Kapittel 5

Porter og logiske funksjoner

Innhold

5.1 Læringsmål . . . 106
5.2 Det begynner med en bryter . . . 107
5.3 Portene . . . 113
5.4 Sannhetsverditabellen . . . 117
5.5 Fra tabell til uttrykk . . . 120
5.6 Forenkling . . . 122
5.7 Kretsen som legger sammen . . . 126
5.8 Mentekjeden . . . 131
5.9 Kretser som velger . . . 136
5.10 Kretsen som husker . . . 139
5.11 Oppsummering . . . 139
5.12 Videre lesning . . . 141
5.13 Refleksjon . . . 141
5.14 Selv-test . . . 142

Et sted i koden din står det en linje som ligner på denne:

```
if alder >= 18 and har_billett:
    slipp_inn()
```

Du har skrevet noe slikt selv. Tre små ord gjør hele arbeidet i slike linjer: **AND**, **OR** og **NOT**. De føles like selvfølgelige som komma og punktum, og du bruker dem riktig lenge før noen har forklart deg hvorfor.

Note-to-myself

I Python skrives de med små bokstaver, fordi språket krever det. I resten av denne boka skriver vi dem med **store**: AND, OR, NOT. Det er for å skille den logiske operasjonen fra den konkrete kodebiten, og fordi det er slik de står på portsymbolene og i all faglitteratur.

Så kan vi stille et spørsmål som er litt ubehagelig å tenke på. Hvordan vet maskinen hva AND betyr?

Den er laget av sand, metall og strøm. Det sitter ingen inne i den og leser setningen din. Likevel treffer den riktig hver eneste gang, milliarder av ganger i sekundet, uten noen gang å nøle.

Da må AND være en *ting* der nede. Noe du kunne pekt på, hvis du bare hadde et godt nok mikroskop og visste hvor du skulle lete.

Den tingen er det vi skal finne i dette kapitlet. Den er mindre og langt dummere enn du tror.

Tankevekker

Alt maskinen gjør, er satt sammen av deler som kan svare på ett eneste slags spørsmål: ja eller nei, om ett eller to signaler.
Ingen av dem vet hva et tall er. Ingen av dem vet hva addisjon er. Likevel legger de sammen — og før du er ferdig med dette kapitlet, har du bygget kretsen som gjør det.

Vi begynner et sted alle har vært: en bryter og en lampe. Derfra går vi ett skritt om gangen, og hvert skritt er kort nok til at du kan ta det i hodet.

Du trenger ingen matematikk for å lese videre. Ingen formler du skulle hatt fra før, ingen regneregler du burde ha pugget. Alt som kreves, er at du godtar at noe er enten sant eller usant. Har du hatt følelsen av at tidligere kapitler forutsatte noe du ikke hadde, slipper du den følelsen her.

Vi kommer riktignok til å lage oss en liten skrivemåte underveis, med noen få tegn som sparer plass. Den lærer du her, på stedet, og den er kortere enn alfabetet.

Til gjengjeld er stoffet *nytt*. Det ligner ikke på noe annet i faget, og de aller fleste trenger å se det to ganger før det setter seg. Det er helt normalt, og det sier ingenting om hvor flink du er. Senk skuldrene, og ta det i din egen fart. Ha gjerne et ark ved siden av deg og tegn med.

Vi kommer dessuten hit med noe til gode.

Kapittel 4 stoppet tre ganger og pekte nedover. Vi åpnet ikke ALU-en. Vi åpnet ikke dekoderen. Vi sa at et register kan huske, uten å si hvordan noe kan huske så lenge strømmen står på. Her er de tre svarene, og det fine er at det er det samme svaret alle tre gangene.

Innen siste side har du bygget en krets som legger sammen to tall. Ikke sett på en — bygget en, ut fra en tabell du selv har satt opp. Du kommer også til å se hvorfor en addisjon tar lengre vei gjennom maskinen enn en sammenligning, og hvor to av flaggene fra forrige kapittel fysisk kommer fra. Det viser seg å være to ledninger og én port.

Og én grunn til at dette er verdt noe også utenfor dette faget.

Telefonen i lomma di har noen milliarder av disse små delene. Hver eneste av dem tar plass, trekker strøm og blir varm. Bruker du dobbelt så mange deler som nødvendig, betaler du for det i alle tre valutaene samtidig — og på en brikke med milliarder av dem blir det fort merkbart på både varme og batteri.

Store deler av dette kapitlet handler derfor om å ikke sløse. Det er den samme disiplinen du bruker når du skriver en betingelse som ikke sjekker det samme to ganger.

Vi begynner med lampa.

5.1 Læringsmål

Etter å ha arbeidet med dette kapitlet skal du kunne:

1. **Forklare forskjellen på en funksjon og en krets.** Dette er det ene skillet alt det andre i kapitlet hviler på.

2. **Kjenne igjen de sju portene og si hva hver av dem gjør.** Du skal kunne gå fra symbol til tabell til uttrykk og tilbake, uten å pugge noe av det.
3. **Sette opp en sannhetsverditabell for en krets, og lese en krets begge veier.** Framlengs fra inngangene, og baklengs fra utgangen. De to måtene oppdager ulike ting.
4. **Skrive funksjonsuttrykket som hører til en tabell.** Dette er den ene metoden alt det andre i kapitlet hviler på.
5. **Forenkle et boolsk uttrykk med et lite sett regler.** Og vite når du er ferdig, som er vanskeligere enn det høres ut.
6. **Forklare hvordan en krets legger sammen to bit.** Halvadderen og fulladderen, utledet fra tabellen og ikke bare vist fram.
7. **Gjøre rede for hvorfor mentekjeden koster tid.** Her ligger svaret på hvorfor en addisjon kan bruke lengre vei gjennom kretsen enn en ELLER, og hvor to av flaggene fra forrige kapittel fysisk kommer fra.
8. **Bygge en dekoder av porter.** Og dermed se hele veien fra opkoden i et instruksjonsord til ett enkelt styresignal.

5.2 Det begynner med en bryter

5.2.1 Sant og usant, skrevet som ett bit

Når maskinen kommer til linja `if alder >= 18`, må den finne ut én ting: er påstanden sann eller ikke?

Det finnes bare to svar. Ikke «nesten», ikke «litt», ikke «kommer an på». Enten er den sann, eller så er den det ikke.

To svar er nøyaktig det et bit rommer. Så maskinen skriver sann som **1** og usann som **0**, og dermed er hele spørsmålet redusert til ett eneste bit.

Dette er den samme ideen som bar kapittel 2: et bitmønster betyr ingenting i seg selv, det er avtalen om hva det skal bety som gir det mening. Der var avtalen at bitene var et tall eller en bokstav. Her er avtalen at ett bit er et ja eller et nei. Samme bit, ny avtale.

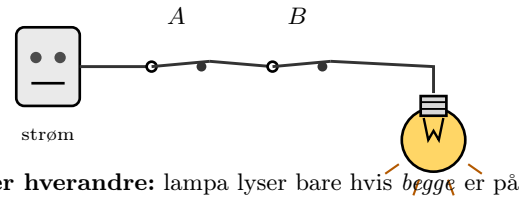
Resten av kapitlet handler om hva du kan bygge når det er alt du har.

5.2.2 To brytere og en lampe

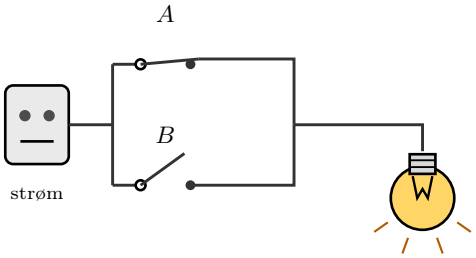
Glem maskinen et øyeblikk. Vi tar en stikkontakt, en lyspære og to brytere.

En bryter gjør én ting: den slipper strøm gjennom, eller den gjør det ikke. En lyspære gjør én ting: den lyser når det går strøm gjennom den.

Nå kan vi koble de to bryterne på to forskjellige måter, og det er verdt å stoppe ved hvor stor forskjell det blir.



etter hverandre: lampa lyser bare hvis *begge* er på



ved siden av hverandre: lampa lyser hvis *minst én* er på

Figur 5.1 De to måtene å koble to brytere på. Øverst må strømmen gjennom begge for å nå fram til pæra. Nederst har den to veier å velge mellom, og her er A på mens B er av — det holder likevel. Returledningen tilbake til kontakten er utelatt i begge tegningene; den finnes, men den har ingenting med saken å gjøre.

Øverst står bryterne **etter hverandre**. Strømmen må gjennom den første for å komme til den andre. Slår du av én av dem, er veien brutt, og pæra er mørk. Begge må være på.

Nederst står de **ved siden av hverandre**, og strømmen har to veier å velge mellom. Er bare én av dem åpen, kommer strømmen fram likevel. Det holder at minst én er på.

Se nå på de to beskrivelsene med programmeringsøyne:

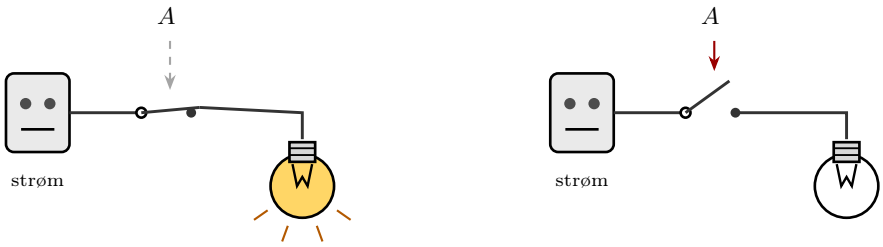
- *Begge* må være på. Det er **AND**.
- *Minst én* må være på. Det er **OR**.

Du har nettopp bygget to av de tre ordene vi startet kapitlet med, av en stikkontakt, en lyspære og to brytere. Det er ikke en analogi eller en forenkling. Det er det samme.

5.2.3 Og den som gjør det motsatte

Det tredje ordet var NOT, og her må vi bytte ut en del.

Bryterne vi har brukt til nå, er åpne når ingen rører dem. Det finnes også brytere som er motsatt: de er lukket i hvilestilling, og trykker du dem inn, bryter de strømmen. Setter du en slik i kretsen, lyser lampa helt til noen trykker.



ingen rører den: pæra lyser noen trykker: pæra er mørk

Figur 5.2 En bryter som er lukket i hvilestilling. Den gjør det motsatte av de andre: trykker du den inn, bryter du strømmen. Pila viser hvor du trykker.

Du har sett det i praksis. Lyset inne i et kjøleskap lyser når døra *ikke* er lukket; det er døra selv som trykker inn bryteren når den går igjen.

Dermed har vi alle tre. AND, OR og NOT, laget av ledninger og to slags brytere.

5.2.4 En som ikke lar seg lage slik

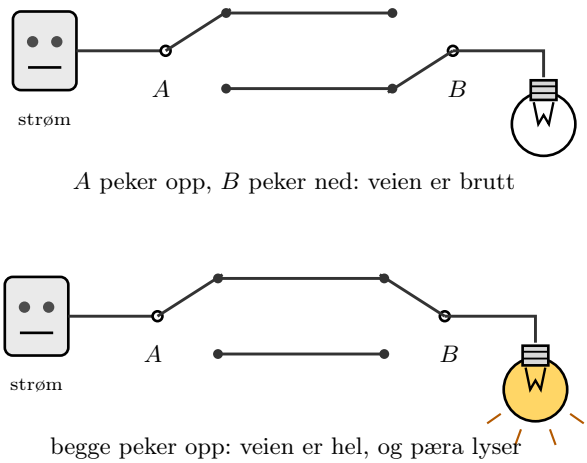
Men vent litt. Det finnes en kobling du har brukt mange ganger, og den passer ikke inn i noen av de to mønstrene.

Det henger to brytere i en trappeoppgang. Du kommer inn nederst, slår lyset på og går opp. Oppe i andre etasje slår du det av igjen med den andre bryteren. Neste gang går kanskje noen motsatt vei, og det virker like godt.

Legg merke til hva som er rart her. Ingen av de to bryterne har en på-stilling og en av-stilling. Hver av dem har bare to stillinger, opp og ned, og ingen av stillingene betyr «lys» i seg selv.

Prøv å forklare det med det vi har til nå. Er det «begge må stå opp»? Nei — lyset kan være på med én oppe og én nede. Er det «minst én må stå opp»? Nei — med begge oppe kan lyset være av.

Det som faktisk avgjør, er om de to står i *samme* stilling eller ikke. Og grunnen er lett å se når kretsen er tegnet.



Figur 5.3 Trappekoblingen. Mellom de to bryterne går det *to* ledninger, ikke én, og hver bryter velger hvilken av dem den skal bruke. Snu én av bryterne i den nederste tegningen, så er du tilbake i den øverste.

Det går to ledninger mellom bryterne, og hver bryter velger hvilken av dem den skal bruke. Peker begge på den samme, er veien hel. Peker de på hver sin, er den brutt.

Snu nå én av dem. Sto de likt, står de nå ulikt. Sto de ulikt, står de likt. Uansett hvilken vei det gikk, skifter lyset — og det er derfor koblingen virker uansett hvilken bryter du tar, og uansett hvem som var der før deg.

Note-to-myself

Er dette «lyser når de står likt» eller «lyser når de står ulikt»?
Det kommer an på hva du bestemmer deg for å kalle hva. Sier du at «opp» er 1 på begge bryterne, lyser lampa når de er like. Sier du at «opp» er 1 på den ene og «ned» er 1 på den andre, lyser den når de er ulike. Samme kobling, samme ledninger — bare en annen avtale om hva tallene betyr. Det er nøyaktig poenget fra kapittel 2: bitene betyr det vi blir enige om at de skal bety. Det som er sant uansett hvilken avtale du velger, er at ett bytte alltid snur lyset.

Uansett hvilken av de to lesemåtene du velger, er dette en tredje oppførsel. Den lar seg ikke lage med brytere etter hverandre, og ikke med brytere ved siden av hverandre.

Så det finnes flere oppførsler enn de to første. Hvor mange finnes det i det hele tatt? Vi kommer tilbake til det spørsmålet, og svaret er mindre enn du skulle tro.

5.2.5 Porten: et navn på oppførselen

Vi kunne fortsatt å tegne brytere resten av kapitlet. Det ville vært en dårlig idé, av to grunner.

Den første er at det blir for mange. En prosessor inneholder milliarder av slike småting, og du blir aldri ferdig med tegningen.

Den andre er at bryterne der inne ikke ser slik ut. Det sitter faktisk brytere i en brikke — milliarder av dem — men de har ingen bevegelige deler og ingen vippearm. De styres av spenning i stedet for av en finger, og de heter transistorer. Hvordan de er laget, er kapittel 6.

Modellen du nettopp bygde er altså ikke feil. Den er bare tegnet med feil slags bryter.

Derfor gjør vi som folk alltid gjør når noe blir for detaljert: vi tegner et symbol i stedet, ett for hver oppførsel. Symbolet kaller vi en **port**.

Navnet kommer fra den første av dem, der bildet passet: en port som slipper noe gjennom, eller lar være. For de andre er det bare blitt hengende ved som navn, og du skal ikke lese for mye ut av det.

En port har tre faste egenskaper, og de gjelder alle sammen:

- Den har noen få innganger — som regel to. Hver inngang er 0 eller 1.
- Den har *nøyaktig én* utgang, som også er 0 eller 1.
- Den har en fast regel for hva utgangen blir. Regelen endrer seg aldri.

Det er hele byggesettet. Vi kommer til å bruke sju porttyper i denne boka, og fire av dem har du allerede møtt som brytere og lamper.

Note-to-myself

På engelsk heter en port *gate*, og det er det ordet du finner i all faglitteratur og i alle kretsprogrammer. «AND gate», «NOR gate».
Vi bruker det norske ordet *port* om tingen, og de engelske forkortelsene AND, OR, NOT og så videre om de enkelte typene. Det er slik de står på lysarkene, og det er slik de står på brikkene.

5.2.6 Funksjon: ordet som betyr tre ting

Nå trenger vi et ord til, og det er det vanskeligste ordet i kapitlet. Ikke fordi det betyr noe komplisert, men fordi du har møtt det før i to andre betydninger.

Vi trenger et ord for *selve oppførselen* — for «lampa lyser bare når begge er på» — løsrevet fra om den er laget av brytere, av en port eller av noe helt annet.

Det ordet er **funksjon**.

Du så faktisk en av dem allerede i kodelinja på første side: `slipp_inn()` er en Python-funksjon. I Python gir du en funksjon noen verdier, og du får en verdi tilbake. I matematikken har du kanskje sett $f(x) = 2x + 1$, som ser annerledes ut, men gjør det samme.

Den logiske funksjonen ligger nærmest den matematiske. Den er en regel, ikke en handling. Sett inn verdier, og svaret følger — ikke fordi noe kjører, men fordi det bare *er* slik.

Det er verdt å holde fast ved. En Python-funksjon gjør noe: den starter, den bruker tid, den blir ferdig. En port gjør ingenting. Den står der med noen signaler på inngangene, og da er utgangen det den er. Du trykker ikke på start.

Ellers er likheten med Python god nok til å støtte seg på: noe går inn, én verdi kommer ut. To ting skiller:

- Alt som går inn og ut er bare 0 eller 1. Ingen tekst, ingen desimaltall, ingen lister.
- Det finnes så få muligheter at du kan skrive opp *alle sammen*.

Den siste er verdt å stoppe ved, for det er den som gjør dette stoffet så håndfast. En Python-funksjon som tar et tall, kan du aldri prøve for alle tall; det er uendelig mange av dem. En logisk funksjon med to innganger har nøyaktig fire tilfeller. Fire. Du kan skrive opp alle på en serviett og så er du ferdig, for alltid, med absolutt sikkerhet.

Den lista har et navn, *sannhetsverditabell*, og den kommer i seksjon 5.4. Den er det viktigste verktøyet i kapitlet.

Note-to-myself

Tre betydninger av det samme ordet, og de er i slekt uten å være like:

- **I Python:** en kodeblokk med `def` foran.
- **I matematikken:** en regel som gir ett svar for hver verdi du setter inn.
- **Her:** oppførselen til en krets — hva som kommer ut for hver eneste kombinasjon av det som går inn.

I denne boka betyr «funksjon» alltid den siste. Mener vi en av de andre, sier vi det uttrykkelig.

5.2.7 Krets: veien, ikke målet

Det tredje ordet krever minst forklaring. En **krets** er porter koblet sammen med ledninger. Det fysiske.

Og her er hele poenget med kapitlet, i én setning: **den samme funksjonen kan bygges av mange forskjellige kretser.**

Du har allerede sett det. Serieoppkoblingen med to brytere og en AND-port er to helt ulike ting. Den ene er av metall og plast, den andre er et symbol på en brikke. Men de oppfører seg likt i alle fire tilfellene, og da er funksjonen den samme.

Tenk på funksjonen som et sted du skal, og kretsen som veien dit. Det finnes mange veier fra den ene siden av en by til den andre. De ender samme sted, men de er ikke like lange, og noen av dem koster mer enn andre.

Tankevekker

Dette er den samme forskjellen som i kapittel 4, der skillet gikk mellom hva en maskin kan gjøre og hvordan den gjør det.
Her er det igjen, flere lag lenger ned. Funksjonen er avtalen om hva som skal skje, kretsen er byggingen, og du står fritt til å bygge billigere så lenge avtalen holdes.
Legg merke til at dette skillet kommer tilbake på hvert eneste nivå i faget. Det er ikke en tilfeldighet; det er sånn hele stabelen er satt sammen.

5.2.8 Derfor er det verdt å spare porter

Hvis flere veier fører fram, hvorfor bry seg om hvilken?

Fordi hver port koster noe. Det er lett å overse når kretsen er tegnet på papir og består av tre symboler. Det er ikke like lett å overse når den er trykt i noen milliarder eksemplarer på en brikke.

- **Plass.** Porten tar areal, og arealet på en brikke er det dyreste du har.
- **Strøm.** Hver port trekker strøm, også når den ikke gjør noe særlig.
- **Varme.** Strømmen blir til varme, og varmen må ut et sted.
- **Tid.** Et signal bruker tid gjennom en port. Flere porter etter hverandre betyr lengre vei, og lengre vei betyr at klokka må gå saktere.

Den siste er den som binder kapitlet til det forrige. Kapittel 4 slo fast at klokka er stilt etter det tregeste steget i maskinen. Nå skal vi se hva det tregeste steget faktisk består av.

Derfor er forenkling ikke pynt, og ikke en skoleøvelse. Å fjerne en port er det samme som å gjøre brikken litt mindre, litt kjøligere og litt raskere, alt sammen på én gang.

5.3 Portene

Nå skal vi møte hele samlingen. Sju porter: fire grunnleggende, og tre som er de samme med en boble på utgangen.

Fire av dem kjenner du allerede fra forrige seksjon, du har bare ikke sett symbolene. De kommer først.

For hver port får du tre ting: symbolet, tabellen over hva den gjør i hvert enkelt tilfelle, og en kort skrivemåte.

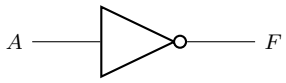
Den siste trenger en setning. Å skrive «lampa lyser bare hvis begge er på» hver gang blir tungt, så vi forkorter. Inngangene kaller vi A og B, og utgangen F. OG skrives som et gangetegn, eller bare ved å sette bokstavene inntil hverandre: $A \cdot B$ eller AB. ELLER skrives som pluss: $A + B$. IKKE skrives som en strek over: $\overline{A}$.

Det er hele skrivemåten. Tre tegn, og du har sett dem alle nå.

5.3.1 IKKE-porten

NOT-porten, også kalt **inverteren**, er kjøleskapslyset fra forrige seksjon. Den er den eneste porten med bare én inngang, og den snur signalet. Inn 0 gir ut 1, inn 1 gir ut 0.

$$F = \overline{A}$$



A	F
0	1
1	0

Figur 5.4 NOT-porten. Trekanten er selve porten, og den lille sirkelen i spissen er inverteringen.

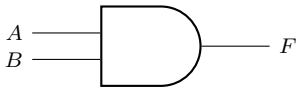
Note-to-myself

Du kommer til å møte en annen skrivemåte enn vår. På lysarkene og på portarket står «ikke A» med **apostrof**: A'. Det betyr nøyaktig det samme som $\overline{A}$; forskjellen er bare typografi.
Overstreken har én fordel. Sammenlign $\overline{A}\,\overline{B}$ med $\overline{AB}$. Den første er «ikke A, og ikke B». Den andre er «ikke (A og B)». Det er to helt forskjellige ting, og streken viser forskjellen med en gang. Med apostrof blir det $A'B'$ mot $(AB)'$, der alt hviler på en parentes som er lett å overse.
Hvorfor det er to forskjellige ting, kommer vi tilbake til. Det er en av fellene i dette stoffet.

5.3.2 OG-porten

AND-porten er de to bryterne etter hverandre. Den gir 1 bare når begge inngangene er 1, og 0 i alle tre andre tilfeller.

$$F = A \cdot B$$



A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Figur 5.5 AND-porten. Flat venstreside og rund høyreside, som en D.

En mikrobølgeovn starter bare når døra er lukket *og* noen har trykket på start. Begge må være oppfylt. Det er en AND, og det er også grunnen til at den stopper i det du åpner døra.

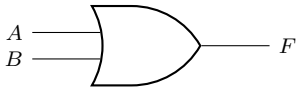
Fun Fact

AND-porten er den eneste av de sju som er vanlig multiplikasjon. Se i tabellen: $0 \cdot 0 = 0$, $0 \cdot 1 = 0$, $1 \cdot 0 = 0$, $1 \cdot 1 = 1$. Det stemmer rad for rad.
Derfor er gangetegnet ikke en lånt metafor her, slik plusstegnet er for ELLER. For AND stemmer det faktisk.

5.3.3 ELLER-porten

OR-porten er de to bryterne ved siden av hverandre. Den gir 1 når minst én av inngangene er 1, og bare 0 når begge er 0.

$$F = A + B$$



A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Figur 5.6 OR-porten. Buet venstreside, og den møtes i en spiss til høyre.

Lyset i en fellesgang slås på når bevegelsessensoren merker noe *eller* noen trykker på bryteren. Én av delene holder.

Her ligger en liten språkfelle. På norsk sier vi gjerne «enten kaffe eller te» og mener én av dem, ikke begge. Det boolske ELLER er mer rundhåndet: det er sant også når begge er sanne. Den strenge varianten, der nøyaktig én skal være sann, er den neste porten.

Advarsel :)

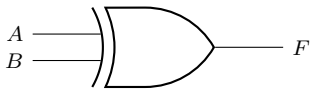
Plusstegnet i $A + B$ er ikke addisjon.
Det er lånt fra matematikken fordi ELLER oppfører seg som pluss i tre av fire tilfeller: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$. Men i det fjerde brister det. Her er $1 + 1 = 1$, fordi «minst én av dem er sann» fortsatt bare er sant. Det er ikke 2, og det er ikke 0 med mente.
Si «A eller B» høyt når du leser det, ikke «A pluss B». Det koster ingenting og sparer deg for den

vanligste feilen i dette stoffet.

5.3.4 ENTEN-ELLER-porten

XOR-porten, etter engelsk *exclusive or*, gir 1 når inngangene er *forskjellige*. Står de likt, gir den 0.

$$F = A \oplus B$$



A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Figur 5.7 XOR-porten. Som OR, men med en ekstra bue utenfor venstresiden.

Dette er trappelyset. Koblingen som ikke lot seg lage av brytere etter hverandre eller ved siden av hverandre, har altså fått sin egen port.

Husker du notatet om at det kom an på hva du kalte hva? Her er oppgjøret. Velger du avtalen der lampa lyser når bryterne står *ulikt*, er trappekoblingen en XOR. Velger du den andre avtalen, er den en XNOR — porten vi møter om et par sider. De to er hverandres motsetninger, og koblingen i veggen er den samme uansett.

Det som gjelder for begge, er egenskapen vi fant: endrer du én av to verdier, snur svaret alltid. Det er derfor lysbryteren i oppgangen virker uansett hvem som var der før deg.

Elektrikere kaller koblingen en toveisbryter og tenker neppe på boolsk algebra mens de monterer den. Funksjonen er like fullt den samme.

XOR er porten du kommer til å møte oftest i resten av kapitlet. Den dukker opp fire ganger: i halvadderen, i fulladderen, i kretsen som oppdager overflyt, og i subtraksjonskretsen.

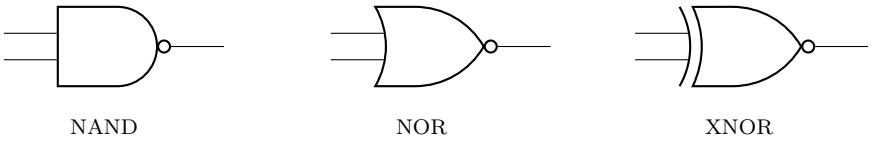
Grunnen er at XOR er den eneste porten som svarer på spørsmålet «er disse to forskjellige?». Og det spørsmålet er, merkelig nok, det samme som «hva blir summen av dem?» når du legger sammen to bit. Vi kommer tilbake til det i seksjon 5.7.

5.3.5 De tre med boble

De tre siste portene er ikke nye funksjoner å lære utenat. De er de tre du nettopp har sett, med utgangen snudd.

- **NAND** er AND med invertert utgang: $F = \overline{A \cdot B}$.
- **NOR** er OR med invertert utgang: $F = \overline{A + B}$.
- **XNOR** er XOR med invertert utgang: $F = \overline{A \oplus B}$.

Symbolene følger samme logikk. Du tegner grunnporten og setter en liten sirkel i utgangen.



Figur 5.8 De tre inverterte portene. Sirkelen i utgangen er hele forskjellen fra AND, OR og XOR.

Sirkelen betyr invertering, hvor den enn står. Når du ser den i en kretstegning, vet du at signalet snus akkurat der. Det er den eneste huskeregelen du trenger for de tre siste portene — da er det fire porttyper å holde styr på, ikke sju.

XNOR fortjener en ekstra setning, fordi den er den motsatte av XOR og dermed svarer på «er disse to *like*?». En krets som sammenligner to bit og sier fra om de er identiske, er altså én port. Den kommer tilbake i seksjon 5.6.

5.3.6 Oversikten

Tabell 5.1 er hele byggesettet samlet.

Port	**Uttrykk**	**00**	**01**	**10**	**11**
AND	$F = A \cdot B$	0	0	0	1
OR	$F = A + B$	0	1	1	1
XOR	$F = A \oplus B$	0	1	1	0
NAND	$F = \overline{A \cdot B}$	1	1	1	0
NOR	$F = \overline{A + B}$	1	0	0	0
XNOR	$F = \overline{A \oplus B}$	1	0	0	1

Tabell 5.1 De seks toinngangsportene. Kolonnene er de fire kombinasjonene av A og B. NOT står utenfor, siden den bare har én inngang.

Sammenlign radene parvis: AND mot NAND, OR mot NOR, XOR mot XNOR. I hvert par er hvert eneste tall snudd. Det er den samme observasjonen som boblen i symbolet gjør visuell.

Fun Fact

Tidligere spurte vi hvor mange oppførsler som finnes i det hele tatt. Nå kan vi svare, og svaret krever bare telling.
En funksjon med to innganger har fire tilfeller. For hvert av dem kan utgangen settes til 0 eller 1, helt fritt. Det gir $2 \cdot 2 \cdot 2 \cdot 2 = 16$ mulige oppførsler, og flere finnes ikke.
Seks av de seksten har fått navn og symbol som toinngangsporter, og de står i tabellen over. De ti andre er ikke uinteressante, bare mindre: fire av dem overser den ene inngangen fullstendig, og da trenger du uansett ingen toinngangsport — det holder med en NOT eller en bar ledning. To av dem gir samme svar uansett hva du mater inn. De fire siste har navn i logikken, men står sjelden som egne porter på en brikke.
Hele verden av toinngangsporter er altså seksten store. Det er ikke noe mer der ute.

Note-to-myself

Denne tabellen er ikke noe du skal pugge.
Du skal kunne *bruke* den: gå fra en krets til en tabell, fra en tabell til et uttrykk, og se at to kretser gjør det samme. Selve oversikten er et oppslagsverk, og den er det lov å ha foran seg.
Det du bør ha i hodet uten å slå opp, er de tre grunnportene, og at en boble snur utgangen.

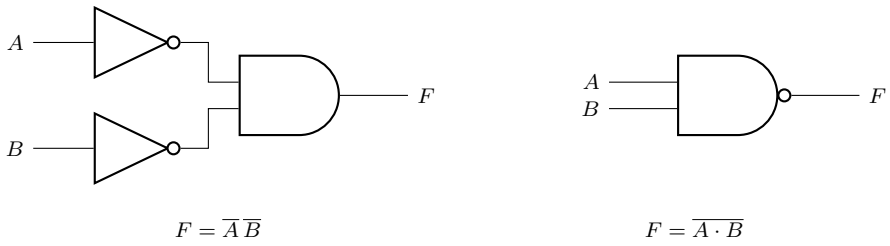
5.3.7 To feller

To feil går igjen år etter år, og begge er lette å unngå når du vet om dem.

Den første er plusstegnet, og den er allerede nevnt. $1 + 1 = 1$.

Den andre er viktigere, og den koster flest poeng. Det er forskjell på å invertere inngangene og å invertere utgangen.

Se på disse to kretsene.



Figur 5.9 To kretser som ser beslektet ut og gjør helt forskjellige ting. Til venstre snus signalene før porten, til høyre snus svaret etter.

Til venstre snus A og B hver for seg, og så spør vi om begge de snudde er 1. Til høyre spør vi først om begge de opprinnelige er 1, og snur så svaret.

Regn gjennom et enkelt tilfelle: $A = 1$, $B = 0$.

Venstre krets: $\overline{A} = 0$, $\overline{B} = 1$. OG mellom 0 og 1 er 0. Utgangen er **0**.

Høyre krets: $A \cdot B = 0$. Snudd blir det **1**.

Samme innganger, motsatt svar.

Advarsel :)

Når du ser en krets med invertere i, er det første spørsmålet: *hvor* står de?
Står de foran porten, hører de til hver sin inngang, og du skriver $\overline{A}$ og $\overline{B}$ hver for seg. Står boblen i utgangen på porten, hører inverteringen til hele uttrykket, og overstreken går over alt sammen.
Dette er den feilen flest gjør på dette stoffet. Den oppstår fordi øyet ser «det er en invertering her et sted» og hopper rett til svaret.

5.4 Sannhetsverditabellen

En **sannhetsverditabell** lister opp alle mulige kombinasjoner av innganger, og hva utgangen blir for hver av dem.

Det er en uttømmende beskrivelse. Når tabellen er fylt ut, finnes det ingen situasjon kretsen kan havne i som ikke står der. Det er noe av det fineste med digital logikk: antall tilfeller er endelig, og du kan sjekke alle.

5.4.1 Hvor mange rader

Hver inngang kan være 0 eller 1. To valg per inngang, og valgene er uavhengige.

Med n innganger blir det 2^n rader.

Antall innganger	Antall rader
1	2
2	4
3	8
4	16
10	1 024

Tabell 5.2 Tabellen dobles for hver inngang du legger til.

Dette er den samme doblingen som i kapittel 2: én bit til gir dobbelt så mange mønstre. Her er konsekvensen at tabeller blir ubrukelige fort. Ti innganger gir over tusen rader, og ingen setter opp den for hånd.

I dette kurset holder vi oss til høyst tre innganger. Åtte rader er nok til å vise alt som er verdt å vise, og få nok til at du kan sjekke hver eneste av dem.

5.4.2 Trikset som gjør at du ikke glemmer noen

Det vanligste problemet med å sette opp en tabell er ikke at man regner feil. Det er at man glemmer en rad, eller skriver samme kombinasjon to ganger.

Det finnes et triks som fjerner problemet helt: **skriv radene som binære tall, i tellerekkefølge.**

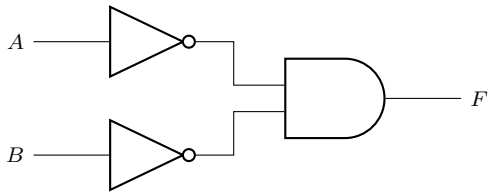
Med to innganger teller du 0, 1, 2, 3 — altså 00, 01, 10, 11. Med tre innganger teller du 0 til 7, altså 000, 001, 010, 011, 100, 101, 110, 111.

Da kommer alle kombinasjonene, hver nøyaktig én gang, og i en rekkefølge alle bruker. Det betyr at to personer som setter opp samme tabell, får radene i samme orden og kan sammenligne direkte.

Legg merke til hva som skjer i kolonnene når du teller slik. Den høyre skifter for hver rad, den neste annenhver, den neste igjen hver fjerde. Det er bare hvordan telling i totallssystemet ser ut, men det er praktisk: du kan fylle ut inngangskolonnene rett nedover uten å tenke, og konsentrere deg om utgangen.

5.4.3 Å lese en krets forlengs

Her er en krets. Tre porter, to innganger.



Figur 5.10 Tre porter. To invertere og en AND.

Framlengs betyr at du starter til venstre og følger signalene mot høyre. Du navngir hver ledning etter hvert som du kommer til den.

1. Inn kommer A og B.
2. A går gjennom en inverter. Ut av den kommer $\overline{A}$.
3. B går gjennom en inverter. Ut av den kommer $\overline{B}$.
4. De to møtes i en AND-port, som gir OG mellom dem.
5. Utgangen er $F = \overline{A} \cdot \overline{B}$.

Metoden er mekanisk, og det er poenget. Du trenger ikke forstå hva kretsen gjør for å skrive opp uttrykket; du trenger bare å følge ledningene og skrive ned hva hver port gjør med det den får inn.

5.4.4 Og baklengs

Baklengs betyr at du starter ved utgangen og spør: hva er den siste porten?

Her er den siste porten en AND. Altså er F et produkt av to ting. Så spør du hva de to tingene er, og går ett hakk til venstre for hver av dem. Begge viser seg å være en invertert inngang.

Du ender på samme svar. Forskjellen er hva metoden egner seg til.

- **Framlengs** er best når du skal finne ut hva en krets faktisk gjør, og når du skal lete etter feil. Du følger signalet slik det går.
- **Baklengs** er best når du har et uttrykk og skal tegne kretsen, eller når du leter etter unødvendige porter. Du begynner med målet.

Kildematerialet til dette kapitlet lovet begge metodene og viste bare den første. Begge er verdt å ha, og de tar fem minutter å lære.

5.4.5 Tre porter som var én

Sett opp tabellen for kretsen i figur 5.10. Regn rad for rad: først hva inverterne gir, så hva AND-porten gjør med det.

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

Tabell 5.3 Kretsen regnet ut rad for rad. Mellomkolonnene er inverterne.

Utgangen er 1 i én eneste rad: den der begge inngangene er 0.

Slå opp i tabell 5.1 på side 116 og let etter mønsteret $1, 0, 0, 0$.

Det er NOR.

Tankevekker

Tre porter gjør nøyaktig det samme som én.
$\overline{A} \cdot \overline{B}$ og $\overline{A + B}$ er to uttrykk for samme funksjon. Kretsen til venstre bruker to invertere og en AND, altså tre porter og mer plass. Kretsen til høyre bruker én NOR-port.
Dette er kapitlet i miniatyr. Funksjonen er den samme, kretsen er ikke, og den ene er tre ganger så dyr som den andre.

Du har nettopp brukt en av de mest kjente reglene i boolsk algebra uten å få navnet på den. Den kommer i seksjon 5.6.

Tenk om. . .

. . . du skulle finne ut dette ved å prøve deg fram i en ferdig brikke, uten tabellen. Du måtte målt utgangen for hver kombinasjon av innganger, notert resultatene, og sammenlignet med noe du allerede kjente.
Det er nøyaktig det tabellen gjør, bare på papir og på et halvt minutt. Derfor er sannhetsverditabellen det første verktøyet du griper til når en krets er ukjent: den gjør gjetting om til oppslag.

5.5 Fra tabell til uttrykk

Hittil har vi gått fra krets til tabell. Nå går vi motsatt vei: fra en tabell til et uttrykk.

Dette er den viktigste metoden i kapitlet. Den er oppskriften som gjør et ønske om hva en krets *skal* gjøre, om til noe du kan bygge. Og den er oppskriften vi bruker til å utlede adderkretsen senere.

5.5.1 Vi leter etter ettallene

Regelen er enkel: **se bare på radene der utgangen er 1.**

Grunnen er praktisk. Når vi beskriver noe digitalt, beskriver vi som regel når noe skal *skje*. Når skal alarmen gå? Når skal motoren starte? Når skal mente sendes videre? Det er lettere å liste opp de få tilfellene der svaret er ja, enn alle de andre.

Tenk på en lampe. Du forklarer den ved å si «den lyser når bryteren står på». Du ramser ikke opp alle situasjonene der den ikke lyser.

5.5.2 Ett ledd per rad

For hver rad der $F = 1$, skriver du ett ledd. Leddet er en AND mellom alle inngangene, og hver inngang skrives slik at den blir 1 i akkurat den raden.

- Er inngangen 1 i raden, skriver du bokstaven som den er.
- Er inngangen 0 i raden, skriver du bokstaven med overstrek.

Hvorfor blir en null til en overstrek? Fordi leddet skal være sant nettopp i den raden. Når $A = 0$, er $\overline{A} = 1$. Så det er $\overline{A}$ som er «på» i den raden, ikke A.

Til slutt kobler du alle leddene med ELLER. Logikken er: F er 1 hvis den første situasjonen inntreffer, *eller* den andre, *eller* den tredje.

La oss gjøre det på en tabell.

A	B	F	Ledd
0	0	1	$\overline{A}\,\overline{B}$
0	1	1	$\overline{A}B$
1	0	0	—
1	1	1	AB

Tabell 5.4 Tre rader har 1 på utgangen, og hver av dem gir ett ledd.

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

Sjekk at det stemmer ved å prøve en rad. Sett $A = 0$ og $B = 1$. Første ledd: $1 \cdot 0 = 0$. Andre ledd: $1 \cdot 1 = 1$. Tredje ledd: $0 \cdot 1 = 0$. ELLER mellom 0, 1 og 0 er 1, som er det tabellen sier.

Legg merke til at i hver rad der F er 1, er det nøyaktig ett ledd som er 1. Det må være slik: hvert ledd er skreddersydd for sin ene rad og er 0 i alle andre. I radene der F er 0, er alle leddene 0.

Note-to-myself

Formen du nettopp skrev har et navn: **sum av produkter**, eller SOP etter engelsk *sum of products*. «Produktene» er AND-leddene, og «summen» er ELLER-en som binder dem sammen. Navnet er verdt å kjenne fordi det dukker opp overalt i dette fagfeltet. Men metoden er viktigere enn navnet.

5.5.3 Tre innganger

Oppskriften endrer seg ikke når det blir en inngang til. Hvert ledd får bare tre bokstaver i stedet for to.

A	B	C	F	Ledd
0	0	0	1	$\overline{A}\,\overline{B}\,\overline{C}$
0	0	1	0	—
0	1	0	1	$\overline{A}B\overline{C}$
0	1	1	0	—
1	0	0	1	$A\overline{B}\,\overline{C}$
1	0	1	0	—
1	1	0	0	—
1	1	1	1	ABC

Tabell 5.5 Åtte rader, fire av dem med 1 på utgangen.

$$F = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$

Dette uttrykket er stygt. Fire ledd, tolv bokstaver, og hvis du bygger det slik det står, trenger du tre invertere, fire AND-porter med tre innganger hver, og en OR-port med fire innganger.

Det er åtte porter for noe som skal vise seg å trenge to. Men vi må ha uttrykket først, før vi kan gjøre noe med det.

5.5.4 Og hvis vi hadde tatt nullene?

Et rimelig spørsmål: hvorfor nettopp ettallene? Kunne vi like gjerne beskrevet når utgangen er 0?

Ja. Det går helt fint, og formen har et eget navn — *produkt av summer* — der du skriver ett ELLER-ledd per nullrad og binder dem sammen med OG. Resultatet er en annen, men likeverdig beskrivelse av samme funksjon.

Vi holder oss til ettallene i denne boka, av to grunner. Den ene er at det passer måten vi tenker på: vi beskriver når noe skal aktiveres. Den andre er at det i praksis ofte er færre ettall enn nuller, og da blir uttrykket kortere.

Men det er verdt å vite at valget finnes. Hvis du noen gang møter en tabell med sytten ettall og tre nuller, er det nullene du vil beskrive.

5.6 Forenkling

Vi har et uttrykk. Nå skal vi gjøre det mindre.

Dette er den delen av kapitlet som ligner mest på vanlig matematikk, og den gode nyheten er at reglene stort sett er de du allerede kan. Du setter utenfor parentes, du stryker ledd som ikke bidrar, og du leter etter mønstre.

5.6.1 Reglene

Sju regler rekker langt. De tre første er nesten for opplagte til å skrive ned, og nettopp derfor glemmer man dem.

Navn	Regel	I ord
Nøytralt element	$X + 0 = X \quad X \cdot 1 = X$	Legger ingenting til
Dominans	$X + 1 = 1 \quad X \cdot 0 = 0$	Avgjør alene
Idempotens	$X + X = X \quad X \cdot X = X$	Å si det to ganger hjelper ikke
Komplement	$X + \overline{X} = 1 \quad X \cdot \overline{X} = 0$	Én må være sann, aldri begge
Felles faktor	$XY + XZ = X(Y + Z)$	Sett utenfor parentes
Absorpsjon	$X + XY = X$	Det lange leddet er overflødig
Absorpsjon, variant	$\overline{X} + XY = \overline{X} + Y$	X i andre ledd kan strykes

Tabell 5.6 Regelsettet. De to siste er de som faktisk sparer porter.

De fire første kan du kontrollere ved å sette inn 0 og 1 for X og regne etter. Felles faktor er distributiv lov, akkurat som i vanlig algebra.

De to absorpsjonsreglene er de eneste som gjør virkelig arbeid, og de er verdt å overbevise seg om.

$X + XY = X$. Hvis $X = 1$, er hele uttrykket 1, uansett hva Y er. Hvis $X = 0$, faller første ledd bort, og andre ledd blir $0 \cdot Y = 0$. Altså er svaret alltid lik X. Leddet XY endrer aldri noe.

$\overline{X} + XY = \overline{X} + Y$. Hvis $X = 0$, er første ledd 1, og begge sider er 1. Hvis $X = 1$, faller første ledd bort på begge sider, og vi står igjen med Y mot Y. Altså like.

Den siste er den som oftest blir oversett, og den som gjør mest.

5.6.2 Første eksempel, helt ut

Vi tar uttrykket fra tabell 5.4 på side 121. Underveis er det **fargede** leddet alltid det vi holder på med akkurat da.

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

Steg 1. De to første leddene har $\overline{A}$ felles:

$$F = \overline{A}\,\overline{B} + \overline{A}B \;+\; AB$$

Sett $\overline{A}$ utenfor parentes:

$$F = \overline{A}\,(\overline{B} + B) \;+\; AB$$

Steg 2. Inni parentesen står $\overline{B} + B$. Den er 1 etter komplementregelen: enten er B sann, eller så er den ikke, og noen tredje mulighet finnes ikke.

$$F = \overline{A} \cdot 1 \; + \; AB$$

Og $\overline{A} \cdot 1$ er bare $\overline{A}$:

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

Steg 3. Her stopper mange. Men se på formen på det som står igjen:

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

Det er nøyaktig $\overline{X} + XY$, med $X = A$ og $Y = B$. Absorpsjonsregelen sier at A-en inne i produktet kan strykes:

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

Fra tre ledd og seks bokstaver til to bokstaver. Kretsen er nå én inverter og én OR-port.

Sjekk mot tabellen: A=0, B=0 gir $1 + 0 = 1$. A=0, B=1 gir 1. A=1, B=0 gir $0 + 0 = 0$. A=1, B=1 gir $0 + 1 = 1$. Mønsteret er $1, 1, 0, 1$, som er tabellen.

Fun Fact

$\overline{A} + B$ har en tolkning det er verdt å stoppe ved: det er påstanden «hvis A, så B».
Prøv den. Utsagnet «hvis det regner, tar jeg med paraply» er bare usant i ett tilfelle — det regner, og du har ikke paraply. Altså $A = 1$ og $B = 0$. Det er nøyaktig den ene raden der uttrykket vårt gir 0.
Implikasjon fra logikken og en to-ports krets viser seg å være samme sak.

5.6.3 Andre eksempel, og en overraskelse

Nå det stygge uttrykket fra tabell 5.5 på side 122. Samme framgangsmåte: det **fargede** er det vi holder på med.

$$F = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$

Steg 1. De tre første leddene har $\overline{C}$ felles:

$$F = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} \; + \; ABC$$

Sett $\overline{C}$ utenfor:

$$F = \overline{C}\,(\overline{A}\,\overline{B} + \overline{A}B + A\overline{B}) + ABC$$

Steg 2. Inni parentesen har de to første $\overline{A}$ felles:

$$F = \overline{C}\big(\overline{A}\,\overline{B} + \overline{A}B + A\overline{B}\big) + ABC$$

$$F = \overline{C}\big(\overline{A}(\overline{B} + B) + A\overline{B}\big) + ABC$$

Og $\overline{B} + B = 1$, så parentesen inni faller bort:

$$F = \overline{C}(\overline{A} + A\overline{B}) + ABC$$

Steg 3. Her er absorpsjonsregelen igjen, nå med $X = A$ og $Y = \overline{B}$:

$$F = \overline{C}(\overline{A} + A\overline{B}) + ABC$$

$$F = \overline{C}(\overline{A} + \overline{B}) + ABC$$

Stopp her et øyeblikk og se på hva som står. Det er $\overline{C}$ ganget med «ikke A eller ikke B», pluss AB ganget med C.

Sett $D = A \cdot B$. Da er $\overline{A} + \overline{B}$ det samme som $\overline{D}$ — det er De Morgans lov, og vi tar den i neste underseksjon. Uttrykket blir:

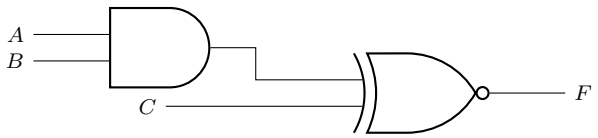
$$F = \overline{C}\,\overline{D} + DC$$

Som er «C og D er like». Det er XNOR.

$$F = \overline{C \oplus (A \cdot B)}$$

Tankevekker

Fire ledd, tolv bokstaver og åtte porter ble til **to porter**: én AND som regner ut $A \cdot B$, og én XNOR som sammenligner resultatet med C.
Og nå kan du lese funksjonen i klartekst, noe som var umulig før. Kretsen svarer på spørsmålet: «er C lik OG-en av A og B?»
Det er dette forenkling er til for. Den sparer porter, ja. Men den gjør også at kretsen får en mening du kan si høyt.



Figur 5.11 Den samme funksjonen som på forrige side, bygget på nytt. To porter i stedet for åtte.

5.6.4 De Morgan

Vi har nå brukt den samme regelen to ganger uten å navngi den. Første gang var da $\overline{A}\,\overline{B}$ viste seg å være en NOR-port. Andre gang var nå nettopp, da $\overline{A} + \overline{B}$ ble til $\overline{AB}$.

Regelen heter **De Morgans lover**, etter en britisk matematiker fra 1800-tallet, og den finnes i to utgaver:

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

I ord: når du drar en overstrek fra utsiden og inn, snur du operatoren. OG blir ELLER, og ELLER blir OG. Hver variabel får sin egen overstrek.

Du kan bruke dagligtalen til å kontrollere den. «Det er ikke sant at både Nora og Ahmed er hjemme» betyr det samme som «enten er ikke Nora hjemme, eller så er ikke Ahmed det». Og «ingen av dem er hjemme» betyr «Nora er ikke hjemme, og Ahmed er ikke hjemme».

Note-to-myself

De Morgan er grunnen til at det finnes to tegnemåter for de samme portene, og at erfarne folk bytter mellom dem midt i en tegning.
En NAND-port kan tegnes som en AND med boble på utgangen, eller som en OR med bobler på begge inngangene. De er samme funksjon, og hvilken du velger avhenger av hva du vil at leseren skal se.
Vi bruker bare den første formen i denne boka. Men hvis du en gang ser en OR-port med to bobler foran seg, vet du nå hva det er.

5.6.5 Hvor langt er langt nok

Et ærlig spørsmål: hvordan vet du at du er ferdig?

Det korte svaret er at med metoden i dette kapitlet vet du det ikke sikkert. Du leter etter felles faktorer, du prøver absorpsjonsreglene, og når du ikke finner flere, stopper du. Det er mulig å gå glipp av noe, og det skjer ofte.

Det finnes en systematisk metode som garanterer at du er i mål for små uttrykk. Den heter Karnaugh-diagram, og den står i appendiks B. Den er ikke pensum, men den er morsom, og den gjør det samme arbeidet på et rutenett i stedet for i en lang rekke av omskrivninger.

Advarsel :)

Pass deg for å erklære at et uttrykk ikke kan forenkles.
Prøv $F = AB + AC + D$. Ved første blikk ser det ut til at ingenting gjentar seg, og det er lett å si seg ferdig. Men A står i begge de to første leddene, og felles faktor gir $F = A(B + C) + D$. Det sparer en AND-port.
Regelen er å gå gjennom listen i tabell 5.6 på side 123 én gang til før du konkluderer. Særlig den enkleste regelen, felles faktor, er den som oftest blir oversett når uttrykket ser rotete ut.

5.7 Kretsen som legger sammen

Nå har vi verktøyet. Det som gjenstår, er å bruke det på noe maskinen faktisk trenger.

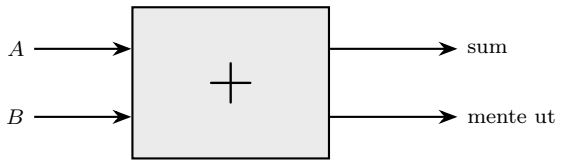
Kapittel 4 tegnet ALU-en som en bred V med tre innganger og to utganger, og sa rett ut at vi ikke åpnet den. Figur 4.5 på side 91 er den figuren. Her går vi inn.

Og vi går inn på det smaleste stedet: *hvordan legges to enkeltbit sammen?* Når det er løst, er resten en kopieringsjobb.

5.7.1 Den svarte boksen

Vi vet hva vi vil ha før vi vet hvordan.

Inn: to bit, A og B. Ut: svaret. Men svaret på $1 + 1$ får ikke plass i én bit, så vi trenger to utganger — summen, og en mente som sendes videre.



Figur 5.12 Det vi er ute etter. To bit inn, og et svar som ikke får plass i én bit.

Binær addisjon på papir går kolonne for kolonne, akkurat som i titallssystemet: legg sammen sifrene i én kolonne, skriv ned det som får plass, og send resten videre som mente. Kapittel 2 regnet gjennom flere slike oppstillinger i toerkomplement. Det er den ene kolonnen vi skal bygge nå.

5.7.2 Tabellen, og det du kjenner igjen

Fire kombinasjoner, og du kan svarene allerede.

$0+0=0$. $0+1=1$. $1+0=1$. Og $1+1$, som er 10 i totallssystemet: null i denne kolonnen, én i mente.

A	B	Sum	Mente ut
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabell 5.7 Addisjon av to enkeltbit. To utganger, og hver av dem er sin egen funksjon.

Her er grepet: **behandle de to utgangskolonnene som to separate funksjoner.** Det er to tabeller stablet ved siden av hverandre, og vi løser dem hver for seg.

Les sum-kolonnen nedover: $0, 1, 1, 0$. Slå opp i tabell 5.1 på side 116.

Det er XOR.

Les mente-kolonnen: $0, 0, 0, 1$. Det er AND.

$$\text{Sum} = A \oplus B \qquad \text{Mente ut} = A \cdot B$$

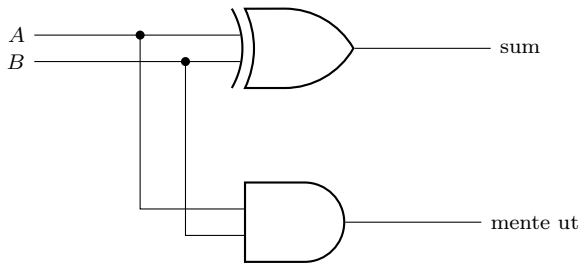
Tankevekker

Dette er ikke en tilfeldighet, og det er verdt ett avsnitt.
Summen av to bit er 1 når nøyaktig én av dem er 1. «Nøyaktig én» er definisjonen på XOR. Det er trappelyset fra anslaget, brukt til å regne.
Og mente oppstår bare når begge er 1, som er definisjonen på AND. Begge må bidra, ellers blir

det ingen overflyt til neste kolonne.
Maskinen legger sammen ved å stille to spørsmål om to bit: er de forskjellige, og er de begge sanne. Det er hele addisjonen.

5.7.3 Halvadderen

To porter, og kretsen er ferdig.



Figur 5.13 Halvadderen. A og B går til begge portene. De fylte prikkene betyr at ledningen deler seg; der linjer krysser uten prikk, er det ingen forbindelse.

Dette er en **halvadder**.

Note-to-myself

Om prikkene: en fylt sirkel der to ledninger møtes betyr at de er koblet sammen og fører samme signal. Krysser to streker hverandre uten prikk, går de bare over hverandre uten å røre. Konvensjonen er verdt å merke seg, for den er den eneste måten en flat tegning kan vise noe som i virkeligheten ligger i flere lag.

5.7.4 Det den mangler

Hvorfor «halv»?

Tenk på hvordan du legger sammen flerbits tall på papir. Den høyre kolonnen har to sifre å forholde seg til. Alle de andre har tre: de to sifrene, pluss menten fra kolonnen til høyre.

Halvadderen har bare to innganger. Den kan ikke ta imot mente. Den duger til den aller første kolonnen og ingen andre.

Vi trenger en versjon med tre innganger. Den heter **fulladder**.

5.7.5 Fulladderen, utledet

Først en navnesak, så slipper vi å snuble i den.

Menten heter **carry** på engelsk, og det er det ordet du møter i all litteratur og i alle kretstegninger. Derfor forkorter vi den med bokstaven C. Menten som kommer *inn* i kretsen fra posisjonen til høyre, skriver vi C_{inn}. Den som går *ut* til posisjonen til venstre, skriver vi C_{ut}.

Det er samme ting som «mente» hele veien. Vi bruker det norske ordet i teksten og den engelske forkortelsen i tabeller og formler, rett og slett fordi C_{inn} tar mindre plass enn «menten som kommer inn».

Tre innganger gir åtte rader. Vi setter dem opp i tellerekkefølge og fyller ut hva summen av de tre bitene blir.

A	B	C_{inn}	Sum	C_{ut}	Hvor mange ettall inn
0	0	0	0	0	ingen
0	0	1	1	0	ett
0	1	0	1	0	ett
0	1	1	0	1	to
1	0	0	1	0	ett
1	0	1	0	1	to
1	1	0	0	1	to
1	1	1	1	1	tre

Tabell 5.8 Fulladderen. Den siste kolonnen står der bare som kontroll: null ettall gir 0, ett gir 1, to gir 0 med mente, tre gir 1 med mente.

Nå bruker vi metoden fra seksjon 5.5 på hver av de to utgangene.

Mente ut er 1 i fire rader. Det gir:

$$C_{ut} = \overline{A}BC_{inn} + A\overline{B}C_{inn} + AB\overline{C_{inn}} + ABC_{inn}$$

Her kommer idempotensregelen til nytte, den som ser ut som den aldri skal brukes til noe. $X + X = X$ betyr også at vi kan skrive det samme leddet flere ganger uten å endre noe. Vi skriver ABC_{inn} tre ganger i stedet for én:

$$C_{ut} = (\overline{A}BC_{inn} + ABC_{inn}) + (A\overline{B}C_{inn} + ABC_{inn}) + (AB\overline{C_{inn}} + ABC_{inn})$$

Nå har hver parentes en felles faktor, og i hver av dem står et komplementpar som blir 1:

$$\begin{aligned}
\overline{A}BC_{inn} + ABC_{inn} &= BC_{inn}(\overline{A} + A) = BC_{inn} \\
A\overline{B}C_{inn} + ABC_{inn} &= AC_{inn}(\overline{B} + B) = AC_{inn} \\
AB\overline{C_{inn}} + ABC_{inn} &= AB(\overline{C_{inn}} + C_{inn}) = AB
\end{aligned}$$

Altså:

$$C_{ut} = AB + AC_{inn} + BC_{inn}$$

Les det høyt: mente ut er 1 når A og B er 1, *eller* A og menten inn er 1, *eller* B og menten inn er 1.

Med andre ord: **mente ut er 1 når minst to av de tre inngangene er 1.** Sjekk mot den siste kolonnen i tabell 5.8, så ser du at det stemmer rad for rad.

Summen er 1 i de fire andre radene:

$$\text{Sum} = \overline{A}\,\overline{B}C_{inn} + \overline{A}B\overline{C_{inn}} + A\overline{B}\,\overline{C_{inn}} + ABC_{inn}$$

Grupper de to første og de to siste etter om A er invertert:

$$\text{Sum} = \overline{A}(\overline{B}C_{inn} + B\overline{C_{inn}}) + A(\overline{B}\,\overline{C_{inn}} + BC_{inn})$$

Se på de to parentesene. Den første er «B og C_{inn} er forskjellige», altså $B \oplus C_{inn}$. Den andre er «de er like», altså $\overline{B \oplus C_{inn}}$.

Kall den første parentesen X. Da er den andre $\overline{X}$, og vi har:

$$\text{Sum} = \overline{A}X + A\overline{X}$$

Som igjen er «A og X er forskjellige»:

$$\text{Sum} = A \oplus B \oplus C_{inn}$$

Tankevekker

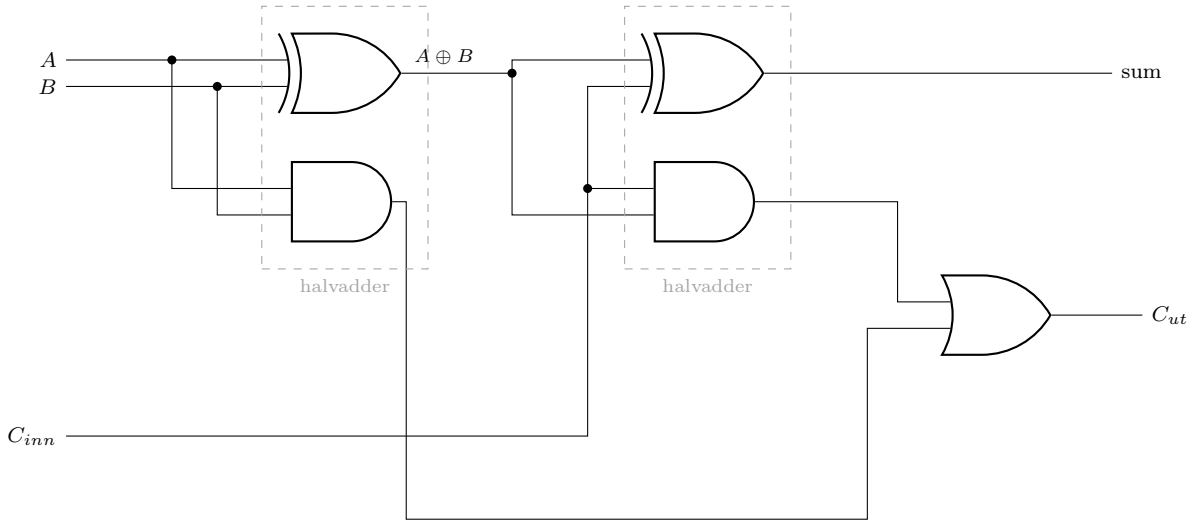
Kapittel 4 sa at vi ikke åpnet ALU-en. Nå er den åpen, og det som lå inni var to formler:

$$\text{Sum} = A \oplus B \oplus C_{inn} \qquad C_{ut} = AB + AC_{inn} + BC_{inn}$$

De er ikke gitt oss av noen. Vi skrev opp hva vi ville ha i en tabell med åtte rader, og regnet oss fram til dem med reglene fra forrige seksjon.
Det er det hele kapitlet har bygget opp til. Fra «slik skal det oppføre seg» til «slik skal det bygges», uten at det krevde noe annet enn tabellen og sju regler.

5.7.6 Fulladderen, tegnet

Du kan bygge kretsen rett fra formlene. Men det finnes en penere vei: en fulladder er to halvaddere og en OR-port.



Figur 5.14 Fulladderen, bygget av to halvaddere. Den første legger sammen A og B. Den andre legger menten inn til resultatet. OR-porten fanger opp at mente kan oppstå i begge trinnene, men aldri i begge samtidig.

Logikken i oppbyggingen er den samme som når du legger sammen tre tall på papir: ta to av dem først, legg så det tredje til summen.

Den første halvadderen regner $A+B$. Den andre legger C_{inn} til det. Mente kan oppstå i begge trinnene, og OR-porten slipper den videre uansett hvilket av dem det var.

Note-to-myself

Hvorfor holder det med OR, når det kunne vært to menter å legge sammen?
Fordi de aldri oppstår samtidig. Gir den første halvadderen mente, var både A og B lik 1, og da er summen deres 0 — og en halvadder med 0 som inngang kan ikke gi mente. Situasjonen «mente fra begge» finnes ikke.
Regn gjennom alle åtte radene, så ser du at det aldri inntreffer.

Kretsen bruker fem porter: to XOR, to AND og en OR. Det er prisen for å legge sammen én bit.

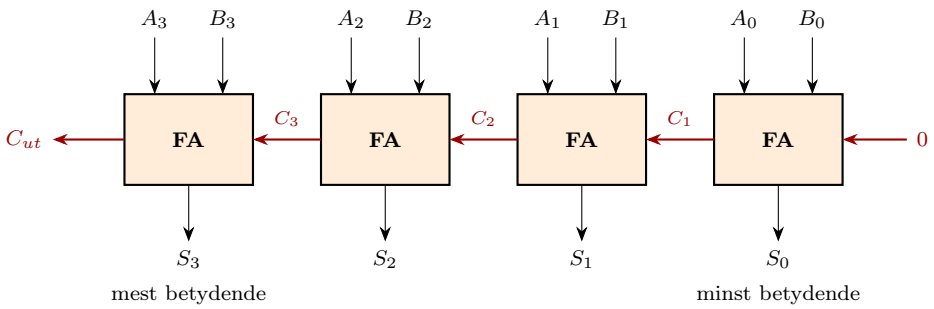
5.8 Mentekjeden

Én fulladder legger sammen én bit. Et tall har flere.

Resten av arbeidet er kopiering: sett én fulladder per bitposisjon, og koble menten fra hver av dem inn i den neste.

5.8.1 Fire på rad

Her er fire bit. Hver boks er en hel fulladder, med de fem portene fra forrige figur inni.



Figur 5.15 Fire fulladdere i rekke. Menten går mot venstre, samme vei som når du legger sammen på papir. Den første får 0 inn, siden det ikke finnes noen kolonne til høyre for den.

Legg merke til retningen. Bitene står slik tall skrives, med det minst betydende helt til høyre, og menten beveger seg mot venstre. Det er nøyaktig samme vei som blyanten din går når du legger sammen i margen.

Kretsen har et navn, eller rettere sagt to. På engelsk heter den *ripple carry adder*, fordi menten «rippler», altså bølger, gjennom rekken. På norsk sier mange *seriell adder*. Vi kaller den **mentekjeden** når vi snakker om problemet, siden det er kjeden som er poenget.

5.8.2 Hvorfor bare det første svaret er riktig med en gang

Her kommer det som gjør denne kretsen interessant.

Slå på alle inngangene samtidig. Hver fulladder begynner å regne i samme øyeblikk. Og hver av dem gir ut et svar nesten med én gang.

Problemet er at de fleste av svarene er feil.

Den høyre fulladderen får 0 som mente inn, og den vet det fra start. Den kan regne ferdig. S_0 er riktig.

Den neste venter på C_1. Den har riktignok noe på menteinngangen sin hele tiden, men det er en verdi som ennå ikke har satt seg. S_1 er bare riktig etter at C_1 er ferdig beregnet.

Og slik fortsetter det. Hver posisjon må vente på den til høyre.

Tankevekker

Dette er dominoeffekten, bokstavelig talt. Den første brikken faller med en gang. Den siste faller ikke før alle de foran har falt.
Og det er ikke bare en treghet. Det er en *feilkilde* hvis du leser av for tidlig. Ser du på utgangene et øyeblikk etter at inngangene ble satt, finner du et tall som ser helt legitimt ut og som er galt. Ingenting sier fra.
Derfor må maskinen vente et fast antall klokkeslag før den bruker svaret. Kapittel 4 sa at klokka er stilt etter det tregeste steget. Mentekjeden er ofte nettopp det steget.

5.8.3 Hva det koster

Si at én fulladder bruker tiden t på å produsere menten sin. Da bruker rekken omtrent

$$T \approx n \cdot t$$

der n er antall bit.

Tiden vokser altså rett proporsjonalt med bredden. Dobler du fra 32 til 64 bit, dobler du ventetiden.

Note-to-myself

«Omtrent» står der med vilje. Den siste fulladderen i rekken trenger bare å produsere summen sin, ikke en ny mente som noen skal vente på. Den eksakte veien er derfor litt kortere enn $n \cdot t$. Formen på svaret er det som betyr noe: tiden vokser *lineært* med antall bit. Det er den egenskapen som gjør konstruksjonen til et problem når tallene blir brede.

Det verste tilfellet er lett å lage. Legg sammen 0111 og 0001:

```
  0 1 1 1
+ 0 0 0 1
---------
  1 0 0 0
```

Menten som oppstår helt til høyre må videre gjennom hver eneste posisjon. Ingen av dem kan gjøre seg ferdig før naboen er det. Hele kjeden må gjennomløpes.

Nå kan vi også svare på noe kapittel 4 bare konstaterte: hvorfor en ELLER mellom to 64-bits tall kan gå fortere enn en addisjon av de samme to tallene.

En ELLER har ingen kjede. Hver bitposisjon har sin egen OR-port, og ingen av dem trenger å vite hva naboen fant ut. Alle 64 blir ferdige samtidig. Addisjonen må vente på menten, posisjon for posisjon.

Tenk om...

... du skulle gjøre addisjonen raskere. Hva ville du gjort?
Den opplagte ideen er å finne ut hva menten blir *før* den rekker fram. Og det går an: om A_0 og B_0 begge er 1, vet du at det blir mente ut derfra, uten å vente på noen.
Den tanken, satt i system, heter *carry look-ahead*, og den er det moderne prosessorer bruker. Prisen er mange flere porter og mer strøm, for å slippe å vente. Det er den samme avveiningen som har gått igjen i hele boka: plass mot tid.

5.8.4 Her kommer C-flagget fra

Kapittel 4 innførte fire flagg og forklarte hva de betyr. Tabell 4.4 på side 93 står der fortsatt. Det eneste som manglet, var hvor de fysisk kommer fra.

For to av dem er svaret i figur 5.15 på forrige side.

C-flagget er mente ut av den venstre fulladderen. Det er ingenting mer enn den ene ledningen som stikker ut til venstre i figuren, og som det ikke finnes noen flere fulladdere til å ta imot.

Det er hele mekanismen. Når det ble mente ut av den øverste posisjonen, var svaret for stort til bitbredden, og den ledningen går rett inn i statusregisteret.

5.8.5 Og her kommer V-flagget fra

Overflyt er litt mer subtilt, og det er en fin liten krets.

V skal settes når svaret er feil *som fortegnstall*. Kapittel 2 forklarte toerkomplement, og kapittel 4 forklarte hvorfor mente og overflyt ikke er det samme. Spørsmålet her er bare hvordan kretsen oppdager det.

Svaret er: sammenlign menten *inn i* den øverste posisjonen med menten *ut av* den.

$$V = C_{inn,\,\text{øverst}} \oplus C_{ut,\,\text{øverst}}$$

Er de like, gikk det bra. Er de forskjellige, er det overflyt.

Hele overflytdeteksjonen er altså **én XOR-port**, koblet på de to menteledningene rundt den venstre fulladderen.

La oss kontrollere det på eksempelet fra selv-testen i kapittel 4: $-5 + (-6)$ i fire bit. I toerkomplement er $-5 = 1011$ og $-6 = 1010$.

Posisjon	A	B	mente inn	sum	mente ut
0 (høyre)	1	0	0	1	0
1	1	1	0	0	1
2	0	0	1	1	0
3 (venstre)	1	1	0	0	1

Tabell 5.9 $1011 + 1010$, regnet posisjon for posisjon. Summen blir 0101.

Øverste posisjon har mente inn $= 0$ og mente ut $= 1$. De er forskjellige, så $V = 1$. Og $C = 1$, siden det ble mente ut.

Svaret 0101 er $+5$. Vi la sammen to negative tall og fikk et positivt. Flagget hadde rett.

Note-to-myself

Legg merke til hvor lite det kostet. Fire fulladdere, og så én XOR-port på toppen, så har maskinen et pålitelig varsel om at svaret er galt.
Det som er dyrt, er ikke å oppdage overflyt. Det er å gjøre noe med det. Kretsen sier fra; om noen hører etter, er opp til programmet. Det var poenget i kapittel 2, og nå ser du at advarselen bokstavelig talt henger der på en ledning.

5.8.6 Subtraksjon, gratis

Kapittel 2 lovet at en maskin ikke trenger en egen subtraksjonskrets, fordi $a - b$ kan regnes som $a + (-b)$. Og i toerkomplement lages $-b$ ved å invertere alle bitene i b og legge til 1.

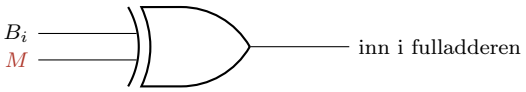
Begge delene har vi allerede.

Å invertere hver bit i b er én NOT-port per posisjon. Å legge til 1 er å sette menten inn til den høyre fulladderen til 1 i stedet for 0 — og den inngangen står der allerede, ubrukt, i figur 5.15 på side 132.

Det siste trinnet er elegant. Vi vil kunne velge mellom addisjon og subtraksjon, så vi trenger en bryter. Bytt hver NOT-port ut med en XOR-port, der den ene inngangen er bitet fra b og den andre er en styreledning vi kaller M.

- Er $M = 0$, gir XOR-porten bitet uendret ut. Og vi setter menten inn til 0. Kretsen legger sammen.
- Er $M = 1$, gir XOR-porten det inverterte bitet ut. Og vi setter menten inn til 1. Kretsen trekker fra.

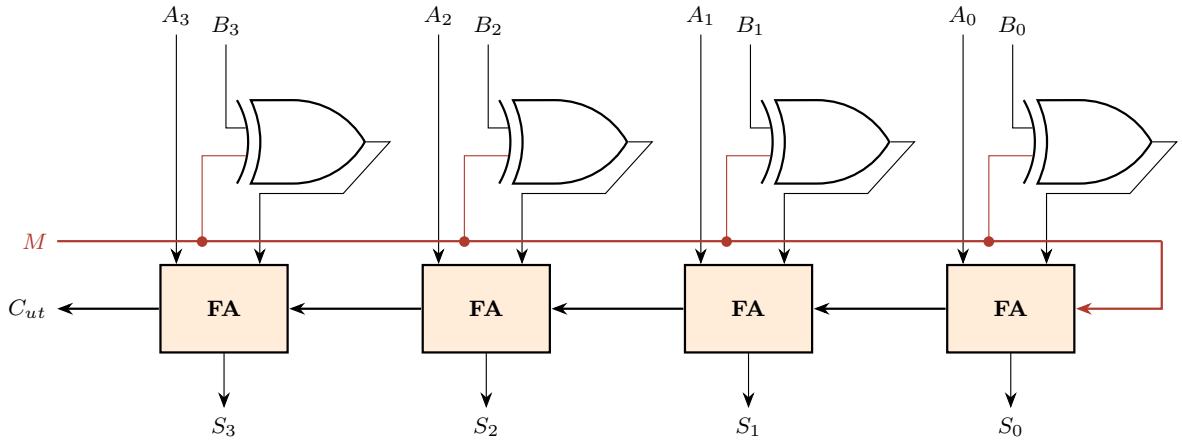
Samme ledning, M, styrer begge delene. Én bit avgjør om kretsen er en adder eller en subtraher.



M	B_i	ut av XOR
0	0	0 (uendret)
0	1	1 (uendret)
1	0	1 (snudd)
1	1	0 (snudd)

Figur 5.16 XOR-porten brukt som *styrt* inverter. Står M på 0, slipper den B_i rett gjennom. Står M på 1, snur den. Det er hele trikset.

Og slik ser hele kretsen ut når den er satt sammen.



Figur 5.17 Den samme mentekjeden som før, med én XOR-port lagt til foran hver B-inngang. Den røde ledningen M gjør to ting samtidig: den forteller alle XOR-portene om de skal snu, og den går helt til høyre og inn som mente til den første fulladderen. Står M på 0, legger kretsen sammen. Står den på 1, trekker den fra.

Tell hva som er lagt til: fire XOR-porter og én ledning. Det er hele prisen for at maskinen også kan trekke fra.

Advarsel :)

Når kretsen trekker fra, betyr C-flagget noe annet.
Ved addisjon er mente ut av den øverste posisjonen et varsel om at svaret ble for stort. Ved subtraksjon er den samme ledningen 1 nøyaktig når $a \geq b$, altså når regnestykket *ikke* trengte å låne.
Det er samme ledning og samme krets, men motsatt fortolkning. Prosessoren vet hvilken av dem

som gjelder, fordi den vet hva M sto på.

Tankevekker

Nå lukker en sirkel som har vært åpen i tre kapitler.
Kapittel 2 valgte toerkomplement, og begrunnelsen var at addisjon da virker uendret for negative tall. Kapittel 4 sa at ALU-en slipper unna uten egen subtraksjonskrets. Her er regningen: det koster *én XOR-port per bit*, og en styreledning som allerede måtte finnes.
Og styreledningen M er ikke en hvilken som helst ledning. Den kommer fra opkoden i instruksjonsordet, via kontrollenheten. Skriver du `SUB` i stedet for `ADD`, er forskjellen til slutt at én ledning inne i ALU-en blir 1 i stedet for 0.

5.9 Kretser som velger

Vi har bygget kretser som regner. Nå to som *velger* — og den første er det tredje løftet fra kapittel 4.

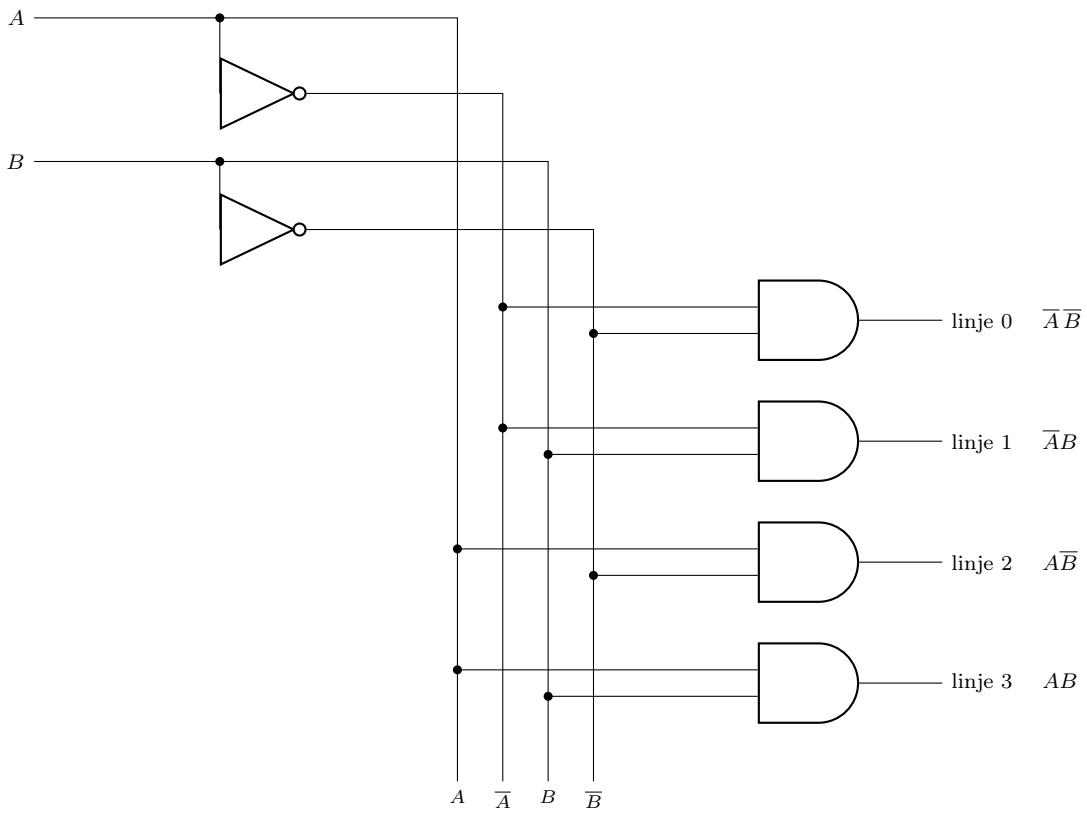
5.9.1 Dekoderen

Kapittel 4 definerte en **dekoder** som en krets med noen få innganger og én utgang per mulig kombinasjon, der nøyaktig én utgang er på til enhver tid. Figur 4.6 på side 95 tegnet den med fire innganger og seksten utganger.

Spørsmålet var hvordan den bygges. Svaret har du allerede skrevet, uten å vite det.

Gå tilbake til seksjon 5.5. Der skrev vi ett ledd per rad i tabellen, og hvert ledd var en AND mellom alle inngangene, invertert der raden hadde 0.

En dekoder er alle de leddene på én gang, hvert med sin egen utgang.



Figur 5.18 En dekoder med to innganger og fire utganger. De fire loddrette ledningene bærer A, $\overline{A}$, B og $\overline{B}$. Hver AND-port henter fra to av dem, og svarer dermed til én rad i sannhetsverditabellen.

Sett $A = 1$ og $B = 0$. Da er $A = 1$, $\overline{A} = 0$, $B = 0$, $\overline{B} = 1$.

Porten for linje 2 henter fra A-bussen og $\overline{B}$-bussen, altså 1 og 1. Den slår ut.

De tre andre får minst én null inn, og en AND med en null er alltid 0. De er mørke.

Nøyaktig én utgang er på. Det var definisjonen, og nå ser du hvorfor den alltid holder: hver port er skreddersydd for sin ene kombinasjon.

5.9.2 Opkoden inn, styresignalet ut

Utvid figuren til fire innganger, så har du seksten AND-porter med fire innganger hver. Det er figuren fra kapittel 4.

Og nå kan vi følge hele veien.

Kapittel 3 skrev `LDA 42`. Kapittel 4 gjorde det om til bitmønsteret 0101 00101010, der de fire første bitene er opkoden 0101.

De fire bitene går inn i dekoderen. Én AND-port har nettopp mønsteret $\overline{b_3} \cdot b_2 \cdot \overline{b_1} \cdot b_0$, og den får fire ettall inn. Den slår ut. Alle de femten andre er mørke.

Utgangen fra den ene porten er `LDA`-linja. Derfra går den til bryterne i datastien og åpner akkurat de veiene som skal åpnes.

Tankevekker

Det er hele koblingen mellom et program og en krets, og den er ikke lenger enn dette.
Du skrev en ordre. Assembleren gjorde den til et tall. Tallet er et bitmønster. Fire av bitene traff én AND-port, som slo ut. Den porten er grunnen til at maskinen gjør `LDA` og ikke `ADD`.
Det er ingen tolkning noe sted. Ingen leser instruksjonen og skjønner hva den betyr. Det er en port som får fire ettall inn.

5.9.3 Multiplekseren

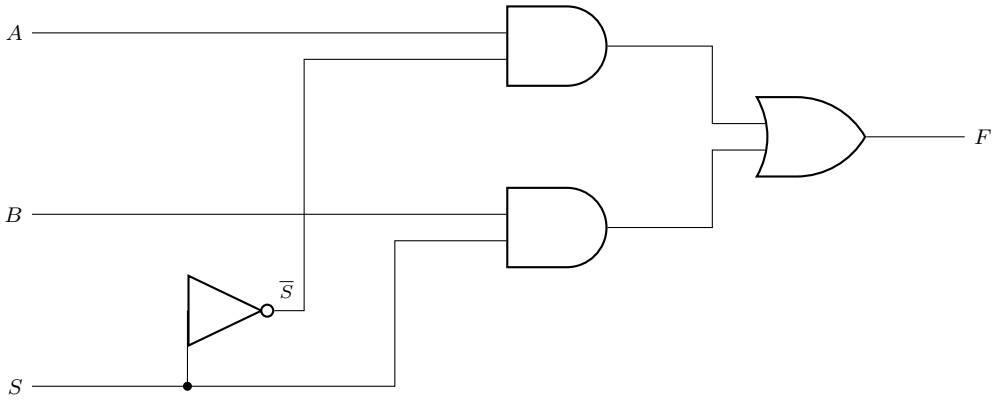
Den motsatte kretsen heter **multiplekser**, ofte forkortet mux. En dekoder tar et tall og velger én linje. En multiplekser tar flere linjer og velger ut én av dem.

Med to datainnganger og én styreinngang blir uttrykket:

$$F = \overline{S} \cdot A + S \cdot B$$

Er $S = 0$, faller andre ledd bort og F blir A. Er $S = 1$, faller første ledd bort og F blir B.

Det er en bryter, bygget av to AND-porter, en OR-port og en inverter.



Figur 5.19 En multiplekser med to datainnganger. Styresignalet S åpner den nedre AND-porten og stenger den øvre, eller omvendt. Nøyaktig én av de to slipper noe gjennom til OR-porten.

Legg merke til at det er samme idé som i dekoderen: en AND-port brukes som en grind, der det ene signalet bestemmer om det andre får passere.

Note-to-myself

Multipleksere er overalt inne i en prosessor, selv om de sjelden tegnes. Hver gang datastien har to mulige kilder til samme sted, sitter det en multiplekser der og velger.
Et eksempel du har sett: subtraksjonskretsen i forrige seksjon valgte mellom b og $\overline{b}$. Vi løste det med en XOR-port, som er billigere her, men en multiplekser ville gjort samme jobb.

5.10 Kretsen som husker

Alt vi har bygget til nå har én egenskap felles: utgangen følger inngangene, hele tiden. Endrer du en inngang, endrer utgangen seg straks etter. Kretsen har ingen fortid.

Slike kretser heter **kombinatoriske**. Alle portene, adderen, dekoderen og multiplekseren er kombinatoriske.

Men kapittel 4 sa at et register holder på en verdi så lenge strømmen står på. Det kan ingen kombinatorisk krets gjøre. Ta bort inngangen, og svaret forsvinner.

Så hvordan kan noe huske?

Ideen er én setning: **koble utgangen tilbake til inngangen.**

Da er kretsen ikke lenger bare avhengig av hva som kommer inn nå. Den er også avhengig av hva den selv sa sist. Og om tilbakekoblingen er satt opp riktig, kan den holde seg selv i ro i den tilstanden den er i, helt til noen med vilje dytter den over i den andre.

To NOR-porter som mater hverandre, holder nøyaktig én bit på denne måten. Registrene i kapittel 4 er bygget av nettopp dette, og det samme er cachen.

Kretser med tilbakekobling heter **sekvensielle**, fordi utgangen avhenger av rekkefølgen på det som har skjedd, og ikke bare av hva som står på inngangene nå. Alt annet i dette kapitlet har vært kombinatorisk.

Her stopper vi. Men historien stopper ikke, og resten ligger rett bak i boka.

Vil du se resten? Appendiks C

Appendiks C, «Kretser som husker», tar denne ene siden videre til seks. Veien dit går slik:

- **SR-latchen.** De to NOR-portene tegnet opp, og en gjennomgang av hvorfor de blir stående i ro — og hva som skjer i den ene kombinasjonen som er forbudt.
- **D-latchen.** Hvordan man gjør det umulig å havne i den forbudte tilstanden, og hvordan man bestemmer *når* kretsen skal lytte.
- **Vippa.** Forskjellen på en dør som står åpen og en som smeller igjen i et bestemt øyeblikk. Dette er byggeklossen alt synkront utstyr er laget av.
- **Registeret.** Åtte vipper med felles klokkeledning — og dermed svaret på hvorfor kapittel 4 sa at registre er raske, dyre og få.

Det er ikke pensum i IN1020, og du kan hoppe over det uten å miste noe du blir prøvd i. Men det er der løftet fra kapittel 4 blir innfridd helt, og det er en fin halvtime hvis du vil se hvordan minne egentlig virker.

5.11 Oppsummering

Kapitlet begynte med to brytere i en trappeoppgang og endte med en krets som kan huske. Imellom har vi bygget noe av det maskinen faktisk består av, av deler som bare kan si ja og nei.

5.11.1 Hva som egentlig var nytt

Begrep	Kort sagt
Logisk port	Krets med én utgang som utfører én logisk operasjon. Sju typer.
Logisk funksjon	*Hva* som skal skje. Kan skrives som uttrykk eller tabell.
Krets	*Hvordan* det skjer. Samme funksjon kan bygges på mange måter.
Sannhetsverditabell	Alle 2^n kombinasjoner og hva utgangen blir. Uttømmende.
Sum av produkter	Ett AND-ledd per rad med 1 på utgangen, bundet sammen med ELLER.
Absorpsjon	$\overline{X} + XY = \overline{X} + Y$. Regelen som oftest blir glemt.
De Morgans lover	Overstrek inn gjennom en parentes snur operatoren.
Halvadder	To bit inn, sum og mente ut. XOR og AND. Mangler mente inn.
Fulladder	Tre bit inn, sum og mente ut. Fem porter. Byggeklossen i en ALU.
Mentekjede	Fulladdere i rekke. Tiden vokser lineært med antall bit.
C-flagget	Mente ut av den øverste posisjonen. Én ledning.
V-flagget	XOR av mente inn og mente ut i øverste posisjon. Én port.
Dekoder	Én AND-port per kombinasjon. Nøyaktig én utgang på om gangen.
Multiplekser	Velger én av flere innganger. $F = \overline{S}A + SB$.
Kombinatorisk krets	Utgangen følger bare inngangene. Ingen fortid.
Sekvensiell krets	Har tilbakekobling, og dermed hukommelse.

Tabell 5.10 Begrepene fra dette kapitlet.

Den gjennomgående ideen kom igjen fire ganger. Tre porter ble til én NOR. Fire ledd og åtte porter ble til én AND og én XNOR. Overflytdeteksjonen ble til én XOR. Subtraksjon ble til én XOR per bit og en styreledning som allerede fantes.

Hver gang var formen den samme: det du først skriver ned, er ikke det du til slutt bygger. Mellom dem ligger et lite sett regler og litt tålmodighet.

Og hver gang kom det noe mer enn sparte porter. Den forenklede kretsen lot seg lese. $\overline{A}+B$ er «hvis A, så B». $\overline{C \oplus AB}$ er «er C lik OG-en av A og B?». Det var ikke mulig å se i det opprinnelige uttrykket.

5.11.2 Videre til neste kapittel

Kapittel 4 pekte tre ganger nedover. ALU-en er åpnet, dekoderen er bygget, og hukommelse viste seg å være tilbakekobling. Den siste fikk bare ideen her; selve kretsene står i appendiks C.

Men det står igjen en ting, og den er ubehagelig.

Portene finnes ikke.

Dette er ingen spissfindighet. Inne i prosessoren din finnes det ikke én eneste AND-port. Det du finner hvis du går ned ett lag til, er transistorer — små brytere av halvleder — og noen av dem er koblet sammen slik at gruppen oppfører seg som en AND.

Porten er et symbol vi har funnet opp fordi det er umulig å tegne en milliard brytere. Den er en abstraksjon, nøyaktig som de kapittel 1 handlet om, og den er like nyttig og like uvirkelig som alle de andre.

Får du et kretsskjema med porter, kan du lese det. Får du det samme skjemaet tegnet med transistorer, er det uleselig. Det er hele grunnen til at porten finnes.

Kapittel 6 tar det siste steget ned, til bryteren selv, og til strømmen og spenningen som gjør at den er en bryter.

Det er bunnen. Lenger ned går vi ikke.

5.12 Videre lesning

Englander dekker ikke dette stoffet i boka. Det er verdt å si rett ut, siden pensumboka har vært hovedkilden i alle tidligere kapitler. Boolsk algebra og digital logikk ligger i *Supplementary Chapter 1*, som forlaget legger ut på nett og ikke trykker. Finner du den, dekker den det samme som dette kapitlet, med flere eksempler og mer sekvensiell logikk.

Harris & Harris, *Digital Design and Computer Architecture*, kapittel 2 og 5. Den beste neste boka hvis du vil videre. Kapittel 2 tar porter, boolsk algebra og Karnaugh grundig. Kapittel 5 bygger addere, multiplikatorer og en hel ALU. Vær klar over at den bruker andre portsymboler enkelte steder, og at den går raskt fram.

Nand2Tetris, fritt tilgjengelig på nett. Et kurs der du bygger en hel datamaskin fra NAND-porter og oppover, i en simulator. Du skriver ingen fysiske kretser, men du må løse hvert trinn selv. Det er den beste måten å oppdage at det faktisk henger sammen.

Appendiks B tar forenkling systematisk med Karnaugh-diagram, og **appendiks C** tar sekvensiell logikk fra tilbakekobling til ferdig register. Ingen av dem er pensum.

Videre i denne boka: kapittel 6 viser hva en port er bygget av. Appendiks D forklarer hvorfor alt må skje i takt med en klokke, og det blir først helt tydelig etter at du har sett mentekjeden.

5.13 Refleksjon

1. Vi så at tre porter kunne erstattes av én, og at åtte kunne erstattes av to. Likevel bygger ingen kretser for hånd lenger; det gjøres av programvare. Hva er da vitsen med å kunne forenkle selv?
2. En mentekjede på 64 bit er treg, men enkel og liten. En carry look-ahead er rask, men stor og strømsulten. En designer må velge. Hvilke opplysninger om bruken trenger hun for å velge riktig?
3. Overflytdeteksjonen koster én port. Likevel har alvorlige feil oppstått fordi ingen sjekket flagget. Hvis varselet er nesten gratis å produsere, hvorfor blir det ikke sjekket?
4. Porter finnes ikke, sa vi. De er et abstraksjonsnivå vi har funnet opp for å kunne tenke. Finn et annet felt enn datateknologi der noe tilsvarende er sant: et begrep alle bruker og regner med, som ikke svarer til noe fysisk.

5.14 Selv-test

Spørsmålene står i stigende vanskegrad, og fasiten finnes bakerst i boka.

1. Forklar med egne ord forskjellen på en logisk funksjon og en krets.
2. Hvor mange rader har en sannhetsverditabell med fire innganger? Og med seks?
3. NOT-porten har én inngang. Hvor mange innganger tegner vi de seks portene i tabell 5.1 på side 116 med, og hvorfor er ikke det en grense for hva de kan ha?
4. Hvilken port gir 1 bare når de to inngangene er forskjellige? Og hvilken gir 1 bare når de er like?
5. Skriv sannhetsverditabellen for en NAND-port.
6. Sett $A = 1$ og $B = 1$. Hva gir hver av de seks portene i tabell 5.1 på side 116?
7. Hvorfor er $1 + 1 = 1$ i boolsk algebra, når det ikke er det i vanlig regning?
8. Hva er forskjellen på $\overline{A}\,\overline{B}$ og $\overline{A \cdot B}$? Gi et par inngangsverdier der de to gir ulikt svar.
9. Tegn kretsen for $F = \overline{A}B + A\overline{B}$ med grunnleggende porter. Hvilken enkeltport gjør nøyaktig det samme?
10. En krets har denne tabellen. Skriv funksjonsuttrykket på sum-av-produkter-form, og forenkle det så langt du kommer.

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

11. Sett opp sannhetsverditabellen for $F = \overline{A} + B$.
12. Forenkle $F = AB + A\overline{B}$.
13. Forenkle $F = \overline{A} + AB$. Hvilken regel brukte du?
14. Forenkle $F = ABC + AB\overline{C}$.
15. Forenkle $F = AB + AC + A\overline{B}\,\overline{C}$. Svaret er kortere enn du tror.
16. Bruk De Morgans lov på $\overline{A + \overline{B}}$, og forenkle.
17. I hvor mange rader av en tabell med tre innganger er leddet $\overline{A}\,\overline{B}\,\overline{C}$ lik 1? Hvilken rad er det?
18. En krets gir $1, 0, 0, 1$ for radene 00, 01, 10, 11. Hvilken port er det, og hva blir uttrykket på sum-av-produkter-form?
19. **Finn feilen.** En student har en krets med en inverter på A, en inverter på B, og begge inn i en OR-port. Studenten skriver $F = \overline{A + B}$. Hva er det riktige uttrykket, og hvilken enkeltport svarer studentens uttrykk til?
20. Hvorfor heter halvadderen «halv»?
21. Hvor mange porter er det i en fulladder bygget av to halvaddere og en OR-port, og hvilke typer?
22. Hvorfor holder det med én OR-port for å lage mente ut i fulladderen, i stedet for en ny adder?
23. Skriv sum-av-produkter-uttrykket for mente ut i fulladderen ut fra tabell 5.8 på side 129, og forenkle det til tre ledd.
24. En 4-bits mentekjede legger sammen 0110 og 0101. Sett opp mente inn og mente ut for hver av de fire posisjonene. Hva blir summen, og hva blir C og V?

25. En mentekjede på 8 bit bruker omtrent 8 nanosekunder. Hvor lang tid bruker en på 32 bit, omtrent, og hvorfor?
26. Hvorfor kan en ELLER mellom to 64-bits tall gå raskere enn en addisjon av de samme to tallene?
27. Hvor mange utganger har en dekoder med tre innganger? Hvor mange AND-porter og hvor mange invertere trengs?
28. Opkoden 0101 kommer inn i en dekoder med fire innganger. Hvilken linje slår ut, og hvilke fire signaler er det AND-porten for den linja får inn?
29. Hva skiller en kombinatorisk krets fra en sekvensiell? Gi ett eksempel på hver fra dette kapitlet.
30. I subtraksjonskretsen styrer ledningen M både XOR-portene og menten inn til den første full-adderen. Hva ville kretsen regnet ut hvis du satte $M = 1$ på XOR-portene, men glemte å sette menten inn til 1?