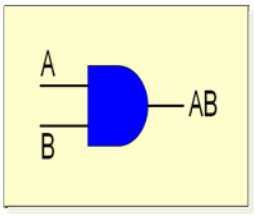
Logische poorten

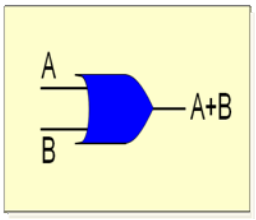
and

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1



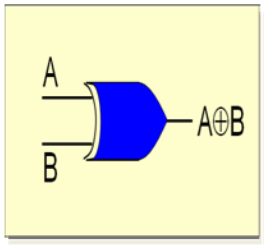
or

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



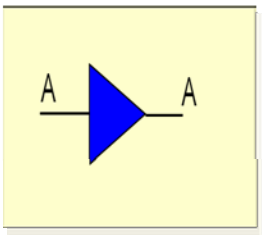
Xor (=EOF)

A	B	A⊖B
0	0	0
0	1	1
1	0	1
1	1	0



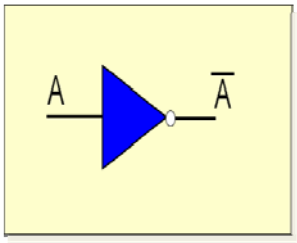
buffer **(signaal versterken over lange afstand)**

A	A
0	0
1	1



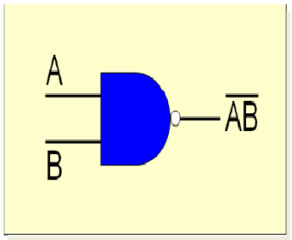
Invertor (NOT)

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



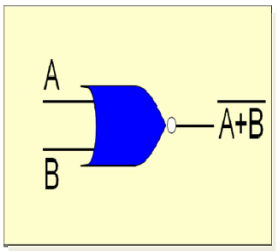
nand

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



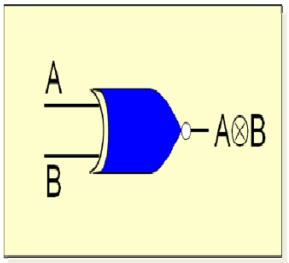
nor

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

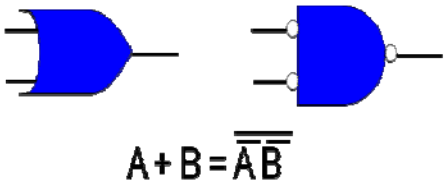


xnor

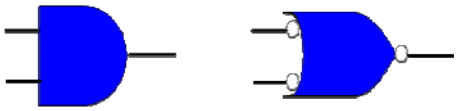
A	B	A⊗B
0	0	1
0	1	0
1	0	0
1	1	1



De Morgan



$$A + B = \overline{\overline{A}\,\overline{B}}$$



$$AB = \overline{\overline{A} + \overline{B}}$$

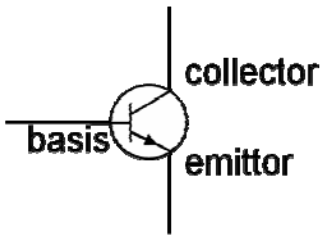
Boolese Algebra

Commutativiteit	$AB = BA$
Distributiviteit	$A(B+C) = AB+AC$
Neutraal element	$1A = A$
Complement	$A\overline{A} = 0$
Nuleigenschap	$0A = 0$
Idempotentie	$AA = A$
Associativiteit	$A(BC) = (AB)C$
Dubbele negatie	$\overline{\overline{A}} = A$
De Morgan	$\overline{AB} = \overline{A} + \overline{B}$
Consensus	$AB + \overline{A}C + BC = AB + \overline{A}C$
Absorptie	$A(A+B) = A$

Commutativiteit	$A+B = B+A$
Distributiviteit	$A+(BC) = (A+B)(A+C)$
Neutraal element	$0+A = A$
Complement	$A+\overline{A} = 1$
Eéneigenschap	$1+A = 1$
Idempotentie	$A+A = A$
Associativiteit	$A+(B+C) = (A+B)+C$
Dubbele negatie	
De Morgan	$\overline{A+B} = \overline{A}\,\overline{B}$
Consensus	$(A+B)(\overline{A}+C)(B+C) = (A+B)(\overline{A}+C)$
Absorptie	$A+(AB) = A$

Transistorniveau

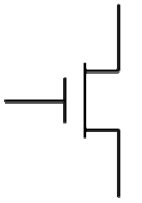
Transistor



halfgeleiderschakelaar

- Het is een schakelaar zoals het relais, maar wordt in veel kleinere oppervlakte geproduceerd, en bevat geen mechanisch, bewegende delen.
- Wordt gemaakt in halfgeleidertechnologie.
- Indien er stroom loopt door de basis, kan er stroom vloeien tussen **connector en emittor (→ schakelaar)**

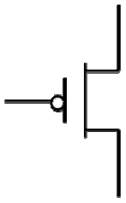
MOS transistor



NMOS

Geleidt indien input hoog

Gebruikt om output laag te brengen



PMOS

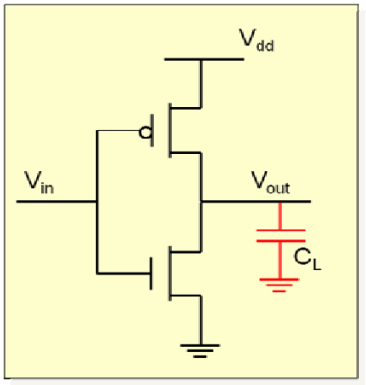
Geleidt indien input laag

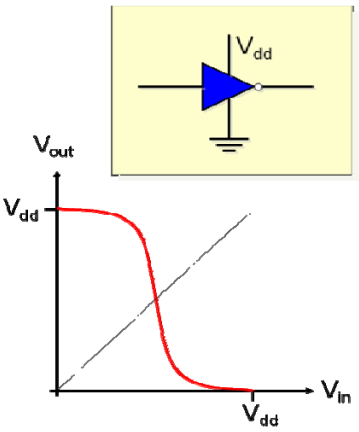
Gebruikt om output hoog te brengen

In alle chips worden tegenwoordig MOSFET transistors gebruikt (metal oxide semiconductor field-effect transistor). Hiervan bestaan twee varianten: NMOS en PMOS. NMOS blijkt bijzonder geschikt voor om een verbinding met de massa (aarding) tot stand te brengen, en PMOS voor een verbinding met de voedingsspanning. Meestal zullen ze dan ook in paren worden gebruikt. Indien beide op dezelfde chip voorkomen spreekt men van **CMOS**.

Voorbeelden

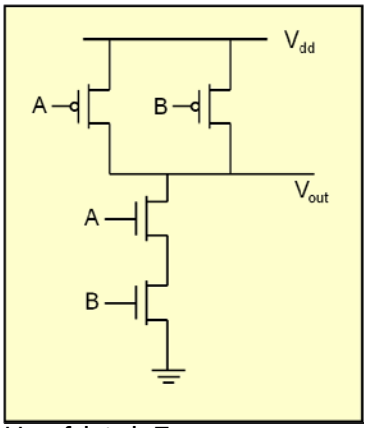
1/ Invertorpoort





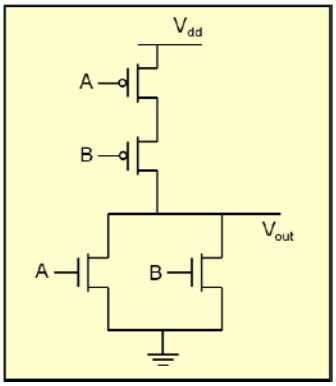
Aan de uitgang zien we: C_L (=capaciteit). Deze is in werkelijkheid niet aanwezig, maar modelleert de capaciteit van alle uitgaande bedrading. De ingang van een transistor in CMOS verbruikt *geen* stroom. De enige stromen die vloeien zijn om de parasitaire capaciteiten van de bedrading te laden en te ontladen. Eenmaal in evenwicht, stopt ook deze stroom. In rust verbruikt deze schakeling dus bijna geen stroom, bij omschakelen wel (hoe hoger de klokfrequentie, hoe meer er omgeschakeld wordt).

2/ Nand-poort



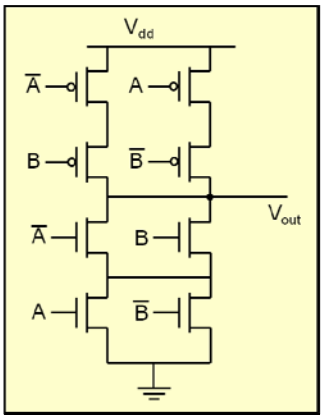
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

3/ Nor-poort



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

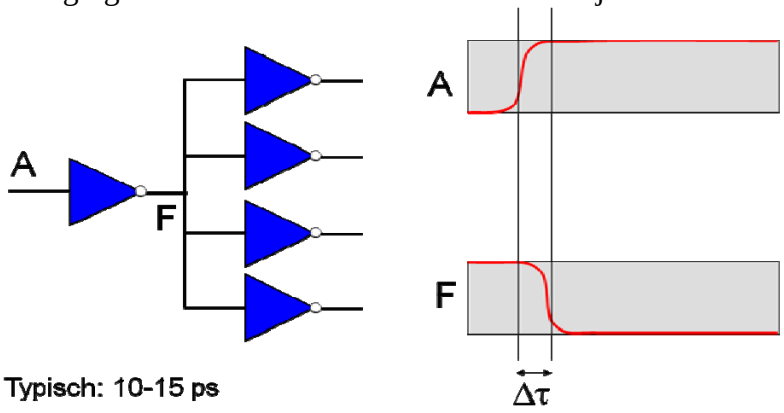
4/ Xnor-poort



A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

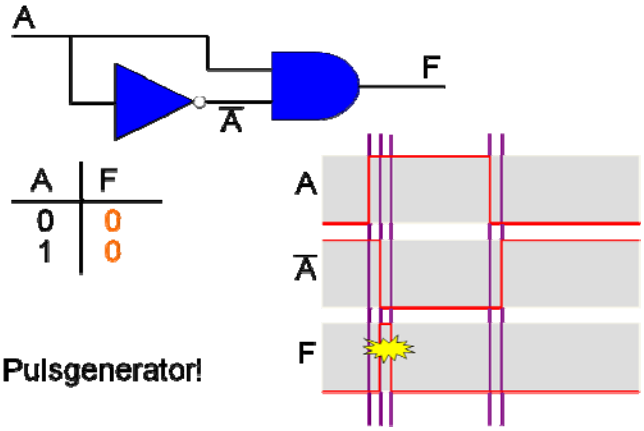
Tijdsgedrag van poortjes

Er is altijd een zekere tijd nodig vooraleer de uitgang van een poort de correcte waarde aanneemt, na een verandering aan de ingang. Dit noemen we **poortvertraging** of **propagatietijd**. Dit kan soms onaangename gevolgen hebben!
Een gangbare manier om een poortvertraging op te geven is FO4 **(fan out of 4)**. Dit is de vertraging die een invertor ondervindt indien hij vier identieke invertoren dient aan te sturen.

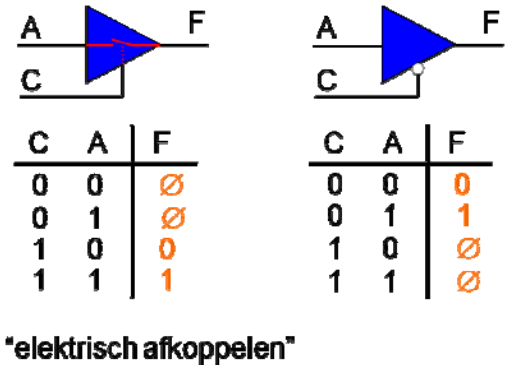


Typisch: 10-15 ps
Klokperiode > 12-16 FO4

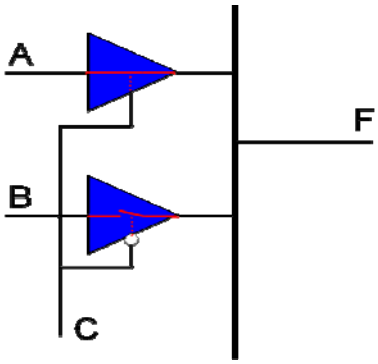
Een mogelijk gevaar van propagatietijden is de **glitch**. Dit is een ongewenste toestand of uitgang, veroorzaakt door het tijdsgedrag van poortjes. Hieronder doet zich een glitch voor wanneer A wisselt van 0 naar 1.



Tri-state buffer



Een tri-state buffer gedraagt zich wel als een gewone buffer, maar er is een extra controle ingang aanwezig die toelaat de buffer af te koppelen. Er zijn dus drie uitgangswaarden: 0, 1 of afgekoppeld. In afgekoppelde toestand is er geen uitgangswaarde omdat de uitgang **elektrisch afgekoppeld** is. We noemen de uitgang ook **hoog-impedant**. Ze bestaat in twee vormen: met **directe controle** (links) en met **geïnverteerde controle** (rechts).



We gebruiken tri-state buffers om een aantal logische poorten aan te sluiten op een gemeenschappelijke lijn (bus), zonder kortsluiting te veroorzaken, zolang er slechts 1 buffer tegelijk is aangeschakeld. Dit gebeurt door een geïnverteerd controlesignaal!

Connecties

Als verschillende verbindingen op een tekening door en over elkaar lopen, moeten we kunnen aanduiden welke kruisen, en welke zijn geconnecteerd. Met een T-kruising, of een bolletje zijn ze connected.

Productie van chips

Chips of halfgeleidercircuits worden geproduceerd op een grote, ronde siliciumplaat = **wafer**, typisch 20-30 cm diameter. Per plaat kunnen enkele tot vele tientallen circuits worden gefabriceerd (1 circuit = **die**). De chip bestaat uit vele miljoenen transistors die onderling verbonden worden door middel van metaalbaantjes in verschillende niveaus.
Tussen het kanaal waar de stroom doorloopt, en de “ingang = gate” van de transistor, bevindt zich een **oxidelaag**.

Wet van Moore

= Het aantal transistors per chip verdubbelt nagenoeg iedere 2 jaar. De geheugenchips liggen voor op de processor omwille van hun eenvoudigere structuur.

Combinatorische schakelingen

Realisatie van Boolese functies

Een meer ingewikkelde Boolese functie kan worden geïmplementeerd via een **som van producten**. Dit is het OR'en van een aantal AND resultaten.
Voor elke rij waar het resultaat een 1 bevat, stelt men de **minterm** op, dit is een productterm die elke variabele precies 1 keer bevat. De som van alle mintermen kan in de logica worden geïmplementeerd aan de hand van NOT-poorten, multi-input AND-poorten, en multi-input OR-poorten.

Je kan evengoed een Boolese functie realiseren aan de hand van een **product van sommen**. Men noteert dan de mintermen voor die rijen die 0 hebben als resultaat, en inverteert de ganse som van producten-rij (via De Morgan)

Minimalisatie

Som van producten of product van sommen kan vaak onnodig complex zijn. Dit proces gaat op zoek naar de eenvoudigste implementatievorm.

Complexiteit van een realisatie

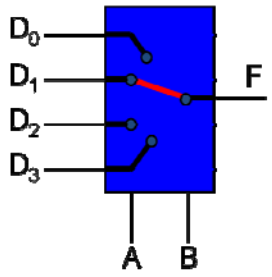
Kan kwantitatief uitgedrukt worden door een telling te maken van het totaal aantal poorten, het totaal aantal inputs, de maximale fan-out (maximaal aantal uitgaande verbindingen), en maximale fan-in (macimaal aantal inkomende verbindingen).

Computationele compleetheid

De minimale poortjes nodig om een willekeurige combinatorische functie te implementeren. **{AND,OR,NOT} en {NAND} en {NOR}** zijn computationeel compleet.

Digitale componenten

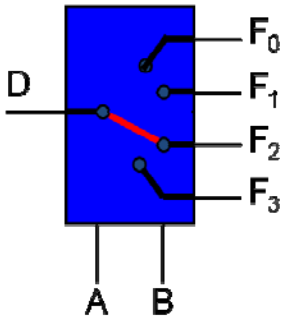
<u>1/ Multiplexer (MUX)</u>



A	B	F
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Zet een aantal ingangen om tot 1 enkele uitgang. Men kan hiermee Boolese functies implementeren door inputs (A,B) aan te leggen aan de controlelijnen en de resultaten uit de waarheidstabel als constanten aan te leggen aan de data-ingangen.

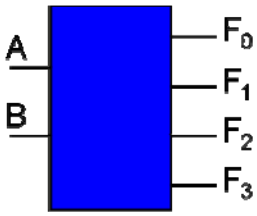
2/ Demultiplexer (DEMUX)



A	B	F_0	F_1	F_2	F_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

Eén data-ingang wordt doorgegeven aan één van de data-uitgangen. Toepassing: zenden van data van 1 bron naar 1 bestemming te kiezen uit een groep van mogelijke bestemmingen.

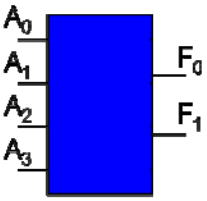
3/ Decoder



A	B	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Vertaalt een code in een spatiale locatie. Welke uitgang op 1 staat, wordt bepaald door de code aan de ingang. Kan gebruikt worden om andere circuits aan te sturen.

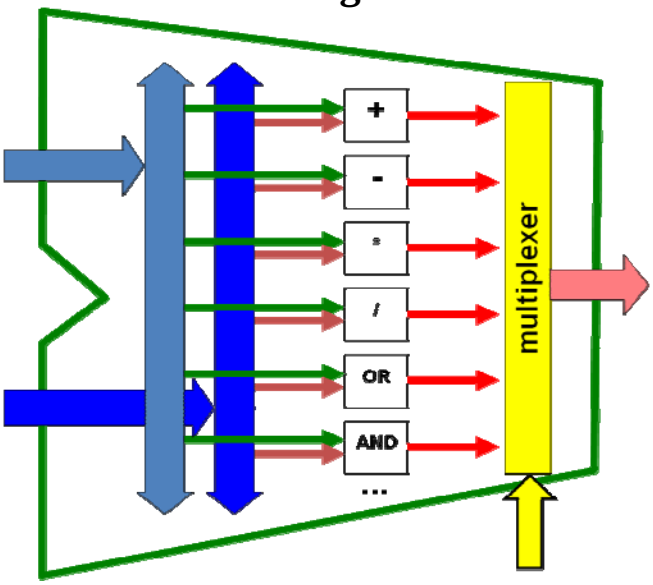
4/ Prioriteitscodeerder



A_0	A_1	A_2	A_3	F_0	F_1
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

Legt een prioriteit op aan de ingangen. De uitgang geeft de binaire waarde weer van de ingang met de hoogste prioriteit die is aangeschakeld. A0 heeft hogere prioriteit dan A1 enz.

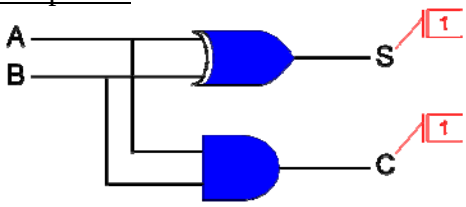
De ALU-schakelingen



Een ALU bestaat uit een parallelle berekening van de ALU-functies en dan de selectie van het gewenste resultaat door de MUX op het einde.

Optellers en aftrekkers

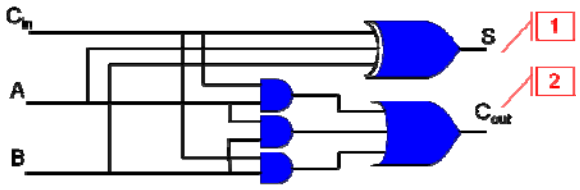
1/ Halve opteller



Berekent de som van 2 bits, en stelt het resultaat voor door 2 bits: som en carry. Zowel som en overdracht worden gegenereerd na 1 poortvertraging.

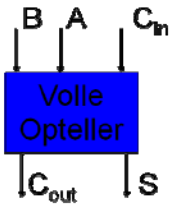
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

2/ Volle opteller

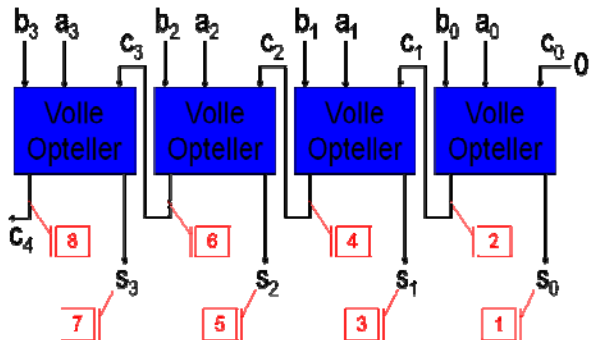


Maakt de som van 3 bits: 2 operandi, en 1 inkomend carry bit. Het resultaat duurt 1 poortvertraging, de overdacht 2.

C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

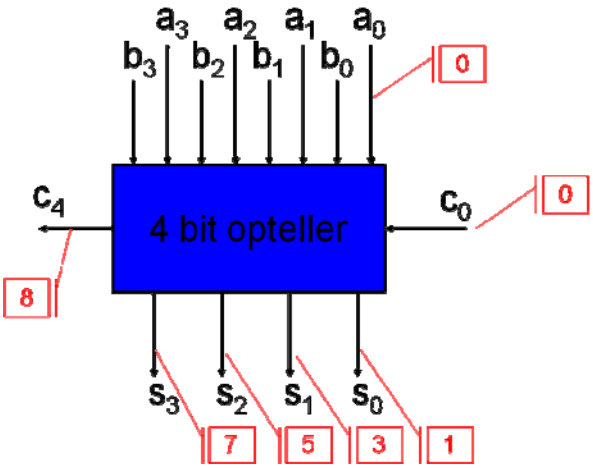


3/ Doorsijpelopteller



m-bit optelling realiseren door m Full Adders in serie te zetten. De som zal worden gegenereerd na 2*m – 1 poortvertragingen, en de overdracht na 2*m vertragingen.
Een methode om carrypropagatie te versnellen is de C_{out} herschrijven aan de hand van minimalisatie: $C_{out} = G + P\,C_{in}$ met G=AB en P = A+B. (slide 72)

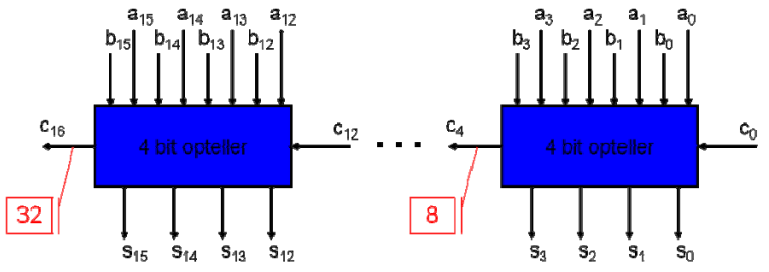
4/ 4-bit opteller

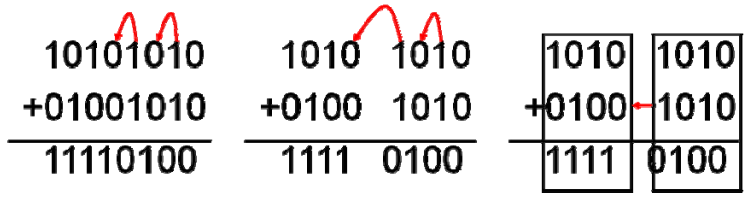


Uitgangen zijn de 4 sommatiebits en de overdracht bij de meest beduidende bit. Kan worden opgebouwd zoals hierboven.
Ook hier kan carry-propagatie worden versneld door minimalisatie.

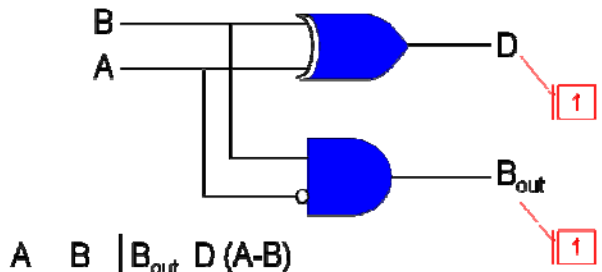
$$C_4 = G_3 + P_3 C_3$$
$$C_3 = G_2 + P_2 C_2$$
$$C_2 = G_1 + P_1 C_1$$
$$C_1 = G_0 + P_0 C_0$$

5/ Cascade van 4-bit optellers





6/ Halve aftrekker

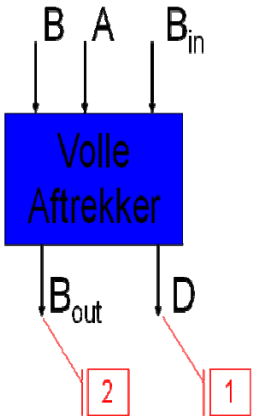


A	B	B_{out}	D (A-B)
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

A-B wordt berekend, en het resultaat wordt voorgesteld door Difference en Borrow.

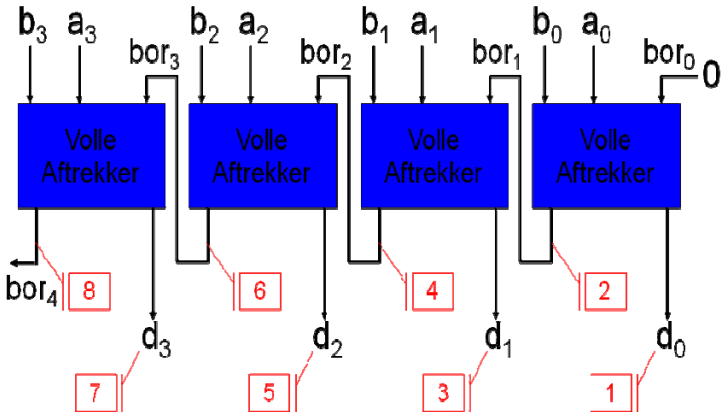
7/ Volle aftrekker

A	B	B_{in}	B_{out}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

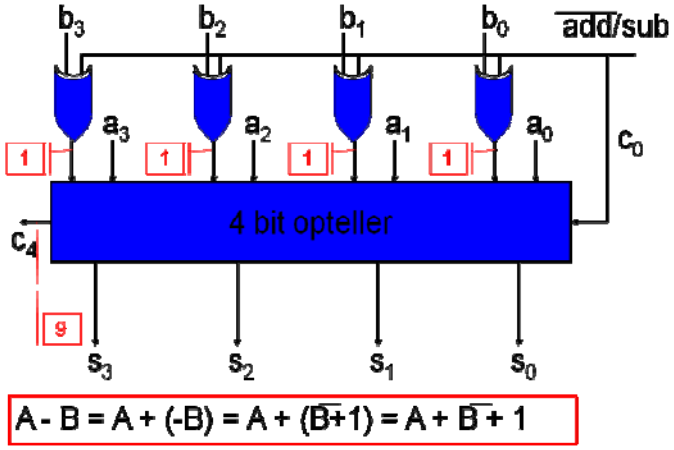


A-B-B_{in}, waarbij <B_{out}:D> het 2bits 2 complement is van A-B-B_{in}.

8/ Doorsijpelaftrekker

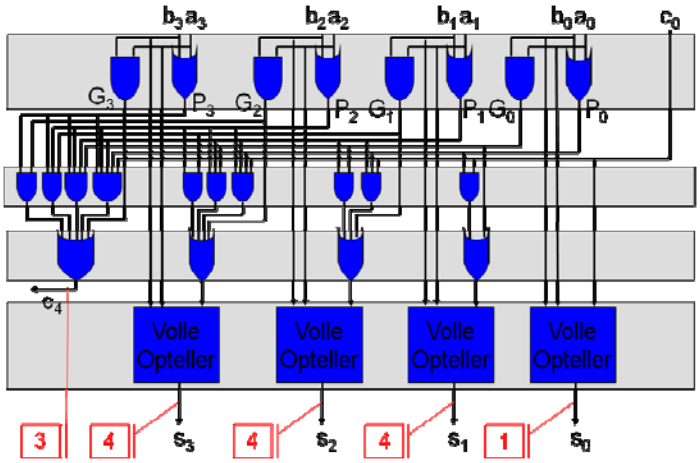


9/ Opteller/aftrekker



Uiteraard kan men A-B ook berekenen als A+(-B). De niet-add/sub lijn geeft aan indien het om een optelling of aftrekking gaat. De overige ingangen zijn A en B of (-B). Als de lijn=1, zullen de XOR poorten en c0=1 ervoor zorgen dat –B berekend wordt (invertering van alle bits + 1 (c0)). Indien lijn = 0, wordt A+B berekend.

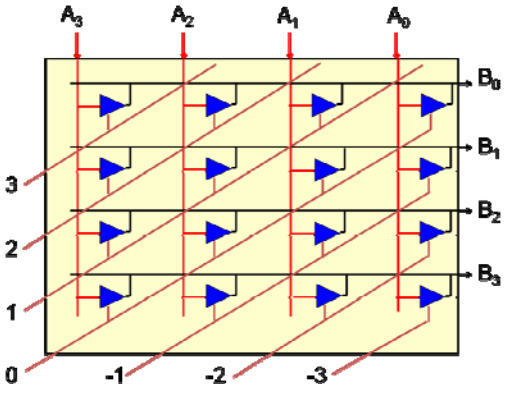
10/ Carry Lookahead opteller



De aanwezige volle optellers berekenen hier geen carry bit meer, omdat uitgaande/inkomende carry bits rechtstreeks worden berekend uit de operandi. Dit is de implementatie van minimalisatie bij doorsijpeloptellers...

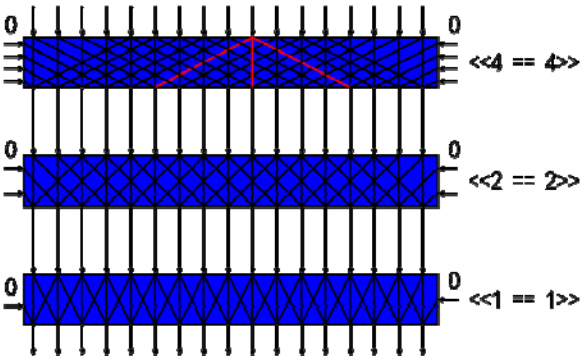
Verschuivers

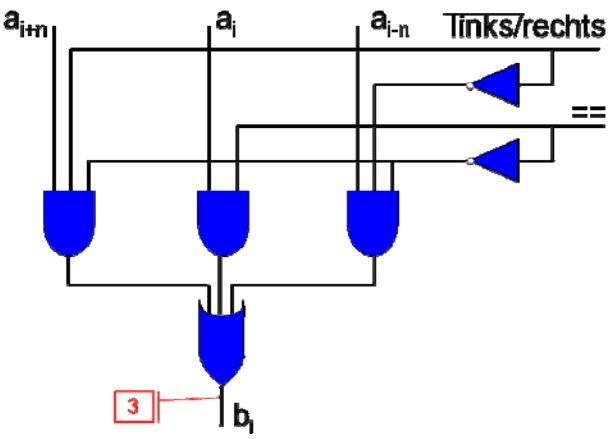
1/ Barrel shifter



In staat om in 1 poortvertraging een bitpatroon over n willekeurige bits voorwaarts of achterwaarts te verschuiven. Hiertoe moeten de correcte tri-state buffers worden aangestuurd. Voor kleine bitpatronen te verschuiven werkt dit goed, maar voor grote loopt het aantal buffers en de fan-out (van A_i) zeer sterk op. Er is echter een **verborgen kost!!** Er zal een decoder nodig zijn om de signalen 3,2,1,0,-1,-2,-3 te generen uitgaande van de 2-complementsvoorstelling van de verschuiving. (→ poortvertragingen!)

2/ Logaritmische shifter





Betere oplossing dan de barrel shifter. Kan een woord (32 bit) aan de ingang verschuiven over een willekeurige afstand afhankelijk van de ingangen.
In elke laag/stap van de shifter wordt een **log shifter cel** gebruikt om per bitpositie de nieuwe bitwaarde te berekenen. Dit gebeurt door combinatie van drie bits (zie tekening) Dit zijn dus de bits van de positie zelf, en de bits van de positie n bits naar links en n bits naar rechts. N zal dan 1,2,4,... zijn.
Daarnaast zijn er nog controlesignalen die aangeven of er geschoven wordt of niet, en in welke richting.

Vermenigvuldigers

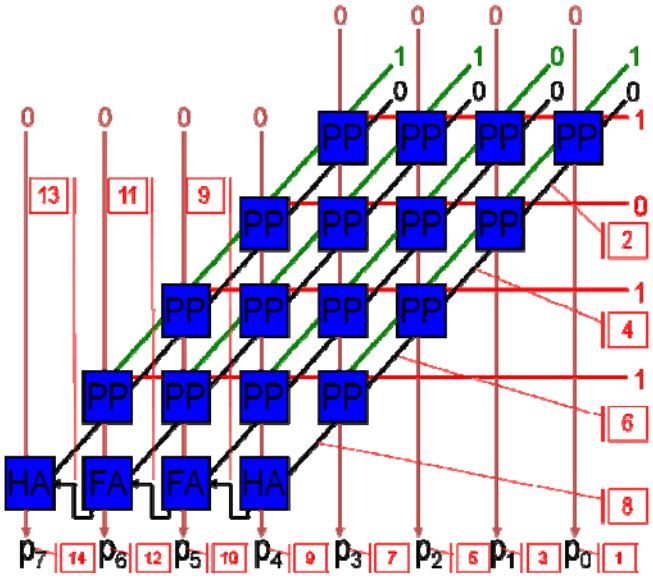
1/ Werkwijze
Hoe het in zijn werk gaat: slides 81-97!

2/ Algoritme van Booth
Indien we in de plaats van een 4 bit opteller, een opteller/aftrekker gebruiken, dan kunnen we het product in sommige gevallen aanzienlijk versnellen door gebruik te maken van het Algoritme van Booth. Indien de vermenigvuldiger uit een groep van 1 bitjes bestaat, kunnen deze worden gegroepeerd en uitgedrukt als een verschil. Men moet dus niet alleen kijken naar het bit waarmee men moet vermenigvuldigen, maar ook naar het bit waarmee men net vermenigvuldigd heeft. Wanneer er een overgang is van 0 naar 1, volgt er een rij 1tjes (soms 1, soms meerdere). Op dit ogenblik moet men het verschil maken ipv de som. Wanneer de rij 1tjes overschakelt naar een 0, moet men de som maken.

M x 00011110
= M x (16 + 8 + 4 + 2)
= M x 30
= M x (32 – 2)

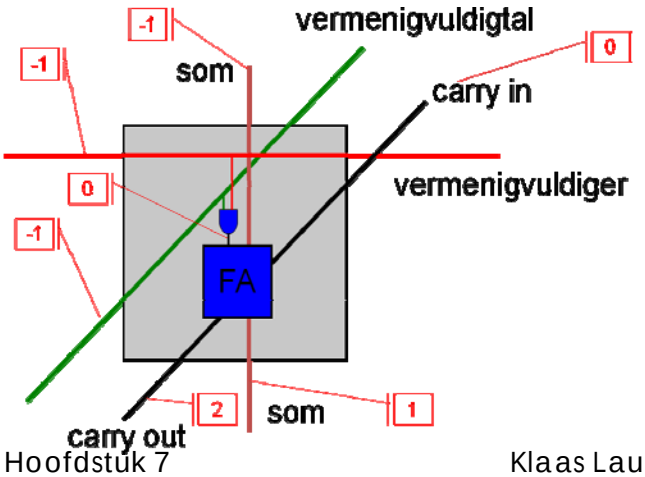
Men moet echter wel vertrekken van een 0 en een leidende 0 toevoegen om het algoritme goed te initialiseren.

3/ Carry-save vermenigvuldiger



In het groen staat het vermenigvuldigtal, in het rood (rechts) de vermenigvuldiger. Eerst wordt hier het product gemaakt van elke bit in vermenigvuldigtal, en elke bit in vermenigvuldiger, en dan worden de partiële productelementen opgeteld. Men noemt dit carry save, omdat de carry naar **links onder** ipv naar links wordt doorgegeven → doorgegeven naar de volgende partiële som. Deze implementatie kan nog worden versneld door de carry look-ahead toe te passen!

Hieronder wordt een partieel product element in detail getoond.



Vermenigvuldigen van 1 bit van vermenigvuldiger en 1 bit van vermenigvuldigtal. Het resultaat wordt opgeteld bij de andere bits in dezelfde kolom, en bij de inkomende carry. Een negatieve poortvertraging wordt gebruikt om aan te geven dat het resultaat reeds 1 poortvertraging aanwezig is.

Delers

1/ Restoring deling
Werkwijze: zie slides 105-128

2/ Non restoring deling
Het idee is dat indien (Deeltal-Deler) < 0 is, het resultaat van
(Deeltal-Deler + Deler) * 2 - Deler = Deeltal * 2 - Deler
wat hetzelfde is als:
(Deeltal-Deler) * 2 + Deler = Deeltal * 2 - Deler

Door te onthouden dat het vorige verschil < 0 was, en in een volgende stap de optelling uit te voeren, en dan de test te doen, bereikt men hetzelfde resultaat, zij het met minder operaties.
Men noemt dit een non-restoring deler.
Merk op dat de hardware nodig om te delen en te vermenigvuldigen nagenoeg dezelfde is.

3/ Iteratieve deling
De berekening van 1/b kan worden benaderd door een iteratief proces.

Benadering van 1/b: $x_{i+1} = x_i(2-x_i b)$

Zodoende zijn enkel vermenigvuldiging en aftrekking nodig!