_5	m_9	m_{10}	m_{12}
A	x	x			
B	x				x
C					x
D		x			
E					
F					
G			ⓧ	ⓧ	

Te zien is dat priemimplicanten E en F geen mintermen dekken; alle bijhorende mintermen zijn don't care. We kunnen E en F dus direct schrappen.

Uit de tabel volgt verder dat priemimplicant G essentieel is We kunnen nu de Petrick-functie opstellen:

$$\begin{aligned} P_{f_{s_3,s_2,s_1,s_0}} &= (A+B)\cdot(A+D)\cdot G\cdot G\cdot(B+C) \\ &= A\cdot B\cdot G + A\cdot C\cdot G + B\cdot D\cdot G + B\cdot C\cdot D\cdot G \end{aligned} \tag{A.41}$$

De laatste term valt af want die bestaat uit vier priemimplicanten. We kunnen nu kiezen tussen de eerste drie.

$$\begin{aligned} f_{s_3,s_2,s_1,s_0} &= A+B+G = \overline{s_3}\cdot s_2\cdot\overline{s_1} + s_2\cdot\overline{s_1}\cdot\overline{s_0} + s_3\cdot\overline{s_2} \\ f_{s_3,s_2,s_1,s_0} &= A+C+G = \overline{s_3}\cdot s_2\cdot\overline{s_1} + s_3\cdot\overline{s_1}\cdot\overline{s_0} + s_3\cdot\overline{s_2} \\ f_{s_3,s_2,s_1,s_0} &= B+D+G = s_2\cdot\overline{s_1}\cdot\overline{s_0} + \overline{s_3}\cdot s_2\cdot s_1 + s_3\cdot\overline{s_2} \end{aligned} \tag{A.42}$$

Elk van de drie functies voldoet aan de minimale functie.

De functie $f_{x_3,x_2,x_1,x_0} = \prod M(1,2,3,5,7,10) + D(11,12,15)$ staat in POS-vorm en schrijven we eerst om naar SOP-vorm: $f_{x_3,x_2,x_1,x_0} = \sum m(0,4,6,8,9,13,14) + d(11,12,15)$. De combinaties zijn te zien in tabel A.16.

We stellen een dekkingstabel op van alle mintermen gecombineerd met de priemimplicanten maar laten de mintermen die als don't care te boek staan weg. Zie tabel A.17.

We stellen nu de Petrick-functie op:

$$\begin{aligned} P_{f_{x_3,x_2,x_1,x_0}} &= A\cdot(A+B)\cdot B\cdot(A+C)\cdot(C+D)\cdot(C+D+E) \\ &= A\cdot B\cdot(C+D) \\ &= A\cdot B\cdot C + A\cdot B\cdot D \end{aligned} \tag{A.43}$$

We kunnen nu kiezen uit twee functies:

$$\begin{aligned} f_{x_3,x_2,x_1,x_0} &= A+B+C = \overline{x_1}\cdot\overline{x_0} + x_2\cdot\overline{x_0} + x_3\cdot\overline{x_1} \\ f_{x_3,x_2,x_1,x_0} &= A+B+D = \overline{x_1}\cdot\overline{x_0} + x_2\cdot\overline{x_0} + x_3\cdot x_0 \end{aligned} \tag{A.44}$$

Tabel A.16: *Uitwerking van de tabellen.*

Kolom I.

	x_3	x_2	x_1	x_0	m.t.	
Groep 0	0	0	0	0	0	✓
Groep 1	0	1	0	0	4	✓
	1	0	0	0	8	✓
Groep 2	0	1	1	0	6	✓
	1	0	0	1	9	✓
	1	1	0	0	12	✓
Groep 3	1	0	1	1	11	✓
	1	1	0	1	13	✓
	1	1	1	0	14	✓
Groep 4	1	1	1	1	15	✓

Kolom II.

x_3	x_2	x_1	x_0	m.t.	
0	–	0	0	0,4	✓
–	0	0	0	0,8	✓
0	1	–	0	4,6	✓
–	1	0	0	4,12	✓
1	0	0	–	8,9	✓
1	–	0	0	8,12	✓
–	1	1	0	6,14	✓
1	0	–	1	9,11	✓
1	–	0	1	9,13	✓
1	1	0	–	12,13	✓
1	1	–	0	12,14	✓
1	–	1	1	11,15	✓
1	1	–	1	13,15	✓
1	1	1	–	14,15	✓

Kolom III.

s_3	s_2	s_1	s_0	m.t.	
–	–	0	0	0,4,8,12	*A*
–	–	0	0	0,8,4,12	
–	1	–	0	4,6,12,14	*B*
–	1	–	0	4,12,6,14	
1	–	0	–	8,9,13,12	*C*
1	–	0	–	8,12,9,13	
1	–	–	1	9,11,13,15	*D*
1	–	–	1	9,13,11,15	
1	1	–	–	12,13,14,15	*E*
1	1	–	–	12,14,13,15	

Tabel A.17: *Dekkingstabel van de priemplicanten.*

	m_0	m_4	m_6	m_8	m_9	m_{13}	m_{14}
A	Ⓧ	x		x			
B		x	Ⓧ				x
C				x	x	x	
D					x	x	
E						x	x

Uitwerking opgave 4.7.

Een multiplexer kan gezien worden als een schakeling die een som-van-mintermen kan nabootsen. Een multiplexer met stuuringangen s_1 en s_0 en data-ingangen i_0, i_1, i_2 en i_3 heeft de logische functie $F = \overline{s_1}\,\overline{s_0} i_0 + \overline{s_1} s_0 i_0 + s_1 \overline{s_0} i_2 + s_1 s_0 i_3$ waarbij F natuurlijk de uitgang is.

In de twee multiplexer-schakelingen wordt s_1 verbonden met Y en s_0 verbonden met Z. Aan de data-ingangen worden nu logische constanten (0 en 1), X of $\overline{X}$ aangeboden.

Kijken we naar functie S dan zien we dat i_0 wordt verbonden met X, i_1 en i_2 met $\overline{X}$ en i_3 met X. De functie voor S is dan $S = \overline{Y}\,\overline{Z}X + \overline{Y}Z\overline{X} + Y\overline{Z}\,\overline{X} + YZX$. We kunnen de functie herschrijven zodat X vooraan staat in de termen.

Kijken we naar functie C dan zien we dat i_0 wordt verbonden met 0, i_1 en i_2 met X en i_3 met 1. De functie voor S is dan $C = \overline{Y}\,\overline{Z}0 + \overline{Y}ZX + Y\overline{Z}X + YZ1$. We kunnen de functie herschrijven als $C = YZ + X\overline{Y}Z + XY\overline{Z}$.

Hieronder zijn de waarheidstabellen gegeven van beide functies. Een geoefend lezer ziet in S de functie voor het sombit van een full-adder en in C de *carry-out*-bit van een full-

adder. Merk op dat de volgorde van de variabelen in de tabel afwijkt van wat gebruikelijk is; variabele X is nu het minst significant. Op deze manier kunnen de functiewaarden van C en S goed vergeleken worden met de aan de data-ingangen aangeboden waarden van 0, 1, X en $\overline{X}$.

Tabel A.18: *Waarheidstabel.*

Y	Z	X	C	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

Uitwerking opgave 4.8.

De schakeling links in de figuur is een 2-input NAND-poort. Dat is als volgt te beredeneren. Als beide ingangen 0 zijn, dan geleiden T_1 en T_2. T_3 en T_4 sperren. Er is zodoende een elektrisch pad van U_{DD} naar de uitgang. De uitgang is dus 1. Als $a = 0$ en $b = 1$, dan geleidt T_1 en spert T_2. T_3 spert en T_4 geleidt. Er is weer een pad van U_{DD} naar de uitgang, de uitgang is 1. Eenzelfde situatie doet zich voor als $a = 1$ en $b = 0$. Als de ingangen beide 1 zijn, dan sperren T_1 en T_2 en geleiden T_3 en T_4. Er is nu een pad naar de referentie, de uitgang is 0. Alle combinaties zijn nog eens weergegeven in de onderstaande tabel. De schakeling werkt dus als 2-input NAND. Merk op dat voor deze NAND vier transistoren nodig zijn.

Tabel A.19: *Waarheidstabel.*

a	b	T_1	T_2	T_3	T_4	f
0	0	geleidt	geleidt	spert	spert	1
0	1	geleidt	spert	spert	geleidt	1
1	0	spert	geleidt	geleidt	spert	1
1	1	spert	spert	geleidt	geleidt	0

De schakeling rechts in de figuur is een samenstelling van de hierboven beschreven NAND-poort en een NOT-poort. De schakeling werkt in zijn geheel als een AND-poort. Voor deze poort zijn zes transistoren nodig.

Uitwerking opgave 5.1.

Hieronder de optellingen mét carry's. De eerste optelling levert een *overflow* op omdat het antwoord niet in zeven bits past.

Uitwerking opgave 5.2.

1110110		10100		11110		carry's
1011001		01010		01111		getal A
0111011	+	01010	+	00001	+	getal B
1)0010100		10100		10000		resultaat

Figuur A.31: *Enkele optellingen met carry's.*

Hieronder de aftrekkingen mét borrows. De derde aftrekking levert een *underflow* op omdat het antwoord niet in zes bits past.

1111100		111100		1)000010		borrows
1011001		110000		010110		getal A
0111011	−	010110	−	100101	−	getal B
0011110		011010		1)110001		resultaat

Figuur A.32: *Enkele aftrekkingen met borrows.*

Uitwerking opgave 5.3.
Het bewijs is niet zo moeilijk. Er wordt gebruik gemaakt van een absorptiewet en de commutatieve wetten.

$$\begin{aligned} c_{out} &= \overline{c_{in}} \cdot (a \cdot b) + c_{in} \cdot (a + b) \\ &= \overline{c_{in}} \cdot a \cdot b + c_{in} \cdot a + c_{in} \cdot b \\ &= a \cdot (\overline{c_{in}} \cdot b + c_{in}) + c_{in} \cdot b \\ &= a \cdot (b + c_{in}) + c_{in} \cdot b \\ &= a \cdot b + a \cdot c_{in} + b \cdot c_{in} \end{aligned} \tag{A.45}$$

Uitwerking opgave 5.4.
Dit is wel een lastige opgave. Van cruciaal belang is dat de term $a \cdot b$ uitgebreid moet worden naar $a \cdot b + a \cdot b$. Daarna kan de absorptiewet worden toegepast. Zie hieronder.

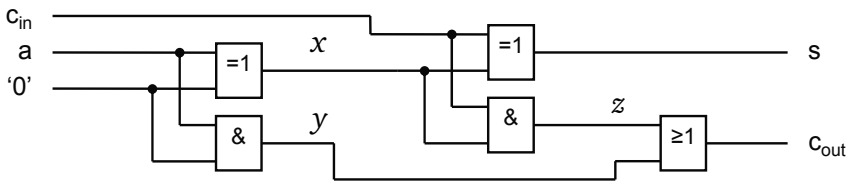
$$\begin{aligned} c_{out} &= a \cdot b + c_{in} \cdot (a \oplus b) \\ &= a \cdot b + c_{in} \cdot \overline{a} \cdot b + c_{in} \cdot a \cdot \overline{b} \\ &= a \cdot b + a \cdot b + c_{in} \cdot \overline{a} \cdot b + c_{in} \cdot a \cdot \overline{b} \\ &= (a + c_{in} \cdot \overline{a}) \cdot b + a \cdot (b + c_{in} \cdot \overline{b}) \\ &= (a + c_{in}) \cdot b + a \cdot (b + c_{in}) \\ &= a \cdot b + c_{in} \cdot b + a \cdot b + a \cdot c_{in} \\ &= a \cdot b + a \cdot c_{in} + b \cdot c_{in} \end{aligned} \tag{A.46}$$

Uitwerking opgave 5.5.
Als de functies van s en c_{out} vanuit de 0-en zouden worden gemaakt, levert dat net zulke grote functies op als wanneer ze vanuit de 1-en zouden worden gemaakt. Dat is ook niet

zo verwonderlijk. Voor s geldt dat (vanuit de 1-en gezien) het aantal 1-en oneven moet zijn, dus dat geldt dan ook voor de 0-en. Voor c_{out} geldt dat (weer vanuit de 1-en gezien) het aantal 1-en twee of meer moet zijn en ook dat geldt dan natuurlijk voor de 0-en.

Uitwerking opgave 5.6.

Hieronder is het schema gegeven met daarin interne signaalnamen.



Figuur A.33: *Schema van een full adder.*

De ingang waar de constante 0 aan verbonden is herkennen we als de b-ingang van een full-adder. Deze ingang koppelen we ook nog eens aan de constante 1 en rekenen beide schakelingen door.

De functies voor $b = 0$

$$\begin{aligned} x &= a \oplus 0 & &= a \\ y &= a \cdot 0 & &= 0 \\ z &= x \cdot c_{in} & &= a \cdot c_{in} \\ s &= c_{in} \oplus x & &= c_{in} \oplus a \\ c_{out} &= y + z & &= a \cdot c_{in} \end{aligned}$$

De functies voor $b = 1$

$$\begin{aligned} x &= a \oplus 1 & &= \overline{a} \\ y &= a \cdot 1 & &= a \\ z &= x \cdot c_{in} & &= \overline{a} \cdot c_{in} \\ s &= c_{in} \oplus x & &= c_{in} \oplus \overline{a} \\ c_{out} &= y + z & &= a + c_{in} \end{aligned} \qquad \text{(A.47)}$$

In de functies aan de linkerkant herkennen we de functies van een half-adder.

Uitwerking opgave 5.7.

De vermenigvuldiger vermenigvuldigt twee *unsigned* getallen van twee bits met elkaar. De rekenkundige functie is dan $M = A \times B$ waarbij A samengesteld is uit de bits a_1 en a_0 en B uit b_1 en b_0. Een 2x2 bits vermenigvuldiging levert een antwoord van maximaal vier bits dus M is samengesteld uit de bits m_3, m_2, m_1 en m_0. De grootste waarde van M is trouwens 9, want $A = 3$ en $B = 3$ levert $M = 3 \times 3 = 9$ op.

We stellen een waarheidstabel op met a_1, a_0, b_1 en b_0 op als ingangen en m_3, m_2, m_1 en m_0 als de uitgangen.

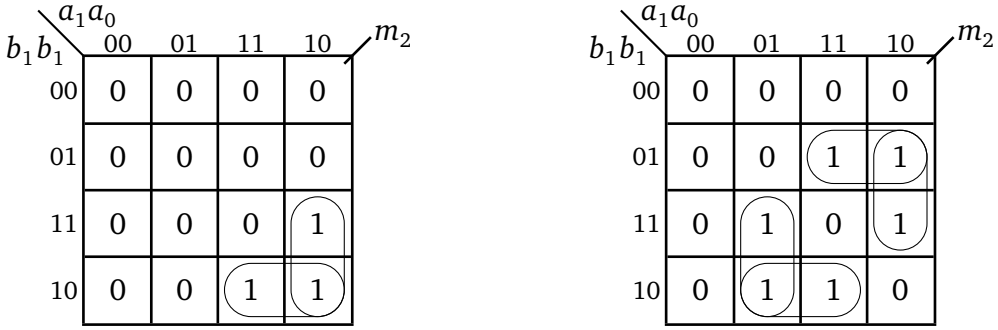
Snel is te zien dat de functie voor m_3 een AND-operatie is van alle ingangen zodat $m_3 = a_1 \cdot a_0 \cdot b_1 \cdot b_0$. De functie voor m_0 is ook redelijk snel te ontdekken: $m_0 = 1$ als $a_0 = 1$ én $b_0 = 1$ dus $m_0 = a_0 \cdot b_0$.

Voor de andere twee functies stellen een Karnaughdiagram op.

Tabel A.20: *Waarheidstabel van een 2x2-bit vermenigvuldiger.*

a_1	a_0	b_1	b_0	m_3	m_2	m_1	m_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1

a_1	a_0	b_1	b_0	m_3	m_2	m_1	m_0
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1



Figuur A.34: *Karnaughdiagrammen.*

De vier functies voor deze schakeling zijn:

$$m_3 = a_1 \cdot a_0 \cdot b_1 \cdot b_0$$
$$m_2 = a_1 \cdot b_1 \cdot \overline{b_0} + a_1 \cdot \overline{a_0} \cdot b_1$$
$$m_1 = \overline{a_1} \cdot a_0 \cdot b_1 + a_0 \cdot b_1 \cdot \overline{a_0} + a_1 \cdot \overline{a_0} \cdot b_0 + a_1 \cdot \overline{b_1} \cdot b_0$$
$$m_0 = a_0 \cdot b_0$$

Het bijbehorende schema met AND-, OR- en NOT-poorten is hieronder gegeven.

Uitwerking opgave 5.8.
Als geldt dat twee getallen $A = a_3a_2a_1a_0$ en $B = b_3b_2b_1b_0$ aan elkaar gelijk zijn, dan moeten cijfers op identieke posities binnen de getallen aan elkaar gelijk zijn:

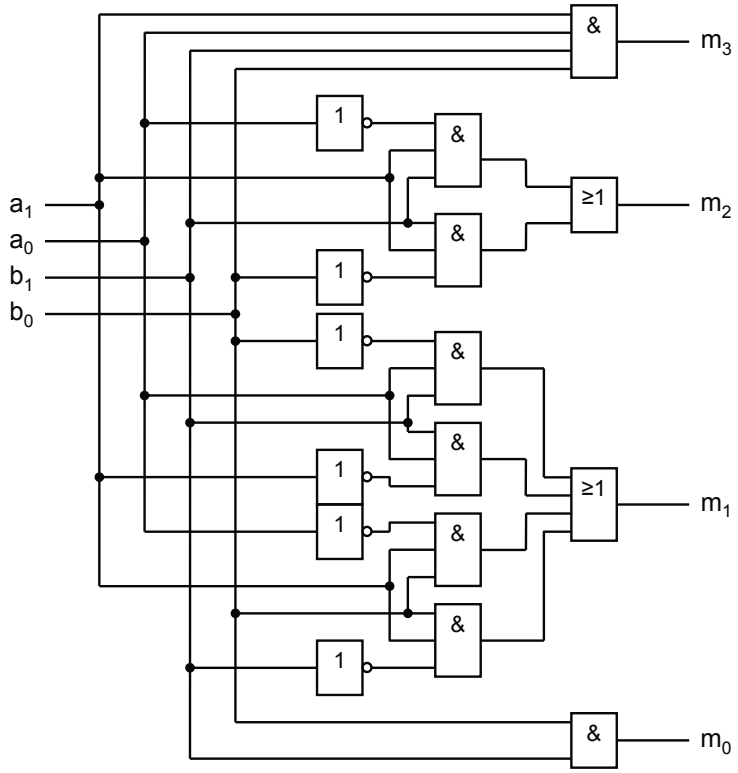
$$A = B \quad \rightarrow \quad a_3 = b_3 \wedge a_2 = b_2 \wedge a_1 = b_1 \wedge a_0 = b_0 \tag{A.48}$$

Het testen of twee bits gelijk zijn kan met de EXNOR-functie. De schakelfunctie is:

$$E = \overline{a_3 \oplus b_3} \cdot \overline{a_2 \oplus b_2} \cdot \overline{a_1 \oplus b_1} \cdot \overline{a_0 \oplus b_0} \tag{A.49}$$

of, m.b.v. De Morgan:

$$E = \overline{(a_3 \oplus b_3) + (a_2 \oplus b_2) + (a_1 \oplus b_1) + (a_0 \oplus b_0)} \tag{A.50}$$



Figuur A.35: *Schema van de 2x2-bit vermenigvuldiger.*

Uitwerking opgave 5.9.

Hieronder is het schema gegeven van 4x4 carry save multiplier. Bij een “normale” multiplier wordt het optellen uitgevoerd met carry ripple adders waarbij de carry steeds wordt doorgegeven aan de hogere macht van dezelfde opteller.

Bij de carry save multiplier wordt de carry van een 1-bit full adder circuit doorgegeven aan de hogere macht van de *volgende* 4-bit opteller.

Er zijn nu wel meer rijen nodig, maar minder kolommen.

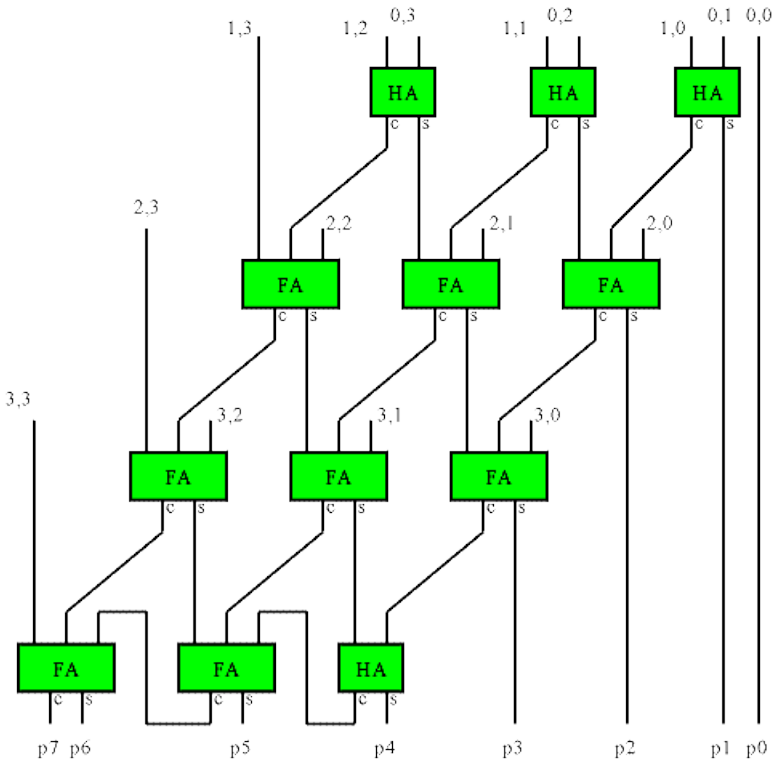
Uitwerking opgave 5.10.

Voor een 5x3-bit vermenigvuldig zijn twee 5-bit optellers, tenminste als de optellers worden uitgevoerd als *carry ripple adders*.

Uitwerking opgave 5.11.

De getallen moeten omgerekend worden naar binaire equivalenten. Van negatieve getallen moet eerst de positieve variant worden omgerekend. Van de getallen +5 en −5 kan direct het binaire equivalent worden uitgerekend (althans van +5).

Het getal +5 als binair getal is 0101_2. Let hierbij op de leidende 0 om het getal positief te houden. Het getal staat nu in 4 bits genoteerd, we moeten het dus aanvullen met meer nullen (het getal is immers positief). Dat levert het binaire getal 00000101_2 op. Dit staat nu al in 2’s complement.



Figuur A.36: *Opbouw van een carry save vermenigvuldiger.*

Het getal −5 moet bepaald worden volgens de 'flip-bit, add-one'-methode. We weten al hoe +5 in 8-bits formaat geschreven wordt.

$$\begin{array}{rl} 00000101 & (+5) \\ \\ 11111010 & \text{flip bits} \\ \qquad\quad 1 & + \\ \hline 11111011 & (-5) \end{array}$$

Figuur A.37: *De binaire representatie in 8 bits van de getallen* +5 *en* −5*.*

Het getal −5 geschreven in 8 bits 2's complement is 11111011_2.

Eerst moet het getal +100 omgezet worden in het binaire stelsel (we laten dat als oefening voor de lezer). Dat levert het binaire getal 01100100_2. Let hierbij op de leidende 0 om het getal positief te houden. Dit getal staat al in 8-bits formaat, dus aanvullen is niet nodig. Omzetten naar de negatieve variant:

Het getal −100 geschreven in 8 bits 2's complement is 10011100_2.

Het getal −64 moet omgezet worden naar 2's complement. Daarvoor moet eerst de positieve variant 64 omgezet worden naar binair (dat laten we weer over als oefening voor de lezer). Dit levert het getal 01000000_2 op (let hierbij op de leidende 0 om het getal positief te houden. Het getal is al in 8 bits genoteerd, er is geen tekenuitbreiding nodig. Omzetten naar de negatieve variant:

01100100	(+100)
10011011	flip bits
1	+
10011100	(−100)

Figuur A.38: *De binaire representatie in 8 bits van de getallen* +100 *en* −100*.*

01000000	(+64)
10111111	flip bits
1	+
11000000	(−64)

Figuur A.39: *De binaire representatie in 8 bits van de getallen* +64 *en* −64*.*

Het getal −64 geschreven in 8 bits 2's complement is 1100000_2.

Het getal +25 is volgens de bekende procedure om te zetten naar het binaire equivalent 011001_2. Uitbreiden naar 8 bits levert 00011001_2 op.

Van het getal −25 wordt alleen het antwoord gegeven: 11100111_2.

Uitwerking opgave 5.12.
Bij deze opgave moet een aantal optellingen en aftrekkingen worden gedaan. Optellingen kunnen direct uitgevoerd worden d.m.v. de bekende kolomsgewijze optelling. Aftrekkingen kunnen worden omgezet in optellingen door bekende vergelijking

$$A - B = A + (-B) \tag{A.51}$$

Hiervoor is wel de negatieve representatie van B nodig. Dit kan eenvoudig in de 2's complement-representatie. Daarnaast moet een aantal getallen met meer bits worden geschreven zodat de op te tellen getallen evenveel bits hebben. Dat kan d.m.v. tekenuitbreiding.

De eerste optelling is 011001 + 100100. Het eerste getal is positief, het tweede getal is negatief. De getallen kunnen direct opgeteld worden. De tweede optelling is 110011 + 100011. Beide getallen zijn negatief en kunnen direct worden opteld.

000000		000110		carry's
011001		110011		getal A
100100	+	100011	+	getal B
0)111101		1)010110		resultaat

Figuur A.40: *Enkele optellingen.*

De linker optelling levert een negatief getal op. Er is geen overflow want de op te tellen getallen hebben verschillend teken. De uitgaande carry is 0. De tweede optelling levert

een overflow op. De op te tellen getallen zijn beide negatief en het antwoord is positief. De uitgaande carry is 1 en moet genegeerd worden. Aan de uitgaande carry is niet te zien of er overflow is opgetreden.

Hierna volgen twee aftrekkingen namelijk $011001 - 001110$ en $101010 - 010101$. De aftrekkingen kunnen worden omgezet in optellingen door het getal dat moet worden afgetrokken om te zetten in de negatieve representatie van het originele getal. Omzetten gaat heel eenvoudig met de procedure *flip bits, add 1* zoals uitgelegd is in opgave 5.11. Het getal 001110 wordt dan 110010 en het getal 010101 wordt 101011.

100000		010100		carry's
011001		101010		getal A
110010	+	101011	+	getal B
1)001011		1)010101		resultaat

Figuur A.41: *Enkele optellingen.*

De uitgaande carry's moeten weer genegeerd worden. De linker optelling levert geen overflow op, de rechter optelling levert wel een overflow. Het betreft hier nu twee negatieve getallen die opgeteld een positief getal opleveren.

De volgende optelling is $1101 + 111001$. Hierin is één van de getallen maar in vier bits geschreven en moet worden uitgebreid naar zes bits. Het getal begint met een 1 (negatief) en er moeten dan twee 1-en voor gezet worden. De optelling wordt dan $111101+111001$. Daarna volgt een aftrekking met de getallen $101011 - 1100111$. Het eerste getal moet uitgebreid worden met een 1, het is immers negatief. Het betreft hier een aftrekking dus het tweede getal moet omgezet worden in de negatieve representatie van het originele getal en levert dan 0011001. Nu kunnen de twee optellingen uitgevoerd worden.

110010		1011110		carry's
111101		1101011		getal A
111001	+	1100111	+	getal B
1)110110		1)1010010		resultaat

Figuur A.42: *Enkele optellingen.*

De laatste twee berekeningen zijn een aftrekking en een optelling. De aftrekking betreft $011100 - 011001$ en de optelling betreft $10001 + 1100111$. Er moeten weer omzettingen in 2's complement en tekenuitbreiding gedaan worden. Uiteindelijk levert dat de onderstaande optellingen op:

111000		1001110		carry's
011100		1111001		getal A
100111	+	1100111	+	getal B
1)000011		1)1011000		resultaat

Figuur A.43: *Enkele optellingen.*

Beide uitkomsten leveren geen overflow op. De uitgaande carry's moeten genegeerd worden.

Uitwerking opgave 5.13.

Een hexadecimaal 2's complement getal dat begint met de cijfers 8 t/m F wordt als negatief beschouwd omdat het binaire equivalent van dat cijfer begint met een 1. Zo is 8_{16} gelijk aan 1000_2 en F_{16} gelijk aan 1111_2. Getallen die beginnen met de cijfers 0 t/m 7 zijn positief omdat het binaire equivalent van deze cijfers beginnen met een 0.

Als we het getal $FFFF_{16}$ opschrijven als binair getal dan levert dat $1111.1111.1111.1111_2$ (punten worden gebruikt ter bevordering van het lezen). Zouden we dit omzetten naar het decimale equivalent dan levert dat behoorlijk wat schrijfwerk op:

$$\begin{aligned} FFFF_{16} &= -2^{15} + 2^{14} + 2^{13} + \ldots + 2^2 + 2^1 + 2^0 \\ &= -32768 + 16384 + 8192 + \ldots + 4 + 2 + 1 \\ &= -1 \end{aligned}$$

Handiger is om direct gebruik te maken van de hexadecimale cijfers waarbij het meest significante cijfer als negatief wordt beschouwd is dit cijfer tussen 8 en F ligt:

$8_{16} = \text{-}8_{10}$ $9_{16} = \text{-}7_{10}$ $A_{16} = \text{-}6_{10}$ $B_{16} = \text{-}5_{10}$ $C_{16} = \text{-}4_{10}$ $D_{16} = \text{-}3_{10}$ $E_{16} = \text{-}2_{10}$ $F_{16} = \text{-}1_{10}$

We kunnen nu $FFFF_{16}$ direct uitschrijven in machten van 16:

$$\begin{aligned} FFFF_{16} &= -1 \cdot 16^3 + 15 \cdot 16^2 + 15 \cdot 16^1 + 15 \cdot 16^0 \\ &= -4096 + 3840 + 240 + 15 \\ &= -1 \end{aligned}$$

En de rest van de getallen:

$$\begin{aligned} 53BA_{16} &= 5 \cdot 16^3 + 3 \cdot 16^2 + 11 \cdot 16^1 + 10 \cdot 16^0 \\ &= 20480 + 768 + 176 + 10 \\ &= 21434 \end{aligned}$$

$$\begin{aligned} CB_{16} &= -4 \cdot 16^1 + 10 \cdot 16^0 \\ &= -64 + 10 \\ &= -54 \end{aligned}$$

$$\begin{aligned} 7EA3_{16} &= 7 \cdot 16^3 + 14 \cdot 16^2 + 10 \cdot 16^1 + 3 \cdot 16^0 \\ &= 28672 + 3584 + 160 + 3 \\ &= 32419 \end{aligned}$$

Uitwerking opgave 5.14.

Een hexadecimaal getal van acht cijfers kan worden geschreven als een binair getal van 32 cijfers. De bereik van zo'n getal is

$$-2^{31} \leq M \leq +2^{31} - 1 \tag{A.52}$$

oftewel

$$-2147483648 \leq M \leq +2147483647 \tag{A.53}$$

Uitwerking opgave 5.15.

Tekenuitbreiding bij hexadecimale getallen is niet zo heel ingewikkeld. Als het meest significante cijfer tussen 0 en 7 ligt, moet het getal uitgebreid worden met voorlopende nullen (een 0 levert vier nullen op: $0 \rightarrow 0000$). Ligt het meest significante cijfer tussen 8 en F, dan moet het getal worden uitgebreid met voorlopende F-en (een F levert vier enen op: $F \rightarrow 1111$). Dus:

$F_{16} \rightarrow FFFF_{16}$ $\quad 6_{16} \rightarrow 0006_{16}$ $\quad 7A_{16} \rightarrow 007A_{16}$ $\quad CB_{16} \rightarrow FFCB_{16}$ $\quad 35B_{16} \rightarrow 035B_{16}$

$D73_{16} \rightarrow FD73_{16}$

Uitwerking opgave 5.16.

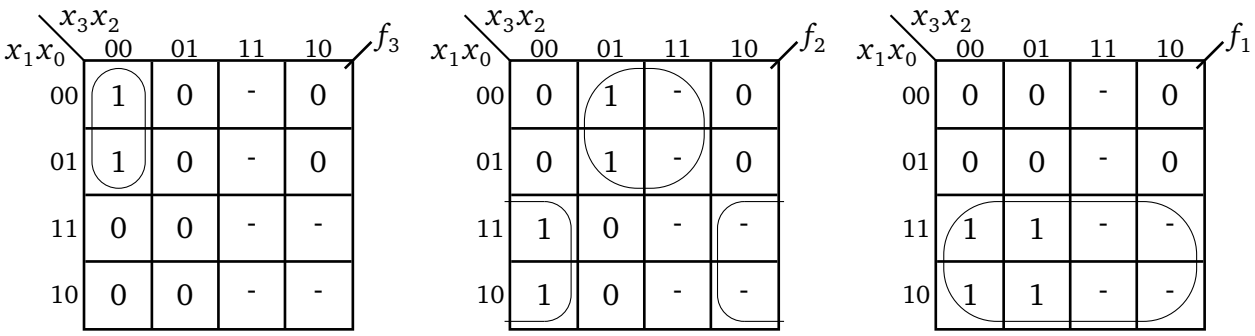
We noemen de ingangen $X = x_3x_2x_1x_0$ en de uitgangen $F = f_3f_2f_1f_0$. De werking van de 9's-complement-functie kan worden beschreven met de functie

$$F = 9 - X \qquad \text{voor } 0 \leq X \leq 9 \tag{A.54}$$

Eerst moet de waarheidstabel worden opgezet. De bitcombinaties $X = x_3x_2x_1x_0$ kunnen gezien worden als een getal dat ligt tussen 0 en 15, waarvan alleen de combinaties 0 t/m 9 gebruikt worden, de overige komen niet voor. Dat worden don't cares in de waarheidstabel. De bitcombinaties $F = f_3f_2f_1f_0$ vormen de getallen $9-X$, dus X moet van 9 afgetrokken worden.

Van alle mogelijkheden wordt een waarheidstabel opgezet.

Vanuit de tabel is direct te zien dat $f_0 = \overline{x_0}$. Voor de overige functies worden Karnaugh-diagrammen opgezet en uitgewerkt. Deze zijn weergegeven in onderstaande figuur.



Figuur A.44: *Karnaughdiagrammen voor de functies f_3, f_2 en f_0.*

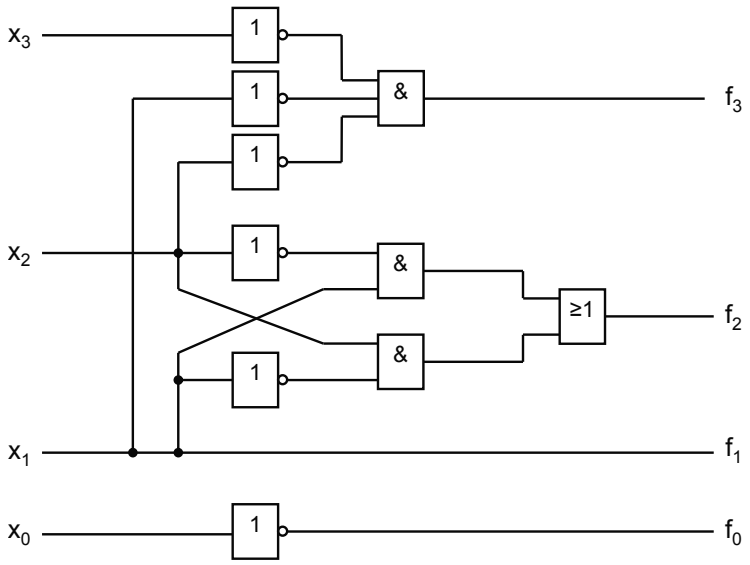
De gevonden functies voor f_3 t/m f_0 zijn:

$$\begin{aligned} f_3 &= \overline{x_3} \cdot \overline{x_2} \cdot \overline{x_1} \\ f_2 &= \overline{x_2} \cdot x_1 + x_2 \cdot \overline{x_1} \\ f_1 &= x_1 \\ f_0 &= \overline{x_0} \end{aligned} \tag{A.55}$$

Tabel A.21: *Waarheidstabel van de 9's complement functie.*

x_3	x_2	x_1	x_0	f_3	f_2	f_1	f_0
0	0	0	0	1	0	0	1
0	0	0	1	1	0	0	0
0	0	1	0	0	1	1	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	0	1	1
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	—	—	—	—
1	0	1	1	—	—	—	—
1	1	0	0	—	—	—	—
1	1	0	1	—	—	—	—
1	1	1	0	—	—	—	—
1	1	1	1	—	—	—	—

De schema's van de vier functies zijn weergegeven in onderstaande figuur.



Figuur A.45: *Schema 9's complement functie.*

Uitwerking opgave 5.17.

Het is een kwestie van interpretatie.

In de 2's representatie systeem wordt een binair getal dat begint met een 1 als negatief beschouwd. Het meest negatieve getal is een 1 gevolgd door alleen 0-en. Maar er is niets op tegen om dit ene getal als positief te beschouwen.

Laten we eens kijken wat er met optellingen gebeurt als we 1000_2 als +8 beschouwen. We tellen bij alle getallen van −7 tot +8(!) de waarde +8 op en interpreteren het resultaat als 2's complement waarbij 1000_2 als +8 wordt beschouwd. Uiteraard gooien we een eventuele carry weg.

$$\begin{array}{rcll}
-7+8 & \rightarrow & 1001+1000=1)0001 & \rightarrow \quad +1 \\
-6+8 & \rightarrow & 1010+1000=1)0010 & \rightarrow \quad +2 \\
-5+8 & \rightarrow & 1011+1000=1)0011 & \rightarrow \quad +3 \\
-4+8 & \rightarrow & 1100+1000=1)0100 & \rightarrow \quad +4 \\
-3+8 & \rightarrow & 1101+1000=1)0101 & \rightarrow \quad +5 \\
-2+8 & \rightarrow & 1110+1000=1)0110 & \rightarrow \quad +6 \\
-1+8 & \rightarrow & 1111+1000=1)0111 & \rightarrow \quad +7 \\
0+8 & \rightarrow & 0000+1000=0)1000 & \rightarrow \quad +8 \\
+1+8 & \rightarrow & 0001+1000=0)1001 & \rightarrow \quad -7 \\
+2+8 & \rightarrow & 0010+1000=0)1010 & \rightarrow \quad -6 \\
+3+8 & \rightarrow & 0011+1000=0)1011 & \rightarrow \quad -5 \\
+4+8 & \rightarrow & 0100+1000=0)1100 & \rightarrow \quad -4 \\
+5+8 & \rightarrow & 0101+1000=0)1101 & \rightarrow \quad -3 \\
+6+8 & \rightarrow & 0110+1000=0)1110 & \rightarrow \quad -2 \\
+7+8 & \rightarrow & 0111+1000=0)1111 & \rightarrow \quad -1 \\
+8+8 & \rightarrow & 1000+1000=1)0000 & \rightarrow \quad 0
\end{array}$$

Figuur A.46: *Gebruik van bitpatroon 1000 als* +8.

De optellingen onder −7 tot 0 leveren correcte antwoorden op, die van +1 tot +8 geven een overflow.

De reden waarom 1000_2 als −8 wordt beschouwd is heel simpel: het levert eenvoudigere hardware op. Begint het getal met een 1, dan is het negatief, anders is het positief. Het tekenbit kan dan ook als onderdeel van het getal worden gezien en heeft dus ook een (negatief) gewicht.

Uitwerking opgave 5.18.

Bij de eerste twee optellingen treedt overflow op; het antwoord past niet in het aantal bits waaruit de opgetelde getallen bestaan. De laatste optelling genereert geen overflow.

0001 0000		0000		0001 0000 0000		carry's
1001 1001		0011		0101 1000 0110		getal A
0011 1001	+	0111	+	0000 0111 0001	+	getal B
1) 0010 1000		1) 0000		0110 0101 0111		resultaat

Figuur A.47: *Enkele optellingen met BCD-gecodeerde getallen.*

Uitwerking opgave 5.19.

We noemen de ingangen $X = x_3x_2x_1x_0$ en de uitgangen $F = f_3f_2f_1f_0$. De werking van de excess-3-functie kan worden beschreven met de functie

$$F = X + 3 \qquad \text{voor } 0 \leq X \leq 9 \tag{A.56}$$

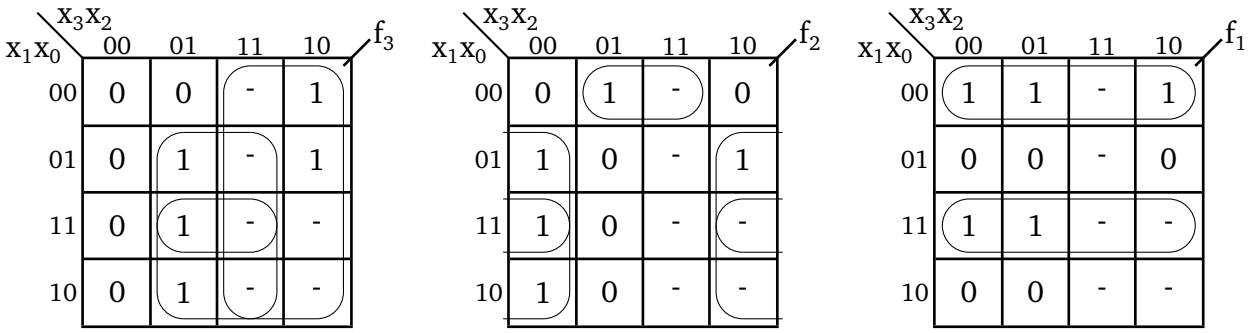
Eerst moet de waarheidstabel worden opgezet. De bitcombinaties $X = x_3x_2x_1x_0$ kunnen gezien worden als een getal dat ligt tussen 0 en 15, waarvan alleen de combinaties 0 t/m 9 gebruikt worden, de overige komen niet voor. Dat worden don't cares in de waarheidstabel. De bitcombinaties $F = f_3f_2f_1f_0$ vormen de getallen $X+3$, dus bij X moet 3 opgeteld worden.

Van alle mogelijkheden wordt een waarheidstabel opgezet.

Tabel A.22: *Waarheidstabel voor omzetter van BCD-code naar Excess-3-code.*

x_3	x_2	x_1	x_0	f_3	f_2	f_1	f_0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	-	-	-	-
1	0	1	1	-	-	-	-
1	1	0	0	-	-	-	-
1	1	0	1	-	-	-	-
1	1	1	0	-	-	-	-
1	1	1	1	-	-	-	-

Vanuit de tabel is direct te zien dat $f_0 = \overline{x_0}$. Voor de overige functies worden Karnaugh-diagrammen opgezet en uitgewerkt. Deze zijn weergegeven in onderstaande figuur.



Figuur A.48: *Karnaughdiagrammen BCD-naar-Excess-3-code.*

De gevonden functies voor f_3 t/m f_0 zijn:

$$\begin{aligned} f_3 &= x_3 + x_2 \cdot x_1 + x_2 \cdot x_0 \\ f_2 &= \overline{x_2} \cdot x_1 + \overline{x_2} \cdot x_0 + x_2 \cdot \overline{x_1} \cdot \overline{x_0} \\ f_1 &= \overline{x_1} \cdot \overline{x_0} + x_1 \cdot x_0 \\ f_0 &= \overline{x_0} \end{aligned} \tag{A.57}$$

De functie voor f_3 kan nog geschreven worden als

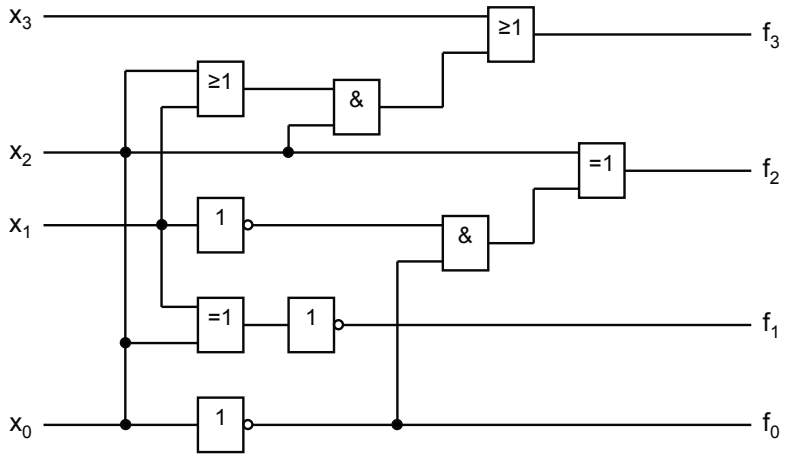
$$f_3 = x_3 + x_2 \cdot (x_1 + x_0) \tag{A.58}$$

waardoor een AND-poort komt te vervallen.

De functies voor f_2 en f_1 kunnen nog omgewerkt worden naar

$$\begin{aligned} f_2 &= x_2 \oplus \left(\overline{x_1} \cdot \overline{x_0}\right) \\ f_1 &= \overline{x_1 \oplus x_0} \end{aligned} \tag{A.59}$$

zodat de inverter voor x_2 komt te vervallen. De schema's van de vier functies zijn weergegeven in onderstaadne figuur. Noot: er zijn ook andere schema's mogelijk die de juiste functies weergeven.



Figuur A.49: *Schema omzetter BCD-naar-Excess-3-code.*

Uitwerking opgave 5.21.
Het aantal bit van de mantisse is 23. De nauwkeurigheid is dan 2^{-23}. Het aantal significante decimale cijfers laat zich als volgt berekenen:

$$2^{-23} = 10^x \qquad \longrightarrow \qquad x = -23 \cdot \log 2 = -6{,}9\cdots \tag{A.60}$$

Het aantal significante cijfers is ongeveer 7.

Uitwerking opgave 6.1.
De twee don't cares kunnen als volgt worden ingevuld:

Tabel A.23: *Waarheidstabel voor de twee don't cares bij de SR-latch.*

S	R	Z_{oud}	Z_{nieuw}	O.S.	O.R.	#3	#4
0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1
0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0
1	0	0	1	1	1	1	1
1	0	1	1	1	1	1	1
1	1	0	-	1	0	0	1
1	1	1	-	1	0	1	0

Hierin is Z_{nieuw} de originele specificatie en zijn O.S. en O.R. de realisatie van resp. overheersende set en overheersende reset. Bij de realisatie van #3 valt op dat de uitgangswaarden bij $SR = 11$ precies hetzelfde zijn als de waarde van Z_{oud}, dus de latch onthoudt de waarde (= stand) van Z_{oud}. Dit is dus een latch met een extra onthoudstand ($SR = 00$ is er ook één). Bij de realisatie van #4 valt op dat de uitgangswaarden bij $SR = 11$ precies de inverse zijn van de waarde van Z_{oud}, dus de latch inverteert de stand. Omdat dit direct werkt (er is geen klok- of enablesignaal om de latch te synchroniseren) zal zodra een stand aan de uitgang bekend is, de inverse worden berekend en na enige poortvertragingen deze nieuwe stand op de uitgang beschikbaar zijn. Deze schakeling oscilleert bij $SR = 11$. #3 is een bruikbare latch maar wordt in de praktijk niet gebruikt omdat het ontwerp meer poorten kost dan O.S. en O.R. #4 is niet bruikbaar vanwege de oscillatie.

Uitwerking opgave 6.2.

De functie van Z is: $Z_{nieuw} = \overline{A + \overline{B \cdot Z_{oud}}} = \overline{A} \cdot B \cdot Z_{oud}$. Het laatste deel is gevonden met behulp van De Morgan en geeft duidelijk aan dat de latch alleen maar te resetten is ($A = 1$ of $B = 0$) of kan onthouden ($AB = 01$). Setten is niet mogelijk. Dit is dus geen goedwerkend geheugenelement.

Uitwerking opgave 6.3.

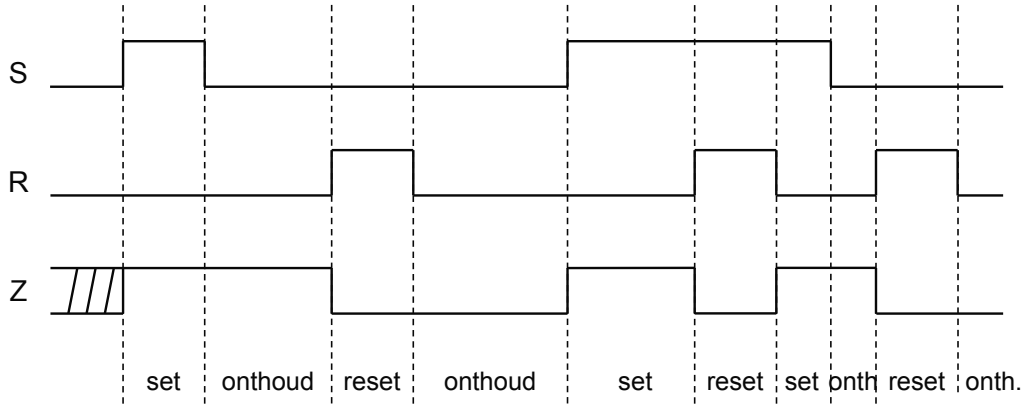
De functie van Z is: $Z_{nieuw} = A \oplus B \oplus Z_{oud}$. We kunnen dit omwerken naar: $Z_{nieuw} = \overline{A \oplus B} \cdot Z_{oud} + (A \oplus B) \cdot \overline{Z_{oud}}$. Hieruit wordt duidelijk dat de latch alleen kan onthouden of de inverse kan inlezen. Er is geen mogelijkheid om de latch te setten of te resetten.

Uitwerking opgave 6.4.

Dit is geen goede codering. De onthoudstand is de "ruststand", dus vanuit set of reset moet je terug kunnen gaan naar onthouden. Dat mag maar met één bitwisseling in de stuurcodes. Het is namelijk praktisch niet mogelijk om twee bits tegelijk te wisselen; er gaat er altijd wel één eerder om. Vanuit set moet je twee bits wisselen om bij onthouden te komen. Dat gaat dus niet goed.

Uitwerking opgave 6.5.

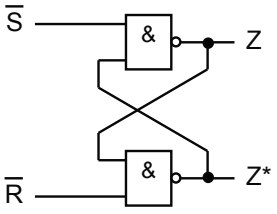
Zie onderstaand timingdiagram. Dit diagram wijkt op één punt af van het diagram voor de SR-latch met overheersende set: als S en R beide logisch 1 zijn, is de uitgang Z logisch 0.



Figuur A.50: *Signaaldiagram voor de SR-latch met overheersende reset.*

Uitwerking opgave 6.6.

Als we kijken naar de diverse latches die de revue gepasseerd zijn, dan kunnen we constateren dat alleen latches die gebouwd zijn met inverterende poorten (NAND-NAND en NOR-NOR) in aanmerking komen. Zie figuur. In een tabel kunnen we zetten hoe Z en Z^* zich verhouden. Z en Z^* zijn inderdaad elkaars inverse voor de drie combinaties de nuttig te gebruiken zijn. Bij de combinatie $SR = 11$ zijn Z en Z^* beide logisch 1.



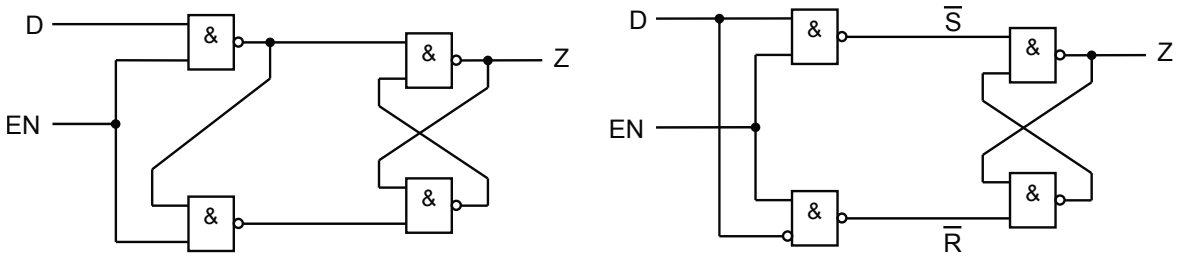
S	R	Z_{oud}	Z_{nieuw}	Z^*_{nieuw}
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

Figuur A.51: *SR-latch met NAND-poorten en bijbehorende waarheidstabel.*

Uitwerking opgave 6.7.

We tekenen twee gated D-latches naast elkaar. Links is de nieuwe te onderzoeken gated D-latch, rechts is de bekende gated D-latch die volgt uit de aangepaste gated SR-latch.

Te zien is dat beide latches bestaan uit een op NAND's gebaseerde SR-latch met voorzet-logica. De SR-latch wordt aangestuurd met de inverse van S(et) en R(eset). De signalen zijn ingetekend in de rechter latch.



Figuur A.52: *Twee realisaties van een gated D-latch.*

De functies voor de set er reset zijn (let hierbij op de naamgeving voor $\overline{S}$ en $\overline{R}$, dit zijn *signaalnamen*):

$$\begin{aligned}[\overline{S}] &= \overline{D \cdot EN} \\ [\overline{R}] &= \overline{\overline{D} \cdot EN}\end{aligned} \tag{A.61}$$

Dit moet dus ook gelden voor de te onderzoeken latch. Het is duidelijk dat de functie voor de set voor beide schakelingen identiek is. De functie voor de reset voor de te onderzoeken latch is

$$\begin{aligned}[\overline{R}] &= \overline{[\overline{S}] \cdot EN} \\ &= \overline{\overline{D \cdot EN} \cdot EN} \\ &= \overline{(\overline{D} + \overline{EN}) \cdot EN} \\ &= \overline{\overline{D} \cdot EN + \overline{EN} \cdot EN} \\ &= \overline{\overline{D} \cdot EN + 0} \\ &= \overline{\overline{D} \cdot EN}\end{aligned} \tag{A.62}$$

Merk op dat hier een keer De Morgan is gebruikt en diverse andere wetten en theorema's. Het uiteindelijke resultaat voor de reset is hetzelfde voor beide latches dus de *logische* werking is identiek. De te onderzoeken latch is met vier NANDs te bouwen en heeft geen inverter nodig bij de resetingang.

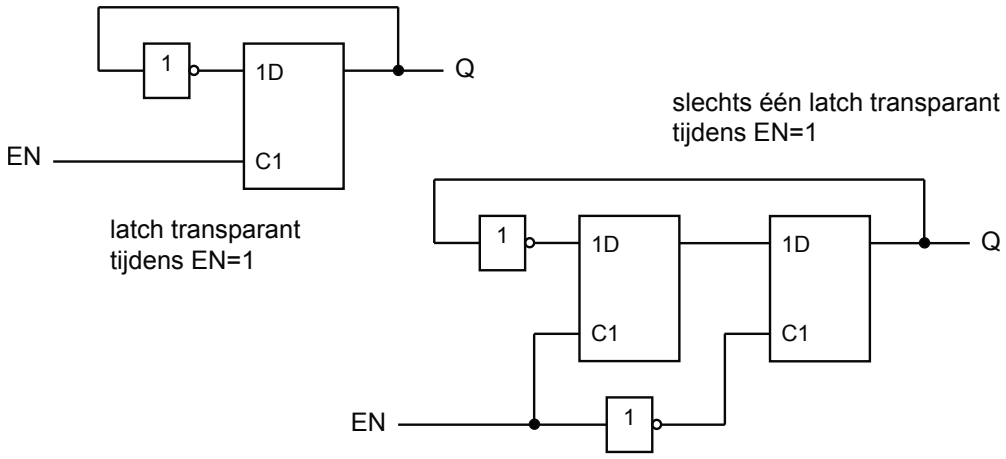
Uitwerking opgave 6.8.

De tweedeler is niet te maken met een (D-)latch. De uitgangswaarde van de latch moet geïnverteerd worden doorgegeven aan de ingang. Zodra de latch transparant is, verandert de uitgangswaarde als gevolg van de nieuwe ingangswaarde. Dit de inverse van de uitgangswaarde. Het geheel wordt instabiel en gaat *oscilleren*. Dit oscilleren volgt uit het transparant zijn van de latch. Door gebruik te maken van twee latches kan dit transparant zijn worden onderbroken door één van de latches gesloten te houden terwijl de andere geopend is. De enable-signalen zijn dus elkaars inverse. Zie figuur.

Uitwerking opgave 6.9.

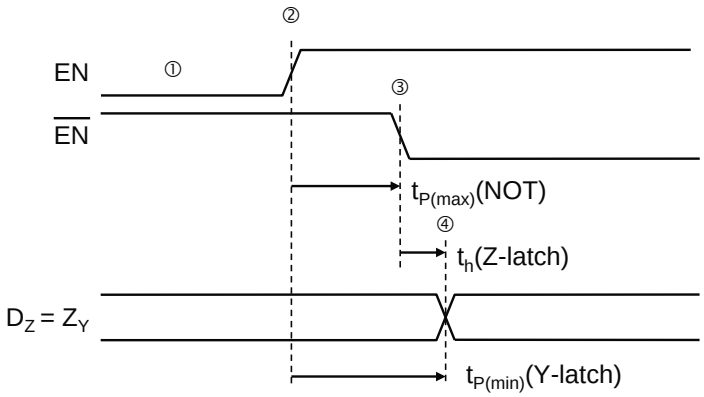
De timing-voorwaarde voor de correcte werking van de latch is

$$t_{P(max)}(\text{NOT}) + t_h(\text{Z-latch}) < t_{P(min)}(\text{Y-latch}) \tag{A.63}$$



Figuur A.53: *Poging tot realisatie van een tweedeler met gated D-latches.*

Dit is een doordenker. Eerst maar eens het timingdiagram tekenen. Zie onderstaande figuur.

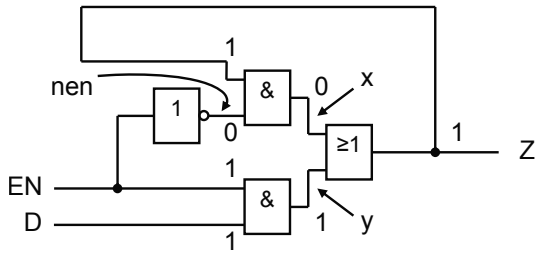


Figuur A.54: *Timing-voorwaarde van een D-flipflop.*

In de bovenstaande figuur zijn het enable-signaal en de geïnverteerde versie te zien. De geïnverteerde versie wordt gegenereerd door een NOT-poort (inverter) en dat kost enige tijd. Het enable-signaal wordt aan de Y-latch aangeboden. De Y-latch heeft een uitgang genaamd Z_Y. Het geïnverteerde enable-signaal wordt aan de Z-latch aangeboden. De Z-latch heeft een ingang D_Z. De uitgang van de Y-latch is direct gekoppeld aan de ingang van de Z-latch, dus $D_Z = Z_Y$. De situatie op moment (1) is dat de Y-latch gesloten is en de Z-latch open. De Y-latch neemt nog geen data over. Op moment (2) gaat de Y-latch open en zal nieuwe data gaan doorgeven, maar door de vertraging van de NOT-poort is de Z-latch óók nog open. Op moment (3) is het geïnverteerde enable-signaal laag (0) waardoor de Z-latch gaat sluiten (Y-latch staat nog steeds open). Dat is dus zeker na de maximale propagatietijd van de NOT-poort. De data moet nog enige tijd stabiel gehouden worden (holdtijd Z-latch) dus pas na moment (4) mag de data op de ingang van de Z-latch veranderen. Gelukkig heeft de Y-latch enige vertraging met het doorgeven van nieuwe data, dus mag de uitgang van de Y-latch pas veranderen (minimale propagatie) nadat holdtijd van de Z-latch is verstreken.

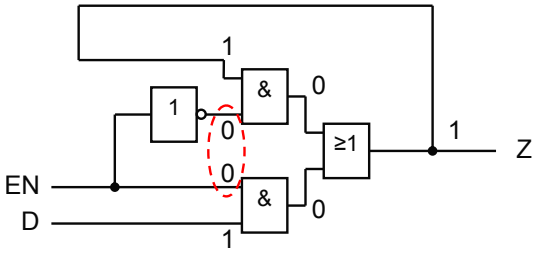
Uitwerking opgave 6.10.

Het probleem zit in de timing. Zie onderstaande figuur. Er zijn twee delen in de D-latch

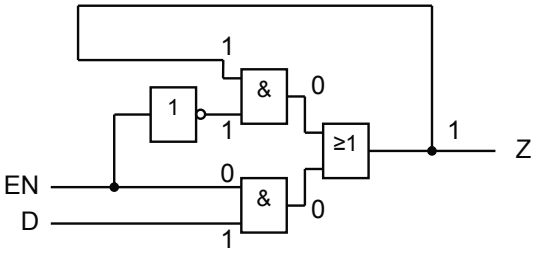


Kort *voordat* EN van 1 naar 0 gaat

te onderscheiden. De onderste AND-poort (uitgang y) zorgt er voor dat de waarde van D "in de lus" wordt gebracht als $EN = 1$, de bovenste AND-poort (uitgang x) is "gesloten". Als EN nu logisch 0 wordt, zal de onderste AND-poort "sluiten" en zorgt de bovenste AND-poort samen met de lus dat de waarde op Z behouden blijft. De inverse van EN wordt gemaakt met behulp van een inverter (*nen*) en dat kost enige tijd. *Er is dus een moment dat beide AND-poorten een logisch 0 op één van de ingangen aangeboden krijgen.* Zie onderstaande figuur *links*. Merk op dat de signaalverandering door de OR-poort nog

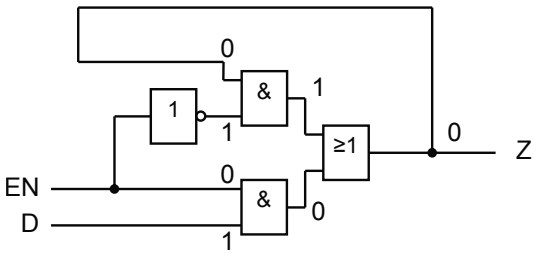


Kort *nadat* EN van 1 naar 0 gaat (signaalverandering door OR-poort nog niet gezien)



De inverter heeft nu de juiste waarde, maar de bovenste AND-poort heeft dat nog niet gezien.

niet is "gezien", wel door de inverter. Ondertussen is de inverter op de juiste waarde gekomen (logisch 1) nog voordat de OR-poort een logische 0 levert en samen met de logisch 1 van de uitgang levert dit weer een logische 1 op voor de bovenste AND-poort. Zie bovenstaande figuur *rechts*.

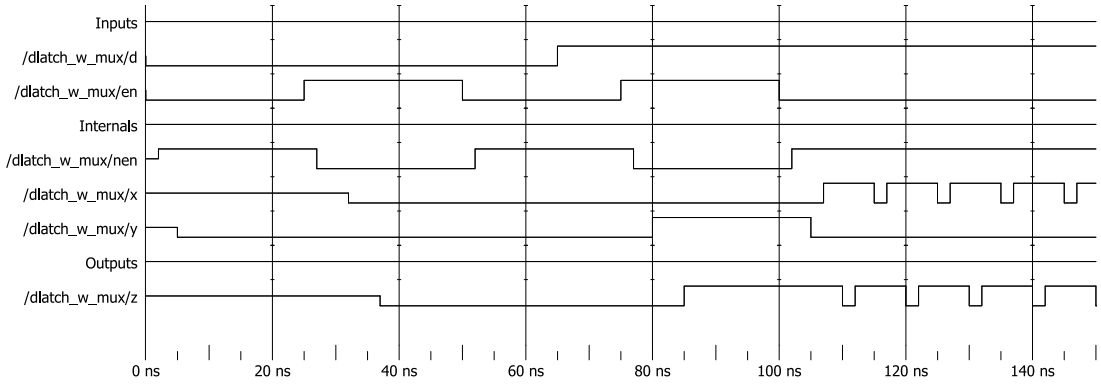


De OR-poort levert een logische 0 maar de bovenste AND-poort geeft ondertussen een logische 1 af, de OR-poort zal na een tijdje dus weer een logische 1 afgeven.

De OR-poort zal uiteindelijk een logische 0 leveren. Deze logische 0 wordt wel weer via de lus "in het systeem" gebracht. Maar de bovenste AND-poort geeft ondertussen een logische 1 af. Zie figuur hiernaast.

Nu zijn er een drie mogelijkheden: de uitgangswaarde blijft oscilleren tussen 0 en 1, de uitgang wordt na een tijd logisch 1 of logisch 0. Dit hangt helemaal af van de eigenschappen van de poorten en verbindingen (wires): propagatietijd is een belangrijke maar ook de *inertial rejection time*: als een aangeboden puls op een ingang te kort is, “ziet” een poort dit niet en de uitgang verandert dan ook niet.

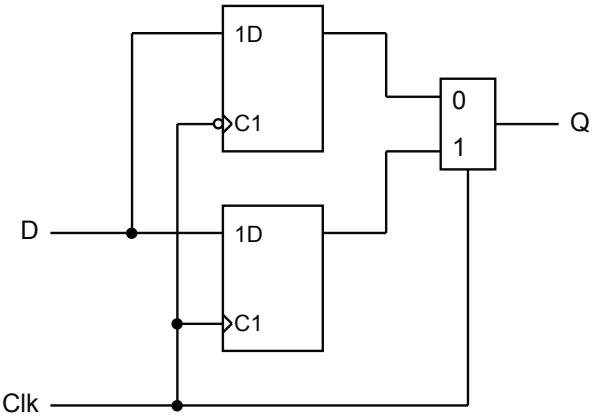
In onderstaande simulatie blijft de uitgang oscilleren. De vertragingstijden van de AND- en OR-poort zijn 5 ns, die van de inverter is 2 ns. Alle poorten hebben een *inertial rejection time* van 1 ns.



Figuur A.55: *Timing van een gated D-latch op basis van een multiplexer.*

Uitwerking opgave 6.11.

De meest eenvoudige oplossing is om twee flipflops die elk een flank voor hun rekening nemen te combineren. De uitgang is dan de waarde van de flipflop waar het laatst de flank is gepasseerd. Dit kan eenvoudig met een mutiplexer gerealiseerd worden. Zie onderstaande figuur.



Figuur A.56: *Double Edge Triggered flipflop op basis van D-flipflops en een multiplexer.*

Helaas heeft deze schakeling wel wat nadelen. Zo is de uitgang niet *single transition*. Bij het veranderen van het kloksignaal zal steeds de andere flipflop geactiveerd worden, maar ook de multiplexer zal de andere ingang selecteren. De uitgangen van de flipflops geven niet direct de nieuwe waarde af. Als de multiplexer snel is, zal deze eerst nog even de oude

waarde van de na de flank geselecteerde flipflop doorgeven. Beter is om een double edge-triggered D-flipflop geheel opnieuw te ontwerpen en dat is lastig. Een leuke realisatie met multiplexers is te vinden op http://atrak.usc.edu/~massoud/Papers/det-ff.pdf.

Uitwerking opgave 6.12.

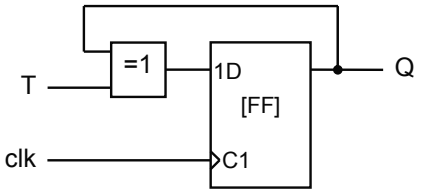
Het toggelen laat zich op eenvoudige wijze beschrijven. Hiervoor wordt uitgegaan van de D-flipflop. De nieuwe waarde is immers de inverse van de oude waarde: $Q^{n+1} = \overline{Q^n}$. Dit kan prima gedaan worden door een D-flipflop zijn eigen inverse waarde te laten inklokken.

Er wordt nu een signaal T geïntroduceerd waarmee dat toggelen te besturen is. De keuze is arbitrair maar een goede codering is als volgt:

Als T $= 0$ moet de flipflop zijn waarde vasthouden $\rightarrow Q^{n+1} = Q^n$

Als T $= 1$ moet de flipflop toggelen (inverse stand) $\rightarrow Q^{n+1} = \overline{Q^n}$

Dit is in z'n geheel te schrijven tot $Q^{n+1} = \overline{T^n} \cdot Q^n + T^n \cdot \overline{Q^n} = T \oplus Q^n$. Dus met behulp van een EXOR-poort is de T-flipflop op basis van een D-flipflop te realiseren. Zie onderstaande figuur.



Figuur A.57: *T-flipflop op basis van een D-flipflop en een EXOR-poort.*

Uitwerking opgave 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. Eerst maar eens de actietabel van de JK-flipflop weergeven:

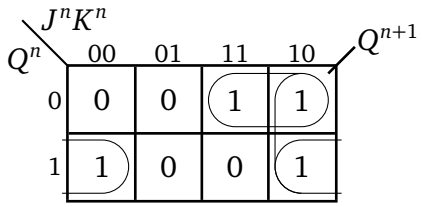
J^n	K^n	Q^{n+1}	actie
0	0	Q^n	onthouden
0	1	0	reset
1	0	1	set
1	1	$\overline{Q^n}$	inverse

Hiernaast de actietabel. Merk op dat bij de laatste mogelijkheid ($JK = 11$) de inverse waarde wordt "ingeklokt". We schrijven de actietabel om naar een waarheidstabel.

J^n	K^n	Q^n	Q^{n+1}	actie
0	0	0	0	onthouden
0	0	1	1	onthouden
0	1	0	0	reset
0	1	1	0	reset
1	0	0	1	set
1	0	1	1	set
1	1	0	1	inverse
1	1	1	0	inverse

Hiernaast is de waarheidstabel gegeven van de JK-flipflop. De eerste zes combinaties zijn identiek aan die van de SR-latch of (de niet besproken) SR-flipflop. De laatste combinatie wordt nu gebruikt om de inverse stand van de flipflop in te klokken. De flipflop "toggelt" dus. Bij een latch geeft dat problemen (oscillatie) maar een flipflop is flankgestuurd.

Vervolgens kan een Karnaughdiagram worden opgesteld en uitwerkt.



De vereenvoudigde functie is

$$Q^{n+1} = J^n \cdot \overline{Q^n} + \overline{K^n} \cdot Q^n$$

De functie is dus zoals in de vraagstelling is gegeven.

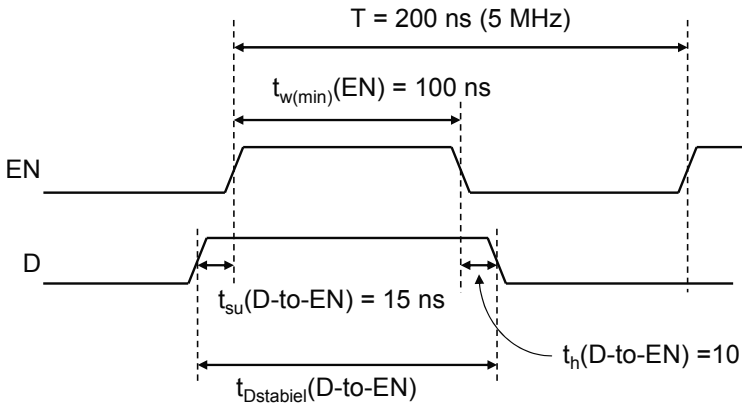
Uitwerking opgave 6.14.

Van een 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%. Wat is de tijd dat D stabiel moet blijven? Hiervoor moet een timingdiagram getekend worden. Daarin moet natuurlijk het kloksignaal getekend worden. Merk op dat het kloksignaal het enablesignaal is (EN = klok). Van het kloksignaal is gegeven een frequentie van 5 MHz, de periodeduur (of periodetijd) is dan 200 ns. Verder is de duty cycle 50% wat inhoudt dat de klok gedurende 100 ns (50% van de periodeduur) logisch '1' is (dat wordt ook wel klok-hoog genoemd). Gedurende die tijd moet D zeker stabiel blijven. Daarnaast zijn nog een setuptijd en een holdtijd gegeven. D moet stabiel blijven van de setuptijd (voordat EN logisch 1 wordt) tot na de holdtijd (nadat EN logisch 0 is geworden). Zie de figuur. De totale tijd dat D stabiel moet zijn is:

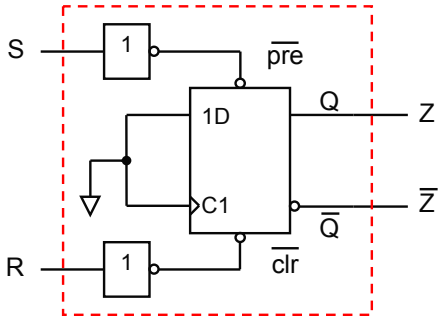
$$\begin{aligned} t_{Dstabiel}\text{(D-to-EN)} &= t_{su}\text{(D-to-EN)} + t_{w(min)}\text{(EN)} + t_h\text{(D-to-EN)} \\ &= 15 + 100 + 10 = 125 \text{ ns} \end{aligned}$$

Uitwerking opgave 6.15.

Zie onderstaande figuur. De D- en klokingang zijn met een logische 0 verbonden waardoor de flipflop nooit data zal inklokken. De asynchrone preset- en clear-ingangen zijn verbonden (via een inverter) met resp. de S(et)- en R(eset)-signalen. Deze configuratie wordt veel gebruikt in een FPGA. Er wordt een minimum aan logische componenten gebruikt en de timing is veel beter in de hand te houden dan wanneer de schakeling met combinatoriek zou worden gerealiseerd. Overigens is het gebruik van latches niet aan te bevelen in een complex digitaal systeem.



Figuur A.58: *Timing van de gated D-latch*

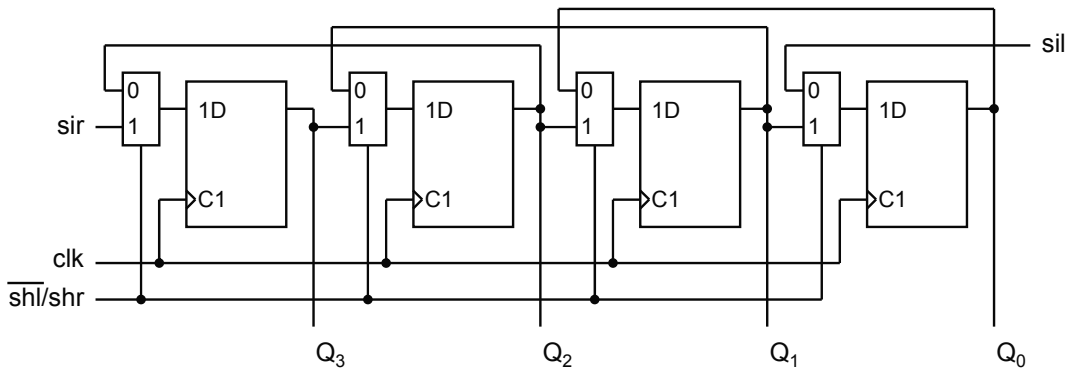


Figuur A.59: *Een SR-lacth op basis van een D-flipflop met asynchrone set en reset.*

Uitwerking opgave 6.16.

Het schuifregister moet twee kanten op kunnen schuiven. Hiervoor is een stuursignaal nodig. We noemen het signaal $\overline{shl}/shr$ (deze naamgeving komt vaker voor, denk hierbij aan het $R/\overline{W}$-signaal bij processoren). Als het signaal logisch 0 is, zal het schuifregister linksom schuiven, als het signaal logisch 1 is, zal het schuifregister rechtsom schuiven.

Tijdens het schuiven wordt en een databit ingeschoven via één van de ingangen sir (serial input right) en sil (serial input left). Natuurlijk verdwijnt er ook een bit, dit bit wordt "eruit geschoven". Verder worden multiplexers gebruikt om de te schuiven bits in de goede flipflops te krijgen.



Figuur A.60: *Een links-rechts schuifregister.*

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] 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. 4).