Kapittel 6

Transistor, strøm og spenning

Innhold

6.1 Læringsmål . . . 146
6.2 Bryteren som ikke trenger en finger . . . 147
6.3 Strøm, spenning og jord . . . 149
6.4 Inverteren: den første porten vi faktisk bygger . . . 151
6.5 NAND, NOR, og hvorfor AND er dyrere . . . 153
6.6 Null og ett er områder, ikke tall . . . 156
6.7 Det tar tid . . . 158
6.8 Den lengste veien . . . 160
6.9 Klokka . . . 162
6.10 Hvorfor klokka stoppet på rundt 3 GHz . . . 165
6.11 Oppsummering . . . 169
6.12 Videre lesning . . . 171
6.13 Refleksjon . . . 171
6.14 Selv-test . . . 172

Telefonen din blir varm.

Du har kjent det. Etter en halvtimes spilling eller en lang videosamtale er baksiden lun, og har du laptop, hører du viften spinne opp og bli stående og suse. Det er litt rart når du tenker etter. Det beveger seg ingenting der inne: ingen motor, ingen tannhjul, ingenting som gnisser mot noe annet. Prosessoren i en telefon har ikke én eneste bevegelig del.

Hva er det da som blir varmt?

Du har ikke svaret ennå, og det er helt greit. Det ligger et lag lenger ned enn vi har vært så langt.

Tankevekker

Varmen er ikke en bieffekt vi kunne konstruert oss bort fra. Den *er* regningen for arbeidet. Hver gang maskinen bytter et bit fra 0 til 1, tar den en liten slurk strøm, og hele slurken ender som varme.
En moderne prosessor gjør det noen milliarder ganger i sekundet, på noen milliarder steder samtidig. Da blir slurkene til et måltid.

Vi kommer hit med en gjeld. Kapittel 5 endte med en ubehagelig setning: portene finnes ikke. Du brukte et helt kapittel på å bygge med AND, OR og NOT, og så viste det seg at ingen av dem sitter

i maskinen. De er tegninger, symboler vi har funnet på fordi ingen orker å tegne en milliard brytere. Da gjenstår spørsmålet om hva som *faktisk* sitter der.

Det du leter etter, kjenner du allerede. Du brukte det gjennom hele kapittel 5, og du bruker det hver morgen når du slår på lyset.

Du trenger ingen fysikk for å lese dette kapitlet: ingen elektronikk, ingen kjemi, ingenting om atomer. Vi snakker om strøm hele veien, men vi gjør det med vann. Vannbildet er ikke helt presist, men det bærer deg lenger enn du skulle tro, og akkurat så langt som dette faget går.

Vi regner litt. Det er bare gange og dele med runde tall, og svaret står rett ved siden av, så du kan lese videre selv om du ikke orker å regne etter. Det eneste du må ha med deg fra før, er at «pico» og «nano» er navn på svært små tall, og du får en tabell over dem når vi kommer dit. Formler får du lite av. Den du får mest bruk for, kobler klokkeperiode til frekvens. En til, om hvorfor brikker blir varme, står i en egen boks som du kan hoppe over.

Dette kapitlet handler om et lag du aldri har tenkt på, så det kommer til å føles fremmed de første sidene. Det er normalt, og det sier ingenting om hvor flink du er. Det går fortere hvis du leser rolig og har et ark ved siden av deg.

Hva du sitter igjen med

Når du er ferdig med kapitlet, kan du svare på tre spørsmål som de fleste ikke kan svare på.
Hvorfor blir en datamaskin varm, når ingenting beveger seg i den?
Hvorfor sluttet prosessorer å bli raskere rundt 2004, etter å ha blitt raskere hvert eneste år i tretti år?
Og hvorfor har maskinen din åtte kjerner i stedet for én kjerne som går åtte ganger så fort?
Alle tre har det samme svaret, og du finner det i bunnen av stabelen.

Her er vi nemlig nede. Kapittel 1 tegnet opp lagene, og siden har vi gått nedover, ett lag om gangen. Nå stopper vi. Under transistoren ligger halvlederfysikken, som hører hjemme i et annet fag, og fra neste kapittel går veien oppover igjen.

6.1 Læringsmål

Når du har arbeidet deg gjennom kapitlet, skal du kunne:

1. **Forklare hva en transistor gjør.** Den er en bryter som styres av et elektrisk signal i stedet for av en finger. Resten av kapitlet er følger av det.
2. **Kjenne igjen de to typene og si hva som skiller dem.** Den ene leder når styresignalet er 1, den andre når det er 0.
3. **Forklare hvorfor en port trenger strøm.** Og hvor nullene og ettallene egentlig kommer fra, som ikke er der du tror.
4. **Lese en inverter bygget av to brytere.** Du skal kunne si hva utgangen blir i begge tilstandene, uten å pugge noe.
5. **Telle transistorer i en port og forklare hvorfor NAND er billigere enn AND.** Her får du svar på et spørsmål kapittel 5 stilte og ikke besvarte.
6. **Forklare hvorfor 0 og 1 er spenningsområder og ikke tall.** Det er også grunnen til at digitale signaler tåler støy og analoge ikke gjør det.
7. **Regne ut hvor lang tid en krets bruker.** Finn den lengste veien gjennom kretsen, tell portene og gang med portforsinkelsen.

8. **Lese et klokkesignal og regne mellom periode og frekvens.** Av alle målene her er det dette du kommer til å bruke mest.
9. **Forklare hvorfor klokkefrekvensen sluttet å øke.** Og hvorfor svaret ble flere kjerner i stedet for raskere kjerner.

6.2 Bryteren som ikke trenger en finger

6.2.1 Noen må trykke på bryteren

Kapittel 5 begynte med to brytere og en lyspære, og alt som kom etterpå hvilte på dem. Brytere i serie ble OG, brytere i parallell ble ELLER, og derfra bygde vi oss helt opp til en krets som legger sammen tall.

Historien hadde ett hull, og vi lot det ligge: noen må trykke på bryterne. En bryter er et stykke metall som enten berører et annet stykke metall eller ikke, og for å flytte det må noe ta i. En finger, en fjær, en knapp. En datamaskin har ingen fingre.

Det verste er ikke engang det. Se på en hvilken som helst krets fra forrige kapittel. Utgangen fra én port går rett inn som inngang på den neste, og da er det utgangen fra den første porten som må trykke på bryterne i den andre. Vi trenger altså en bryter som et signal kan trykke på.

Hvorfor dette er det eneste som mangler

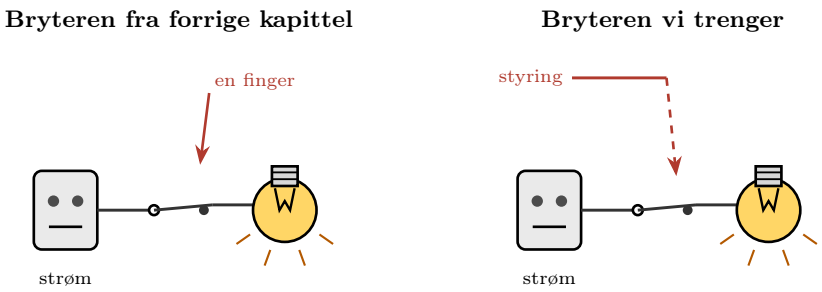
Alt annet i kapittel 5 fungerer allerede med vanlige brytere. Du kan bygge en fulladder av lysbrytere i kjelleren, koble den til et batteri, og den regner riktig. Folk har gjort det.
Den ville bare vært ubrukelig, fordi noen måtte stått der og trykket på bryterne i riktig rekkefølge. Akkurat den jobben skal vi bli kvitt.

6.2.2 En tredje ledning

Tenk deg bryteren fra forrige kapittel en gang til: to kontakter og et metallstykke som enten forbinder dem eller ikke. Gi den nå en tredje ledning, en som strømmen ikke går gjennom, men som bare *bestemmer*.

Vi sier at det er **spenning** på en ledning når den er koblet til strømkilden, og ingen spenning når den er koblet til jord. Det er de to tilstandene vi skriver som **1** og **0**. Hva spenning egentlig er, tar vi skikkelig i seksjon 6.3. Foreløpig kan du tenke på det som «det er trykk på ledningen».

Er det spenning på den tredje ledningen, leder bryteren. Er det ingen spenning, er den stengt.



Figur 6.1 Til venstre bryteren fra forrige kapittel, som noen må trykke inn. Til høyre den samme bryteren med en tredje ledning: spenningen på styreledningen avgjør om den leder. Da kan utgangen fra én port styre den neste.

Ingen bevegelige deler, ingen finger, ingen knapp: et signal på én ledning avgjør om strømmen får gå på en annen. Les den setningen en gang til. Hele kapitlet hviler på den, og den er ikke åpenbar første gang man leser den.

Nå kan du koble utgangen fra én slik bryter til styreledningen på den neste, og da bygger kretsen seg videre av seg selv, uten at noen står ved siden av og trykker.

6.2.3 Og en som gjør det motsatte

Det finnes to slags. Den vi nettopp beskrev, leder når det er spenning på styreledningen. Den andre gjør det motsatte: spenning på styreledningen *stenger* den, og uten spenning leder den.

Det kan høres ut som en kuriositet, men kapitlet henger på at vi har begge to. Hvorfor, ser du i seksjon 6.4.

To ord vi holder oss unna

I elektronikk sier man gjerne at en bryter er *åpen* når den *ikke* slipper gjennom. Det er stikk motsatt av en åpen kran, og siden vi skal bruke vann som bilde hele veien, er det en felle.
Vi sier derfor **leder** og **stengt** om transistorer i denne boka, aldri åpen og lukket. Møter du «åpen bryter» i en annen bok, betyr det det samme som «stengt» her.

Kapittel 5 hadde allerede et tegn for «denne gjør det motsatte»: den lille sirkelen, boblen, på et portsymbol. Den bruker vi igjen. En bryter med boble på styreledningen er den som gjør det motsatte.

6.2.4 Den heter transistor

Nå kan den få et navn. En **transistor** er en bryter som styres av et elektrisk signal i stedet for av en finger. Den er laget av silisium, har ingen bevegelige deler og er så liten at det er plass til noen titalls milliarder av dem i en telefon.

De to typene heter **nMOS** og **pMOS**. nMOS-transistoren leder når styresignalet er 1. pMOS-transistoren leder når styresignalet er 0, og det er den med boblen.

Navnene, for ordens skyld

MOS står for *metal-oxide-semiconductor* og beskriver hvordan transistoren er bygget opp i lag. Bokstavene n og p viser til to måter å behandle silisiumet på, slik at det leder strøm på hver sin måte.
Du trenger ikke kunne noe av dette. Det du må huske, er at den ene leder ved 1 og den andre ved 0. Vil du vite hva som skjer inne i silisiumet, står det i appendiks D.

En transistor har tre tilkoblinger: styreledningen og de to som strømmen eventuelt går mellom. I engelsk faglitteratur heter styreledningen *gate*, og de to andre heter *source* og *drain*. Det kan være greit å ha sett ordene, siden de står i alt du kommer til å lese videre. I denne boka kaller vi dem styreledningen og de to kontaktene, og lar det bli med det.

6.3 Strøm, spenning og jord

6.3.1 Hvorfor en port trenger strøm

Før vi kan bygge noe av bryteren, må vi vite hva det er som går gjennom den. Og vi kan begynne med et spørsmål som er lurt å stille høyt: hvorfor må en datamaskin ha strøm?

Det virker selvsagt, men det er det ikke. En bok trenger ingen strøm for å lagre tekst, og et ferdiglagt puslespill trenger ingen strøm for å bli liggende. Hvorfor kan ikke maskinen bare stå der med nullene og ettallene sine?

Fordi et bit i en datamaskin ikke er en ting. Det er en *tilstand*, en spenning som må holdes oppe et sted i kretsen, og slipper du den, faller den. Dessuten gjør en port mer enn å holde en verdi. Den lager et nytt signal som er sterkt nok til å styre neste port, og det koster energi, på samme måte som det koster energi å løfte noe opp.

6.3.2 Spenning er høyde, strøm er mengde

Vi bruker vann hele veien her, og da trenger vi to ord.

Spenning er høydeforskjellen. Står vannet høyt, er det trykk og vannet vil nedover, mens det ikke skjer noe når det står lavt. Spenning måles i *volt*.

Strøm er hvor mye vann som faktisk renner, og det avhenger både av høydeforskjellen og av hvor mye krana er skrudd opp.

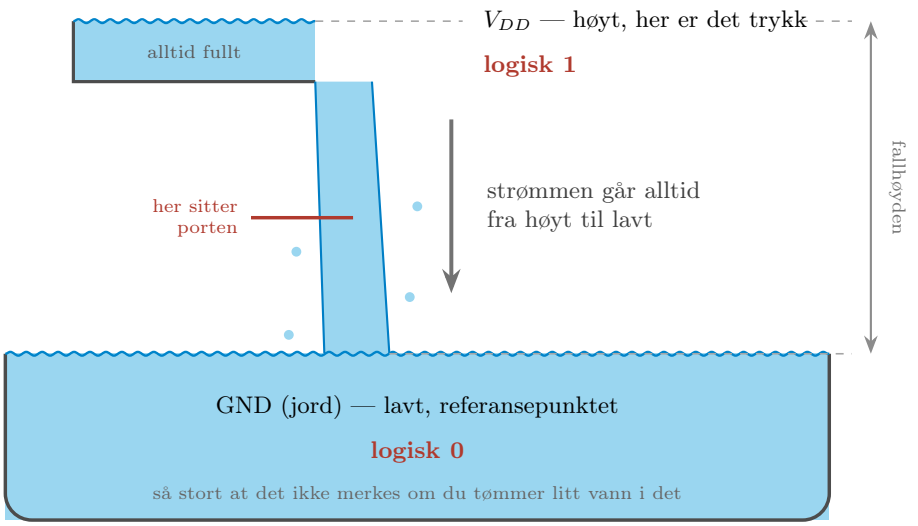
De to er forskjellige ting. Du kan ha stor høydeforskjell og likevel ingen strøm hvis krana er stengt, og akkurat den situasjonen skaper en stengt transistor.

6.3.3 Toppen og bunnen

En krets har et høyt punkt og et lavt punkt, og alt annet ligger et sted imellom.

Det høye punktet heter **VDD**. Det er forsyningsspenningen, stedet der strømmen kommer inn i kretsen. Navnet kommer av *drain supply voltage*, spenningen som mater transistorene, men i denne boka er det bare et navn på toppen. I en moderne prosessor ligger VDD rundt 0,8 volt. Av og til sier vi «1 volt» for å slippe desimaler.

Det lave punktet heter **GND**, kort for *ground*, altså jord. Det er 0 volt og referansepunktet alt annet måles mot.



Figur 6.2 V_{DD} og GND som toppen og bunnen av et fall. Strømmen går alltid nedover. Det nederste bassenget er så stort at nivået ikke rikker seg om du tømmer litt i det, og derfor kan jord brukes som en stabil nullstrek.

VDD er logisk 1, og GND er logisk 0. Mer er det ikke i oversettelsen mellom elektrisitet og logikk. Les den en gang til, for det er denne koblingen som gjør at alt du lærte i kapittel 5 faktisk kan sitte i en brikke.

Er «jord» virkelig jorda?

Spørsmålet kommer hvert år, og det er et godt et.
Svaret er: opprinnelig ja, og nå nesten. I et hus går jordlederen i stikkontakten faktisk ned i bakken. I en telefon på batteri finnes det ingen slik ledning, og «jord» er bare den ene polen på batteriet, som alt annet måles fra.
I begge tilfeller er poenget det samme. Jord er et sted som er så stort at det ikke merkes om du heller litt ladning inn i det, på samme måte som havet står like høyt etter at du har tømt et badekar i Atlanterhavet. Derfor kan jord fungere som en stabil nullstrek, og en stabil nullstrek trenger du for at «0 volt» skal bety det samme over hele brikken.

6.3.4 To tall, og ingenmannsland imellom

Én ting til før vi bygger noe. Strømmen går alltid fra VDD mot GND, aldri andre veien, så enhver ledning i kretsen har en spenning et sted mellom 0 og 0,8 volt. I prinsippet kan den ha hvilken som helst verdi i det intervallet. Logikken kjenner derimot bare to verdier. Hvordan går det i hop?

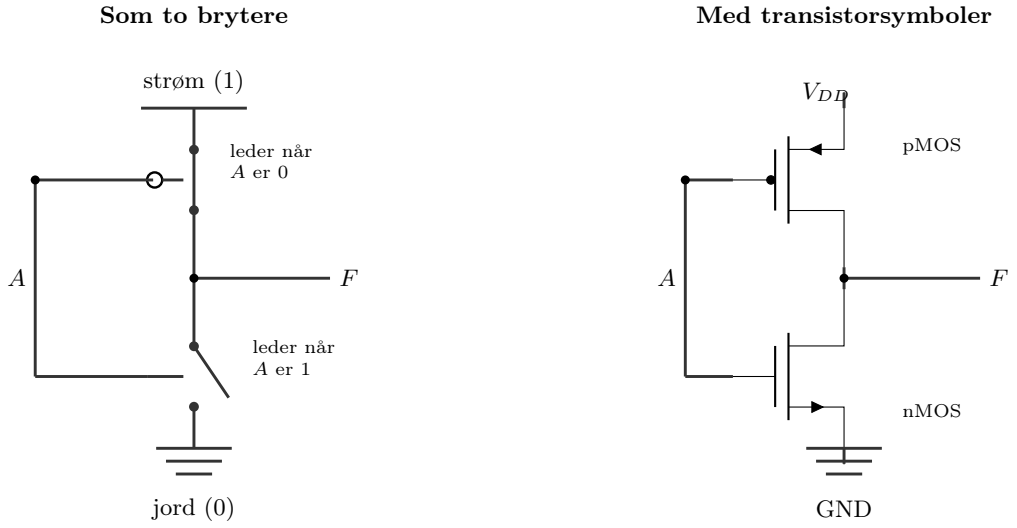
Det går i hop ved at 0 og 1 får hvert sitt *område* å være innenfor, i stedet for hvert sitt tall. Vi kommer tilbake til det i seksjon 6.6, når vi har noe konkret å vise det på. Foreløpig holder det å vite at lavt betyr 0 og høyt betyr 1.

6.4 Inverteren: den første porten vi faktisk bygger

6.4.1 To brytere i serie

Nå har vi det vi trenger, og vi kan bygge den enkleste porten som finnes: NOT-porten, som snur signalet.

Oppskriften er kort: sett de to transistortypene over hverandre mellom VDD og GND, og ta utgangen ut midt imellom. Begge styres av det samme inngangssignalet.



Figur 6.3 Inverteren tegnet på to måter. Til venstre som to brytere, slik vi tegner den i resten av kapitlet. Til høyre med symbolene du møter i faglitteraturen. Det er samme krets.

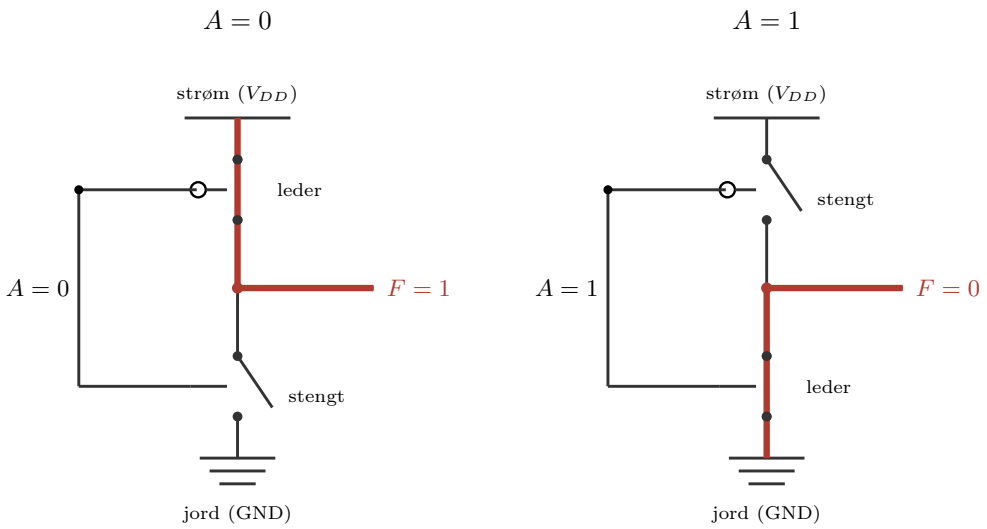
Den øverste er pMOS-en, den med boblen, som leder når inngangen er 0. Den nederste er nMOS-en, som leder når inngangen er 1.

6.4.2 Begge tilfellene, i ro og mak

Det finnes bare to muligheter, så vi går gjennom begge.

Inngangen er 0. Da leder pMOS-en øverst, fordi den leder ved 0, mens nMOS-en nederst er stengt, fordi den trenger 1 for å lede. Utgangen har fri vei opp til VDD og ingen vei ned til GND. Den blir høy, altså 1.

Inngangen er 1. Nå er det motsatt. pMOS-en er stengt, nMOS-en leder, og utgangen har fri vei ned til GND og ingen vei opp. Den blir 0.



Ettallet kommer fra forsyningen, ikke fra inngangen.
Inngangen velger bare hvilken av de to veiene som åpnes.

Figur 6.4 Inverteren i begge tilstandene. Den røde streken viser hvor verdien på utgangen kommer fra. Inngangen rører aldri utgangen, den åpner bare én av de to veiene.

Inn med 0, ut med 1. Inn med 1, ut med 0. Det er en NOT-port bygget av to brytere.

6.4.3 Det som er lett å gå glipp av

Les de to tilfellene en gang til, og se etter hvor ettallet kom fra.

Det kom ikke fra inngangen. Inngangen rørte aldri utgangen, og på tegningen ser du at det ikke går noen ledning mellom dem. Inngangen gikk bare til de to styreledningene. Det den gjorde, var å velge *hvilken av de to veiene som skulle åpnes*, og verdien på utgangen kom fra VDD eller fra GND, altså fra forsyningen.

Hvor nullene og ettallene kommer fra

Hver eneste ener i maskinen din er hentet rett fra strømforsyningen, og hver eneste null er hentet fra jord. Signalet som kommer inn i en port, bærer ikke verdien videre. Det åpner bare en kran. Verdien som renner ut er ny, og like sterk uansett hvor slapp inngangen var.
Det er derfor du kan koble tusen porter etter hverandre uten at signalet blir utvannet. Vi kommer tilbake til det i seksjon 6.6, og det er viktigere enn det ser ut nå.

6.4.4 Aldri begge samtidig

Én ting til i figuren: så lenge inngangen ligger i ro, leder de to transistorene aldri samtidig. Inngangen er enten 0 eller 1, de to typene reagerer motsatt på den, og leder den ene, er den andre stengt. Det er bra. Hvis begge ledet samtidig, ville det gått en rak strøm fra VDD til GND uten å gjøre noe nyttig, og det kalles en kortslutning.

Det gir en følge som er lett å overse. Når kretsen står i ro, går det nesten ingen strøm gjennom den. Strømmen går bare i det korte øyeblikket utgangen skifter.

Med ett lite unntak

I selve omslaget er begge transistorene så vidt ledende samtidig, i de få pikosekundene inngangen er på vei fra 0 til 1 eller omvendt. Da renner det en liten, rak strøm fra forsyningen til jord. Den varer kort, men den kommer hver eneste gang, og er en av grunnene til at det koster å skifte. Appendiks D går nærmere inn på det.

Fun Fact

Det er denne egenskapen som gjør at en telefon kan ligge på bordet i en uke og fortsatt ha batteri. Milliarder av transistorer holder på verdiene sine, og nesten ingen av dem trekker strøm mens de gjør det.
Egenskapen har også gitt teknologien navnet sitt. De to transistortypene utfyller hverandre, og *complementary* betyr nettopp det. Derav **CMOS**, complementary MOS, teknologien praktisk talt all digital elektronikk er bygget i.

Husk ordet «nesten». Vi kommer tilbake til det i seksjon 6.10, og da skjønner du hvor mye det betyr.

6.5 NAND, NOR, og hvorfor AND er dyrere

6.5.1 To i serie, eller to i parallell

Inverteren hadde én inngang og trengte derfor én transistor oppe og én nede. Med to innganger trenger vi to oppe og to nede, til sammen fire. Spørsmålet er hvordan de skal kobles, og det finnes bare to muligheter: etter hverandre eller ved siden av hverandre. Du kjenner dem fra kapittel 5, der det var lysbrytere i en stue.

To brytere *i serie*: strømmen må gjennom begge, så begge må lede. To brytere *i parallell*: strømmen har to veier å velge mellom, så det holder at én leder.

Se på NAND-porten til venstre. Nede står de to nMOS-ene i serie, så veien ned til jord åpner seg bare når begge inngangene er 1. Oppe står de to pMOS-ene i parallell, og siden de leder ved 0, holder det at *én* av inngangene er 0 for at utgangen kobles til VDD. Utgangen er altså 0 bare når begge inngangene er 1, og det er sannhetsverditabellen til NAND.

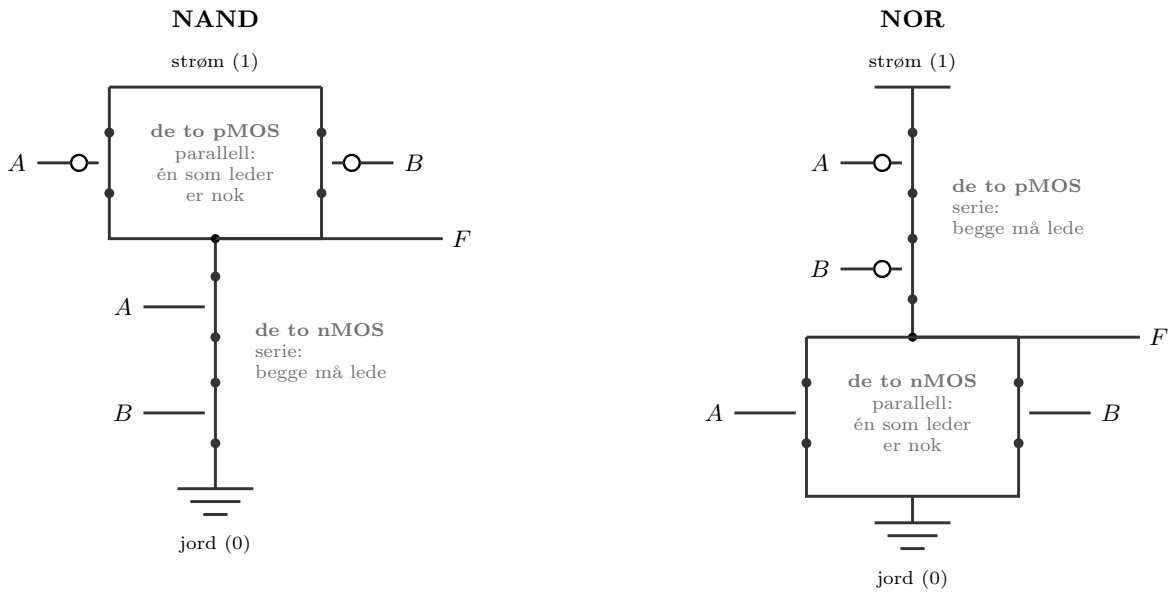
La oss sjekke ett tilfelle, slik vi gjorde med inverteren. Sett $A = 1$ og $B = 0$. Nede leder den ene nMOS-en (A er 1) og den andre er stengt (B er 0). De står i serie, så veien til jord er brutt. Oppe leder pMOS-en som styres av B, siden B er 0, mens den andre er stengt. De står i parallell, så den ene som leder er nok, og utgangen kobles til VDD.

Utgangen blir 1, og det stemmer: NAND skal gi 1 i alt annet enn tilfellet der begge er 1. Prøv de tre andre radene selv. Det tar to minutter, og det er den beste måten å se at det henger sammen.

NOR-porten til høyre er samme idé speilvendt: serie oppe, parallell nede.

Styreledningene er ikke tegnet

Figuren viser hvilken bryter som styres av A og hvilken som styres av B, men den tegner ikke ledningene som går dit. De ville krysset hverandre tre ganger og gjort figuren uleselig.
Det er samme slags forenkling som returledningen vi utelot i kapittel 5. Ledningene finnes, de er bare ikke det figuren handler om.



Figur 6.5 NAND og NOR, fire transistorer hver. Forskjellen ligger i hvem som står i serie og hvem som står i parallell. Styreledningene er utelatt for å holde figuren lesbar, slik returledningen var i forrige kapittel.

6.5.2 Hvert trinn snur signalet

Da kan du spørre: hvor er AND-porten?

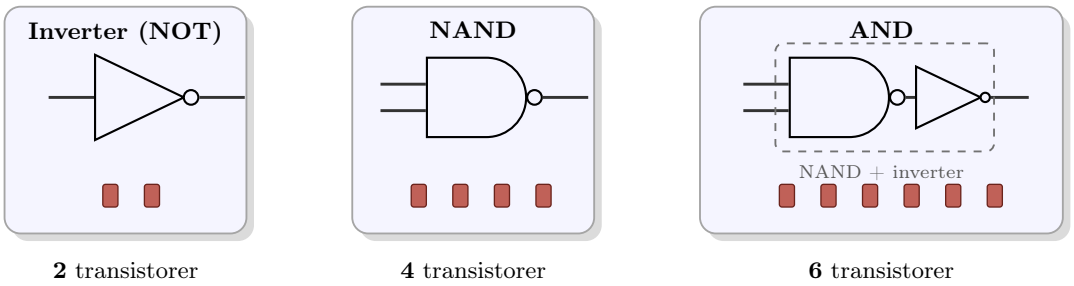
Prøv å bygge den. Setter du de to nMOS-ene i serie nede, trekkes utgangen til *jord* når begge er 1, og du får 0 ut når begge er 1. Det er NAND, ikke AND. Hvordan du enn kobler dem, blir det slik. Transistorene oppe kobler utgangen til VDD, transistorene nede kobler den til GND, og de to gruppene gjør alltid det motsatte av hverandre. Et slikt trinn snur derfor alltid signalet.

Dette betyr ikke at AND-porten ikke finnes

AND-porten finnes. Den ligger i alle kretsbiblioteker, og du tegner den som ett symbol.
Det som er sant, er at hvert enkelt *trinn* av denne typen snur signalet. En AND-port er derfor bygget som en NAND-port med en inverter etter seg, to trinn der det andre snur tilbake. Porten er like virkelig som før. Den er bare satt sammen av mer enn du trodde.

6.5.3 Regnskapet

Nå kan vi telle, og da blir tallene interessante.



Figur 6.6 Hva portene koster. AND er en NAND med en inverter etter seg, og koster derfor halvannen gang så mye som NAND.

En inverter koster to transistorer, en NAND fire og en AND seks, fordi den er en NAND pluss en inverter.

Det er motsatt av det de fleste gjetter. AND føles som den enkle porten og NAND som den rare med boble på. I silisium er det omvendt: NAND er den enkle, og AND er en NAND med et ekstra trinn.

Port	Transistorer	Hvordan
NOT	2	étt trinn
NAND	4	étt trinn
NOR	4	étt trinn
AND	6	NAND + inverter
OR	6	NOR + inverter
XOR	8–12	flere trinn, avhengig av oppskrift

Tabell 6.1 Hva portene koster i transistorer. Tallene gjelder porter med to innganger.

Kapittel 5 sa at det lønner seg å spare porter, og brukte mye plass på å forenkle uttrykk. Nå ser du hva det betales i. Hver port du sparer, er et håndfast antall transistorer som slipper å ta plass, trekke strøm og bli varme. Og nå forstår du hvorfor en kretsdesigner heller skriver om et uttrykk til NAND-er enn til AND-er. Det er ingen vane. Det er to transistorer per port.

Regn på det selv

En krets bruker tolv AND-porter med to innganger. En designer finner en måte å få samme resultat med tolv NAND-porter i stedet.
Tolv ganger seks mot tolv ganger fire: du sparte 24 transistorer av 72, altså en tredel. Gjør det samme i en krets med noen hundre millioner porter, og du har spart noe som merkes på strømregningen til et datasenter.

6.6 Null og ett er områder, ikke tall

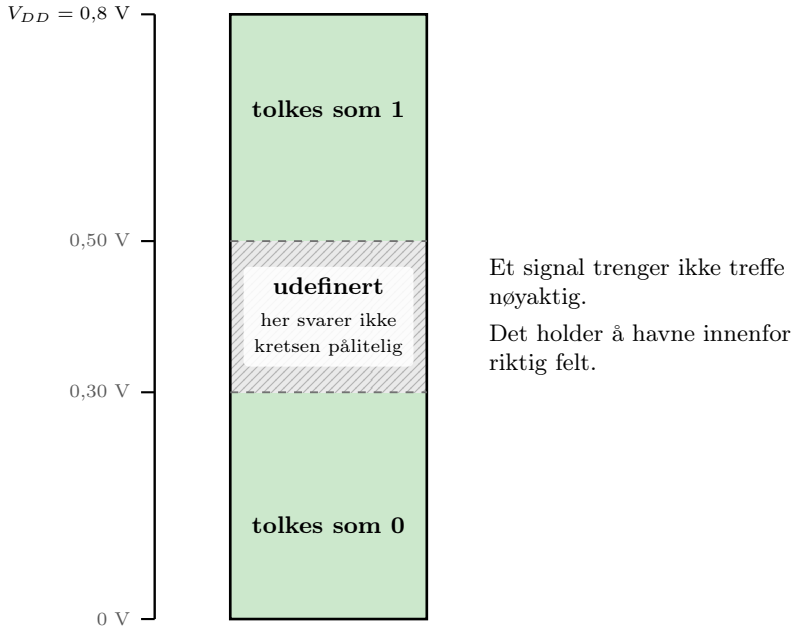
6.6.1 Verden er analog, og det er et problem

Vi lovet å komme tilbake til dette, og nå har vi noe konkret å henge det på.

En ledning fører en spenning som kan være hva som helst mellom 0 og 0,8 volt. Den kan være 0,37 volt, den kan dirre litt opp og ned, og den blir dratt på av naboledninger, av varme og av at noe annet på brikken skifter i samme øyeblikk. Logikken kjenner derimot bare 0 og 1. Hvordan kan et system med to verdier bygges av noe som kan ha uendelig mange?

6.6.2 Svaret: gi hvert tall et rom å bo i

Løsningen er enkel, og hele den digitale teknologien hviler på den. I stedet for å kreve at 0 betyr nøyaktig 0,000 volt, sier vi at alt under en viss grense *teller som* 0, og at alt over en annen grense teller som 1. Mellom de to grensene ligger et belte der kretsen ikke lover noe.



Figur 6.7 0 og 1 er områder, ikke tall. Havner et signal i det skraverte beltet, har kretsen ikke lovet noe om hva den gjør med det.

Signalet trenger altså ikke treffe presist. Det holder at det havner i riktig felt. En ener på 0,74 volt og en ener på 0,79 volt er den samme eneren.

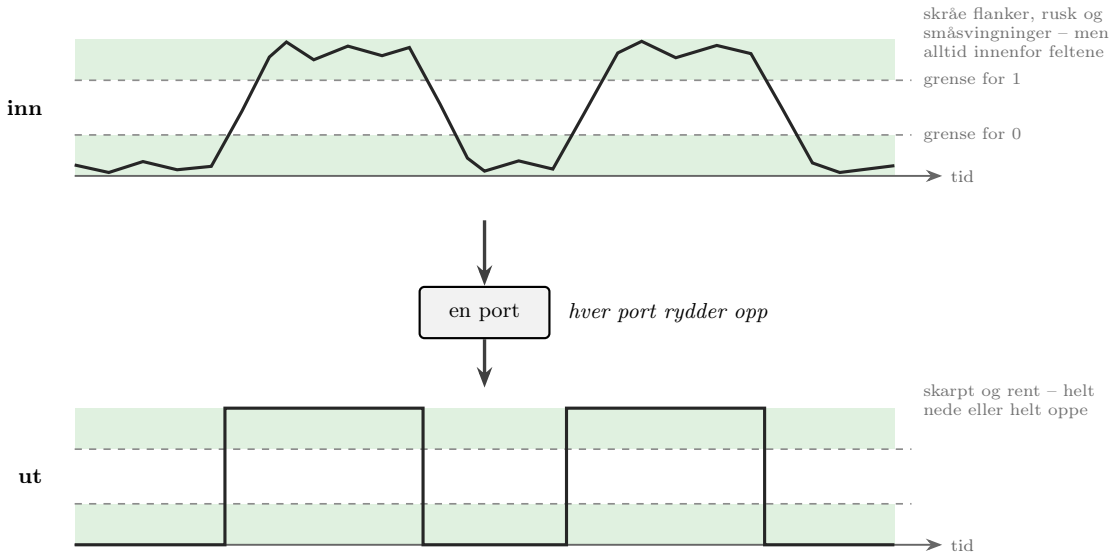
Hvorfor det er et belte i midten

Du kan spørre hvorfor vi ikke bare setter én grense midt på, slik at alt under er 0 og alt over er 1. Problemet er at en krets som får et signal helt på grensa, må velge, og den kan velge ulikt fra gang til gang. Med et bredt ingenmannsland i midten havner ingen der ved et uhell.
Beltet er altså med vilje, og bredden på det er en designbeslutning. Det tekniske navnet er **støymargin**: hvor mye juks signalet tåler før det blir misforstått. Tallene bak står i appendiks D.

6.6.3 Hver port vasker signalet rent

Nå følger noe som gjør digital teknologi mulig i praksis, og det henger sammen med det vi så på inverteren.

Når et slapt, forvrengt signal kommer inn i en port, bærer ikke porten det videre. Den *leser* det, avgjør om det er over eller under grensa, og lager et helt nytt signal fra VDD eller GND. Utgangen er skarp, selv om inngangen var slurvete.



Figur 6.8 Et forvrengt signal inn, et skarpt signal ut. Porten bærer ikke signalet videre, men leser det og lager et nytt fra forsyningen.

Signalet blir vasket rent i hvert eneste ledd. Rusk som samler seg underveis, forsvinner med en gang, og regnskapet starter på nytt etter hver port. Fagordet for dette er **signalgjenoppretting**.

6.6.4 Derfor overlever en kopi av en kopi

Du har kanskje lagt merke til følgen av dette uten å tenke over det. Kopierer du en lydfil og kopierer kopien, og gjør det tusen ganger, er den siste fila bit for bit lik den første. Hvert eneste bit ble lest som enten over eller under grensa, og skrevet på nytt like skarpt. En kassett kopiert tusen ganger blir bare susing, fordi hver liten unøyaktighet fulgte med videre og la seg oppå den forrige.

Forskjellen er ikke at digitalt er mer nøyaktig. Digitalt *retter seg selv* hele tiden, fordi hver port kaster alt som ikke er en ren 0 eller 1.

Tankevekker

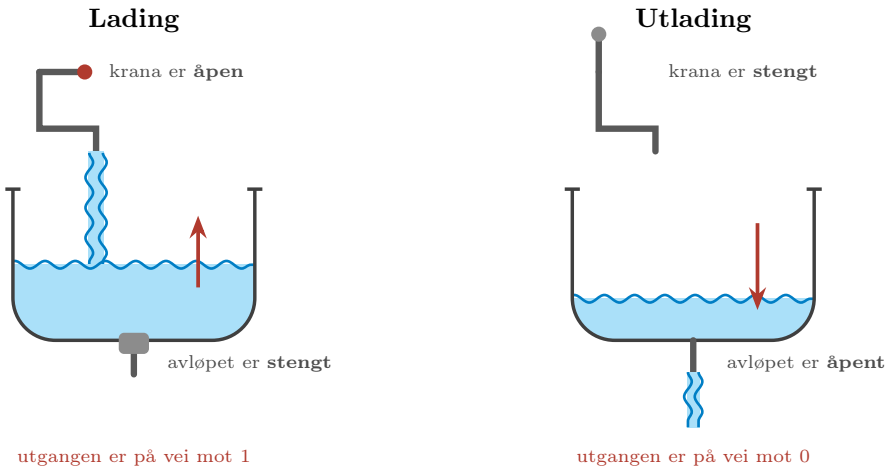
Kapittel 1 sa at en datamaskin er lag på lag med abstraksjoner, og at abstraksjoner lekker. Her står du ved den nederste, og den lekker ikke. «Et bit er en 0 eller en 1» er en påstand om noe fysisk, og den holder fordi hver port tvinger signalet tilbake i form, noen milliarder ganger i sekundet. Resten av boka hviler på dette. Kunne en ener bli litt slapp underveis, ville ingenting høyere opp fungert.

6.7 Det tar tid

6.7.1 Badekaret

Til nå har vi beskrevet alt som om det skjer momentant: bryteren begynner å lede, og utgangen er 1. Så enkelt er det ikke, og forskjellen er det resten av kapitlet handler om.

Tenk deg utgangen på en port som et badekar. Skal porten sette utgangen til 1, åpner den krana øverst og fyller karet. Skal den sette utgangen til 0, stenger den krana og åpner avløpet.



Jo større kran, jo raskere fylles karet. Jo større kar, jo lengre tid tar det.

Figur 6.9 Utgangen på en port, tegnet som et badekar. Å sette utgangen til 1 er å fylle karet, å sette den til 0 er å tømme det. Begge deler tar tid.

Det tar tid å fylle et badekar, og det tar tid å tømme det. Hvor lang tid, avhenger av to ting: hvor stort karet er, og hvor stor krana er.

6.7.2 Hvorfor er det et badekar der?

Her må vi være presise, for det er lett å misforstå. Ingen har *satt inn* et badekar, det oppstår av seg selv.

En ledning som skal føre en spenning, må lades opp til den spenningen, på samme måte som et kar må fylles før vannstanden er høy. Det samme gjelder styreledningen inn til neste port, siden det er spenningen på den som avgjør om transistoren leder. Summen av alt dette, ledningen og styreledningene til portene den går til, er badekaret. Fagordet er **kapasitans**, og det betyr omtrent «hvor mye som må fylles på».

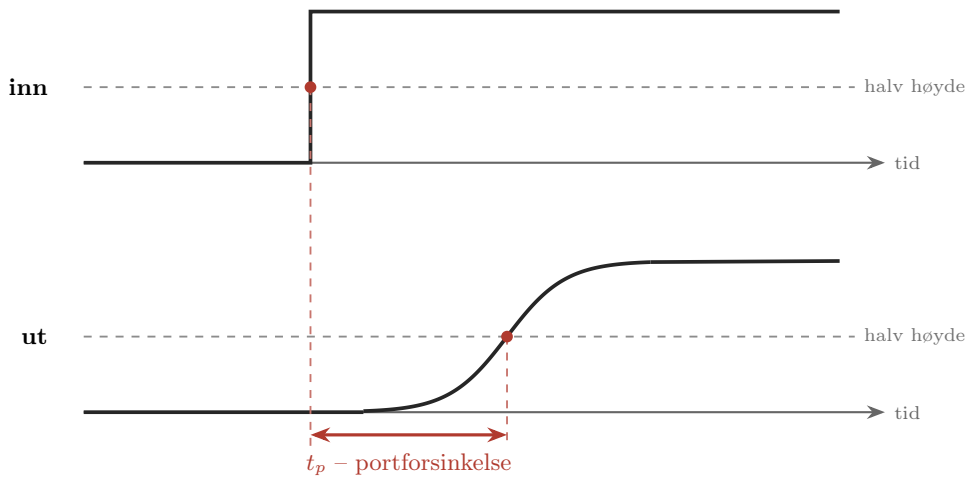
En vanlig misforståelse

Kapasitansen er ikke en komponent noen har plassert der fordi den trengs. Den er en uunngåelig følge av at ledninger og transistorer er fysiske ting som tar plass. Ingen designer er glad i den, og alle jobber for å få den ned, fordi mindre kar gir raskere kretser.
Merk også at denne lille mengden lagret ladning *ikke* er minne. Den hører ikke til minnehierarkiet

du møter i kapittel 7. Den er bare den uunngåelige tregheten i at et signal må bygge seg opp.

6.7.3 Portforsinkelse

Nå kan vi gi tiden et navn. **Portforsinkelse** er tiden fra inngangen til en port endrer seg til utgangen har kommet over på den nye verdien.



Utgangen begynner først å røre seg litt etter inngangen, og bruker så tid på å komme helt opp.

Figur 6.10 Portforsinkelse. Utgangen begynner først å røre seg litt etter inngangen, og bruker så tid på å komme helt opp.

Portforsinkelsen er ikke lik for alle porter. En XOR har flere trinn inni seg enn en NAND, og bruker derfor lengre tid. Vi regner likevel med *samme* tall for alle porter i dette kapitlet, siden poenget er hvordan man teller og ikke hvor presis summen blir.

I en moderne prosess ligger portforsinkelsen på noen få *pikosekunder*, og et pikosekund er en billiondel av et sekund. Vi regner med 5 pikosekunder per port i resten av kapitlet. Tallet er rundt for at regningen skal bli grei, og det ligger i riktig område.

Størrelsesordenene, så du har dem

Enhet	Hvor lenge	Hva som skjer der
pikosekund (ps)	10^{-12} s	én port skifter
nanosekund (ns)	10^{-9} s	en instruksjon kjøres, eller et tall hentes fra arbeidsminnet
mikrosekund (μs)	10^{-6} s	en lesing fra en SSD
millisekund (ms)	10^{-3} s	en lesing fra en treg disk

Tabell 6.2 Hvert trinn er tusen ganger lengre enn det forrige.

Avstanden fra et pikosekund til et sekund er omtrent like stor som fra ett sekund til tretti tusen år.

6.7.4 Større kran, raskere port, og en regning

Hvorfor gjør man ikke bare alle transistorene større, slik at de fyller karet fortere? Det gjør man faktisk, der det haster. En bredere transistor leverer mer strøm og skifter raskere, på samme måte som en bredere kran fyller badekaret fortere.

Men den koster. Den tar mer plass, den trekker mer strøm, og, her kommer den kjipe delen, den er selv et større kar for porten som står foran. Gjør du én port raskere, gjør du porten før den tregere.

Dette er den første av flere avveininger i kapitlet der du ikke kan få alt. Fart, plass og strøm drar hver sin vei, og en kretsdesigner velger hele dagen.

6.8 Den lengste veien

6.8.1 Portene står ikke i kø

Kapittel 5 bygde en fulladder av fem porter. Hvis hver port tar 5 pikosekunder, hvor lang tid bruker fulladderen?

Det fristende svaret er fem ganger fem, altså 25. Det er feil, og grunnen er viktig. Portene jobber nemlig ikke etter tur. De jobber alle sammen hele tiden, og en port begynner i samme øyeblikk inngangene er klare, uten å bry seg om hva de andre holder på med. Tiden gjennom kretsen er derfor ikke summen av alle portene. Den er lengden på den *lengste veien* fra en inngang til en utgang.

6.8.2 Fulladderen, målt

La oss se på det i kretsen du allerede kjenner.

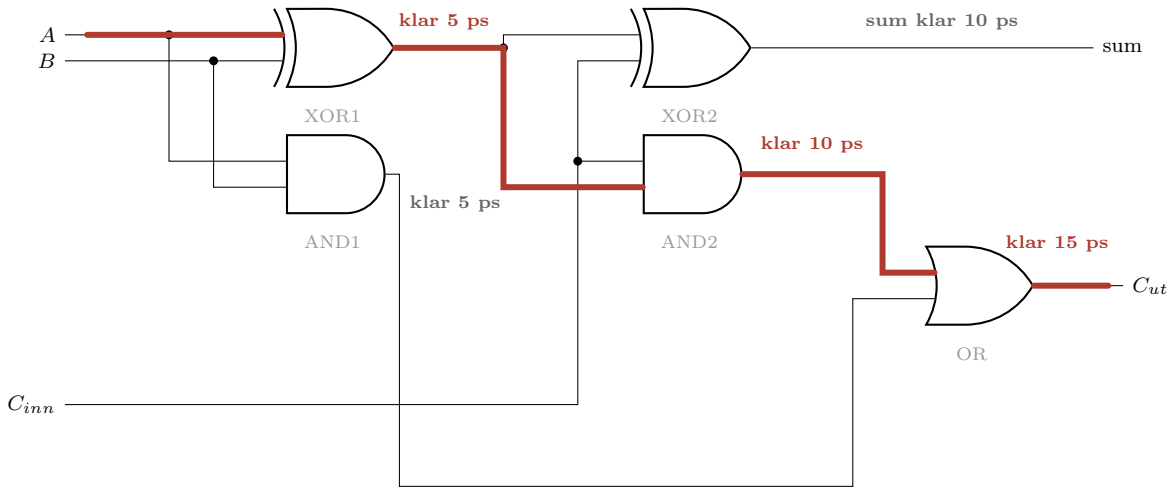
Følg den røde veien. Signalet går først gjennom XOR1, som er ferdig etter 5 pikosekunder, så gjennom AND2, ferdig etter 10, og så gjennom OR-porten, ferdig etter 15. Tre porter på rad gir 15 pikosekunder.

AND1 var ferdig allerede etter 5, men det hjalp ikke. OR-porten har to innganger og kan ikke svare før begge er på plass, så den raske inngangen må stå og vente på den trege.

Denne lengste veien kalles den **kritiske veien**. Det er den som bestemmer hvor rask kretsen er, og den eneste en designer bryr seg om å gjøre kortere. Å gjøre noe på en av de andre veiene raskere, endrer ingenting. Den samme tankegangen kommer du til å bruke når du senere skal få et program til å gå raskere: finn det som faktisk bestemmer tiden, og la resten ligge.

Et bilde fra en helt annen verden

Fire personer lager middag sammen. Potetene tar 20 minutter, kjøttet 15, sausen 10 og salaten 4, og alle jobber samtidig, hver med sitt.
Middagen er klar etter 20 minutter, ikke etter 49. Og skal dere spise tidligere, hjelper det ikke å hakke salaten raskere. Potetene er den kritiske veien.



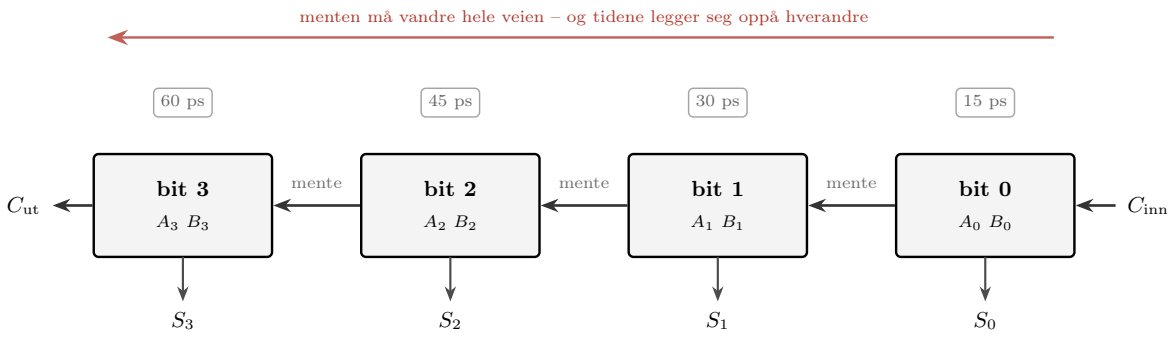
Den lengste veien: XOR1 → AND2 → OR. Tre porter, 15 ps.

OR-porten har to innganger. Den ene er klar etter 5 ps, den andre først etter 10. Porten kan ikke svare før begge er på plass, så den starter etter 10 og er ferdig etter 15. Veiene går samtidig — tidene legges ikke sammen, det er den lengste som bestemmer.

Figur 6.11 Den kritiske veien gjennom fulladderen fra kapittel 5. Kretsen har fem porter, men bare tre av dem ligger etter hverandre.

6.8.3 Og så legger vi dem i rekke

Kapittel 5 sa at mentekjeden koster tid, uten å si hvor mye. Nå kan vi regne.



64 bit: 64×15 ps $= 960$ ps, altså omtrent 1 GHz.

Figur 6.12 Mentekjeden. Her hjelper det ikke at portene jobber samtidig: hvert ledd må vente på menten fra det forrige, så tidene legger seg oppå hverandre.

I en mentekjede må menten ut av den første fulladderen før den andre kan bli ferdig, og den andre må bli ferdig før den tredje. Ingenting går i parallell. Det er en kø, der hvert ledd venter på det forrige. Fire bit koster dermed fire ganger 15, altså 60 pikosekunder.

Hva med en kjede på 64 bit, som er det en vanlig prosessor regner med?

$$64 \times 15 \text{ ps} = 960 \text{ ps}$$

Det er nesten et helt nanosekund for én addisjon. Deler vi ett sekund på 960 pikosekunder, får vi plass til omtrent **en milliard** slike addisjoner i sekundet.

Nå ser du hvorfor kapittel 5 brukte plass på å klage over mentekjeden. En krets som gjør alt annet på 15 pikosekunder, bruker nesten tusen på å legge sammen to tall.

Overslaget er for pessimistisk. Finner du ut hvorfor?

Vi ganget 15 pikosekunder med 64. Men prøv å bruke det du lærte i forrige seksjon på ledd nummer to i kjeden.
Der er A og B klare fra første stund, og det eneste som kommer sent, er menten fra naboen. Menten trenger ikke gjennom hele fulladderen, bare gjennom AND2 og OR, altså to porter og 10 pikosekunder. Det er bare det *første* leddet som koster 15. De 63 andre koster 10 hver:

$$15 + 63 \times 10 = 645 \text{ ps}$$

Det er fortsatt tregt, fortsatt den trege delen av maskinen, og tiden vokser fortsatt rett opp med antall bit. Men eksempelet viser at den kritiske veien ikke alltid er den du ser først, og at det lønner seg å spørre hvilket signal som faktisk kommer sist.

Hvorfor maskiner likevel er raskere enn 1 GHz

Ekte prosessorer bruker ikke en rett mentekjede på 64 bit. De bruker kretser som gjetter menten på forhånd i stedet for å vente på den, og slipper dermed køen.
Den vanligste heter *carry look-ahead*. Den er mye raskere og bruker betydelig flere transistorer, så den er enda et eksempel på samme avveining: fart mot plass og strøm. Du trenger ikke vite hvordan den virker, men du bør vite at den finnes, og hvorfor noen gadd å finne den opp.

6.9 Klokka

6.9.1 Problemet som må løses

Ulike veier gjennom en krets blir ferdige til ulik tid. Noen svar er klare etter 5 pikosekunder, andre først etter 960. I mellomtiden står det tall på ledningene som ikke er ferdig regnet ut, halvferdige verdier som skifter mens signalene arbeider seg gjennom. Hvis noe leser av en ledning i feil øyeblikk, leser det søppel.

Maskinen trenger derfor en avtale om *når* det er trygt å se. Den avtalen er klokka.

En forklaring du kan møte, som ikke stemmer

Noen ganger forklares klokka med at verdiene lekker ut hvis man venter for lenge, så man må rekke å lagre dem før de forsvinner.
Det stemmer ikke for vanlige logikkretser. Så lenge inngangen ligger fast, holder porten utgangen aktivt oppe, fordi én av transistorene leder og kobler den til VDD eller GND. Verdien renner ikke ut.
Grunnen til at vi trenger klokka er en annen: ulike veier blir ferdige til ulik tid, og noen må si fra når alle er det.

6.9.2 Alle leverer samtidig

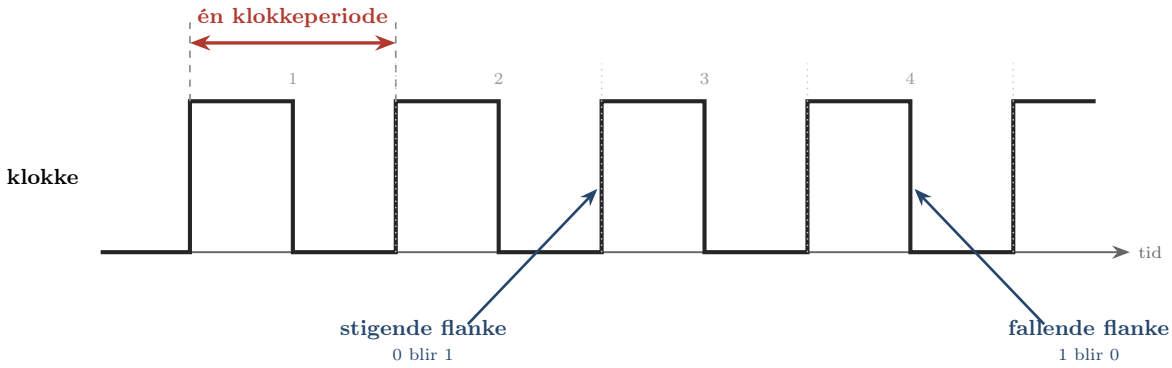
Tenk på en skriftlig prøve. Alle begynner når det ringer inn, alle jobber i sitt eget tempo, og alle leverer når det ringer ut. Den som ble ferdig tidlig, sitter og venter, for tiden settes etter den som trenger lengst.

Klokka i en datamaskin gjør det samme: den deler tiden i like lange biter, innenfor hver bit får alle kretsene jobbe fritt, og på slutten av biten lagres alle svarene samtidig.

Strengt tatt er det sløsing, for de raske kretsene står og venter mesteparten av tiden. Til gjengjeld slipper man å holde rede på hvem som blir ferdig når, og det er en enorm forenkling.

6.9.3 Hvordan et klokkesignal ser ut

Klokka er bare en ledning til, og den går til nesten alt på brikken. På den ledningen veksler spenningen mellom 0 og 1 i et helt jevnt mønster.



frekvens = 1 delt på klokkeperioden

Figur 6.13 Klokkesignalet. Dette er figuren det lønner seg å kunne utenat.

Tre ord hører til figuren, og de bør du kunne.

Klokkeperioden er tiden fra ett sted i mønsteret til samme sted neste gang. Du kan måle fra en stigende flanke til neste stigende flanke, eller fra en fallende til neste fallende. Det gir samme svar.

Stigende flanke er øyeblikket signalet går fra 0 til 1. **Fallende flanke** er det motsatte. Det er på flankene ting skjer: det er da resultatene lagres.

6.9.4 Fra periode til frekvens

Frekvensen er hvor mange perioder som går per sekund. Den måles i *hertz*, forkortet Hz, etter fysikeren Heinrich Hertz. En hertz er én periode i sekundet, og en *gigahertz* er en milliard av dem.

Sammenhengen mellom periode og frekvens får du mest bruk for i hele kapitlet:

$$\text{frekvens} = \frac{1}{\text{klokkeperioden}}$$

Det er samme slags sammenheng som mellom fart og tid. Kjører du strekningen på halvparten av tiden, kjørte du dobbelt så fort.

La oss sette inn et tall. En prosessor på 3 GHz gjør tre milliarder perioder i sekundet, og hver periode varer da:

$$\frac{1}{3 \cdot 10^9}\ \text{s} = 333\ \text{ps}$$

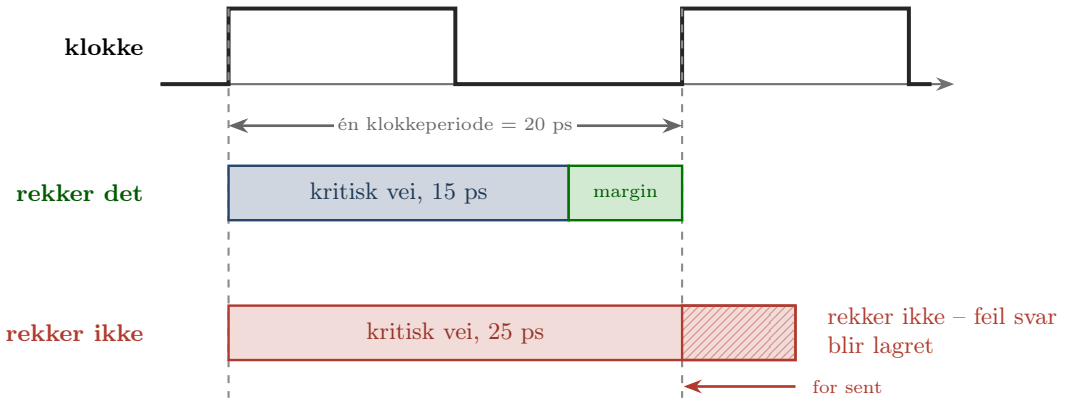
Det er en tredjedels nanosekund. I løpet av den tiden skal alt som skal bli ferdig, bli ferdig.

Regn den andre veien

En prosessor har en klokkeperiode på 500 pikosekunder. Hva er frekvensen?
$1/(500 \cdot 10^{-12}) = 2 \cdot 10^9$, altså 2 GHz.
Prøv også 250 ps, som gir 4 GHz, og 1 ns, som gir 1 GHz. Jo kortere periode, jo høyere frekvens. Det er samme tall snudd på hodet.

6.9.5 Hva som setter taket

Nå kan vi sette de to siste seksjonene sammen, og det gir en av de mest nyttige innsiktene i kapitlet. Alt som skal bli ferdig innen neste flanke, må rekke det. Klokkeperioden må derfor være minst like lang som den kritiske veien.



Figur 6.14 Den kritiske veien må få plass innenfor én klokkeperiode. Rekker den ikke fram, blir et halvferdig svar lagret, og maskinen regner feil.

Er perioden for kort, rekker ikke den trege veien fram. Da lagres et halvferdig svar, og maskinen regner feil. Ikke litt feil, men helt feil, og på en måte som er svært vanskelig å finne i etterkant.

Det er derfor man ikke bare kan skru opp klokkefrekvensen og få en raskere maskin. Over et visst punkt begynner den å regne galt.

Fun Fact

Dette er nettopp det folk gjør når de *overklokker* en maskin. De setter klokkeperioden kortere enn produsenten anbefaler og håper at marginen var romsligere enn oppgitt.
Ofte går det bra, fordi produsenten legger inn margin for varme dager og dårlige eksemplarer. Går det ikke bra, får man tilfeldige krasj. Det er ikke programvarefeil, men kretser som ikke rakk fram.

6.9.6 Hvorfor ikke 200 GHz?

Her er en regning som er fristende å gjøre, og som fører feil. En port tar 5 pikosekunder. Setter vi klokkeperioden til 5 pikosekunder, får vi en frekvens på $1/(5 \cdot 10^{-12})$, altså 200 GHz. Hvorfor går ikke prosessorer i 200 GHz?

Det er tre grunner, og alle tre er greie å ha med seg.

For det første er den kritiske veien aldri bare én port. Den er mange porter på rad, og vi har sett en på 64 ledd. Perioden må romme hele veien.

For det andre trenger en klokkeperiode mer enn selve regnetiden. Den trenger margin for at brikken er varm, for at akkurat dette eksemplaret er litt tregere enn gjennomsnittet, og for at klokkesignalet ikke kommer helt samtidig overalt.

Den tredje grunnen er den avgjørende: brikken ville brent opp. Det er temaet i neste seksjon.

Global og lokal klokke

En moderne brikke har ikke én klokke, men flere. Prosessorkjernene kan gå i én takt, minnegrensesnittet i en annen og grafikken i en tredje.
Det gir to fordeler. Deler som ikke trenger å være raske, slipper å bruke strøm på å være det, og deler som ikke er i bruk, kan få klokka slått helt av. Da står de stille og slutter å fylle badekar. De lekker fortsatt, som vi kommer tilbake til, men den største utgiften er borte.
Når noen oppgir at en maskin går i 3,5 GHz, er det den globale klokka til prosessorkjernene de snakker om.
Hvordan klokkesignalet fordeles til milliarder av transistorer, er et kapittel for seg. Det står i appendiks E og er ikke pensum.

6.10 Hvorfor klokka stoppet på rundt 3 GHz

6.10.1 Femti år med krymping

I 1965 skrev Gordon Moore, som senere var med på å grunnlegge Intel, en kort artikkel i bladet *Electronics*. Han hadde lagt merke til at antallet komponenter man fikk plass til på en brikke, hadde doblet seg hvert år siden teknologien kom, og han gjettet at det ville fortsette en stund.

Det fortsatte i over femti år. I 1975 justerte han anslaget til en dobling hvert annet år, og med den justeringen har observasjonen truffet forbløffende godt helt fram til vår tid. Den kalles **Moores lov**, og navnet er misvisende.

Det er ingen naturlov

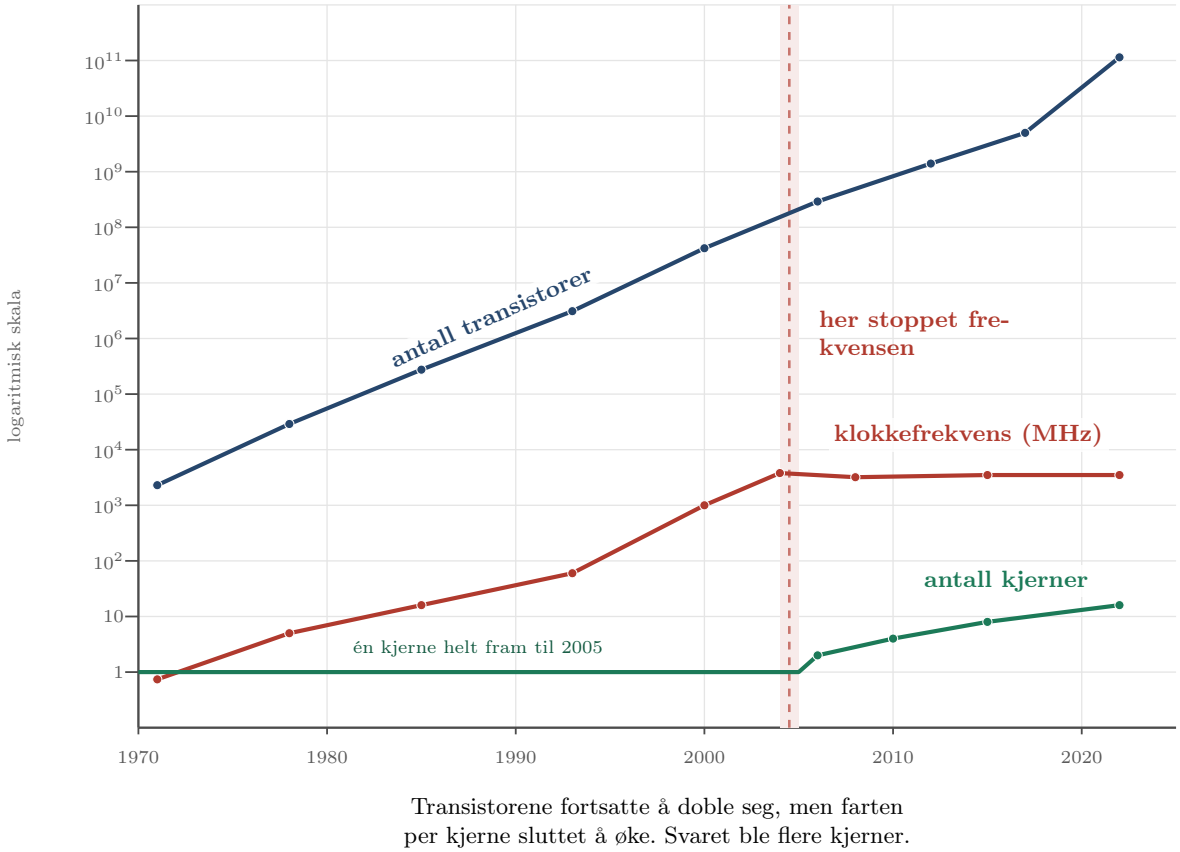
Ingenting i fysikken sier at man skal få plass til dobbelt så mange transistorer annethvert år. Moore observerte en trend og spådde at den ville holde.
Så skjedde noe interessant: hele bransjen begynte å planlegge etter spådommen. Produsenter la toårsplaner, utstyrsleverandører siktet mot de samme målene, og kundene ventet det.
Moores lov ble dermed i stor grad en selvoppfyllende profeti, en felles tidsplan som alle holdt seg til fordi alle andre gjorde det. Det gjør den ikke mindre imponerende, men det forklarer hvordan den kunne stoppe uten at noen naturlov ble brutt.

Hva fikk man ut av krympingen? To ting, og lenge fikk man begge gratis. Man fikk *flere* transistorer på samme flate, altså mer maskineri. Og man fikk *raskere* transistorer, fordi en mindre transistor har

et mindre badekar å fylle.

I tretti år gikk det slik at nye brikker hadde både flere transistorer og høyere klokkefrekvens. Maskinen du kjøpte i år var målbart raskere enn den fra i fjor, og du måtte ikke gjøre noe for å nyte godt av det. Programmene ble raskere av seg selv.

Så sluttet det.



Figur 6.15 Tre kurver fra 1971 til i dag. Antall transistorer fortsetter oppover, men klokkefrekvensen flater ut rundt 2004 — og i samme øyeblikk begynner antall kjerner å stige. Merk at y-aksen er logaritmisk, og at de tre kurvene måler hver sin ting. Det er formen på dem som er poenget, ikke at de kan sammenlignes tall mot tall.

Først en bruksanvisning, for aksen er uvanlig. Hvert hakk oppover er *ti ganger* så mye som hakket under. Uten det grepet ville ikke 2 300 transistorer og 114 milliarder fått plass på samme ark. Prisen er at et lite hopp på papiret er et stort hopp i virkeligheten, og at en kurve som ser ut til å stige jevnt, i virkeligheten dobler seg om og om igjen.

Se så på de tre kurvene. Antall transistorer fortsetter oppover gjennom hele perioden. Klokkefrekvensen følger tett etter fram til rundt 2004 og flater så nesten helt ut. I samme øyeblikk begynner antall kjerner å stige, fra én til to til fire.

Noe skjedde rundt 2004. Resten av seksjonen handler om hva.

6.10.2 Hver eneste vending koster

Gå tilbake til badekaret. Hver gang en utgang skal gå fra 0 til 1, må karet fylles. Vannet kommer fra VDD, og det koster energi å løfte det dit. Når utgangen senere skal tilbake til 0, tømmes karet ut i jord, og energien som lå i det, er borte. Ingenting av det kommer tilbake. Alt ender som varme.

Dette er svaret på spørsmålet vi åpnet kapitlet med. Telefonen blir varm fordi den fyller og tømmer milliarder av små badekar, milliarder av ganger i sekundet, og hvert eneste kar koster en slurk som blir til varme.

Hvor mye av strømmen blir varme?

Alt sammen.
Ikke «det meste», men alt. En datamaskin tar inn elektrisk energi og leverer ikke noe tilbake. Den løfter ingenting, og den lager ingen bevegelse. Det bitte lille som slipper ut som lyd fra viften eller lys fra skjermen, blir også til varme i rommet like etterpå.
En laptop som trekker 45 watt, varmer rommet like mye som en varmeovn på 45 watt. Det er energibevaring, og et av de få stedene i faget der fysikken gir deg et helt presist svar gratis.

6.10.3 Hva som bestemmer hvor varmt det blir

Tre ting avgjør varmen, og alle tre følger av badekarbildet.

Hvor mange kar. Flere transistorer betyr flere kar å fylle, og det er derfor en stor brikke bruker mer strøm enn en liten.

Hvor høyt vannet må løftes. Det er forsyningsspenningen, og den teller dobbelt, fordi du både løfter mer vann og løfter det høyere.

Hvor ofte. Det er klokkefrekvensen. Dobler du frekvensen, fyller og tømmer du dobbelt så mange ganger i sekundet, og bruker dobbelt så mye strøm.

Formelen, hvis du vil se den

Du kan hoppe over denne boksen uten å miste noe. Sammenhengen over skrives gjerne slik:

$$\text{effekt} \;\approx\; \text{kapasitans} \times \text{spenning}^2 \times \text{frekvens}$$

Det eneste du bør merke deg, er at spenningen står i andre potens. Halverer du spenningen, får du en firedel av effekten, og det gjør spenningen til den desidert mest effektive knappen å skru på. Et eksempel, fra en litt eldre brikke der spenningene var høyere enn i dag: den går i 2 GHz på 1,0 volt, og du vil ha den til 4 GHz. Frekvensen dobles, så effekten dobles. Men en høyere frekvens krever som regel også litt høyere spenning for å rekke fram i tide, la oss si 1,2 volt, og da ganges effekten i tillegg med $1{,}2^2/1{,}0^2 = 1{,}44$.
Til sammen: $2 \times 1{,}44 \approx 2{,}9$. Du doblet farten og nesten tredoblet varmen.

6.10.4 Triksen som sluttet å virke

Nå er vi framme ved det som skjedde rundt 2004.

Så lenge transistorene krympet, kunne man senke forsyningsspenningen samtidig, for mindre transistorer trenger mindre spenning. Siden spenningen teller dobbelt, falt effekten per transistor raskt nok

til at man kunne pakke inn flere og kjøre dem fortere uten at brikken ble varmere. Den balansen ga tretti år med gratis fart.

Så traff spenningen gulvet. En transistor trenger en viss spenning for å skille klart mellom på og av. Går du under den, slutter den å være en god bryter: den leder litt når den skal være stengt, og den blir treg. Støymarginen fra seksjon 6.6 krymper også, slik at signalene lettere blir misforstått.

Rundt 2004 sluttet altså spenningen å følge krympingen nedover. Før den tid falt den i takt med transistorene. Etterpå falt den mye saktere, fra rundt 1,3 volt den gangen, via 1,0 volt rundt 2010, til de 0,7 til 0,8 voltene vi ligger på nå. Fra det øyeblikket ble hver ny gigahertz betalt nesten rett i varme.

Dette punktet har fått navnet **varmeveggen**. Det er ingen vegg man kan bryte gjennom med bedre ingeniørkunst, for den står der to ønsker kolliderer: lavere spenning for å spare strøm, og høy nok spenning til at bryteren fortsatt er en bryter.

6.10.5 Og krana som drypper

Det kommer en ting til, og den blir verre jo mindre transistorene er. En stengt transistor er ikke helt stengt.



Figur 6.16 Lekkasje. En stengt transistor slipper likevel gjennom litt, og den regningen løper hele tiden — også når brikken ikke gjør noe.

Det lekker alltid litt strøm gjennom, og jo tynnere og mindre transistoren er, jo mer lekker den. Dette kalles **lekkasje**, og det er en utgift som løper hele tiden, også når brikken ikke gjør noe som helst. Da transistorene var store, var lekkasjen forsvinnende liten. I dagens størrelser er den en betydelig del av regnskapet.

Hvor lite det har blitt

For noen tiår siden kunne du tenke på utgangen til en port som et svømmebasseng. Du hadde god tid, og en liten lekkasje betydde ingenting.
I dag er det samme karet på størrelse med en teskje. Det fylles raskt, og det er poenget, men nå er lekkasjen stor i forhold til karet. Det er vanskelig å holde noe i en teskje med hull i.

6.10.6 Svaret ble flere kjerner

Da spenningen ikke kunne senkes mer, hadde bransjen et valg. De kunne fortsette å skru opp frekvensen og la brikkene bli varmere og varmere. Det gikk en liten stund, og rundt 2004 lå noen prosessorer på 3,8 GHz og krevde kjølere som fylte halve kabinettet. Eller de kunne slutte å skru opp frekvensen og bruke de stadig flere transistorene til noe annet.

De valgte det siste. I stedet for én kjerne som går dobbelt så fort, fikk vi to kjerner som går like fort som før. På papiret gir det like mange beregninger i sekundet. To kjerner har riktignok dobbelt så mange badekar å fylle, så den delen av regningen er uendret, men man slipper spenningsøkningen den doble frekvensen ville krevd, og det er nettopp faktoren 1,44 fra boksen over.

Se på figuren en gang til. Kurven for antall kjerner begynner å stige i samme øyeblikk som frekvenskurven flater ut. De to henger sammen.

Regningen ble sendt videre

Løsningen har en hake, og det er grunnen til at du som skal programmere, bør vite om dette. En raskere kjerne gjør alle programmer raskere uten at noen trenger å gjøre noe. To kjerner gjør bare de programmene raskere som er skrevet for å bruke begge. Da frekvensen stoppet i 2004, ble regningen altså sendt fra maskinvaren til programvaren, og det er derfor parallellprogrammering gikk fra å være et spesialfelt til noe alle må forholde seg til.
Hvor langt man kommer med flere kjerner, og hvorfor det finnes en grense der også, er tema i kapittel 8.

6.10.7 Om tallene du leser i nyhetene

En siste ting, siden du kommer til å møte den.

«2 nanometer» er et navn, ikke et mål

Når en produsent sier at de lager brikker på en «2 nm-prosess», er det lett å tro at noe i transistoren er 2 nanometer langt. Det er det ikke. En gang viste tallet til en faktisk lengde i transistoren, men den koblingen ble brutt for lenge siden. I det som markedsføres som en 2 nm-prosess, er den tilsvarende lengden snarere 12 til 20 nanometer.
Nodenavnet er i dag et generasjonsnavn, omtrent som modellåret på en bil. Det forteller at denne generasjonen er tettere og bedre enn den forrige, og det stemmer, men det er ikke et mål på noe. Du trenger ikke tallene. Du trenger bare å vite at tallet ikke betyr det det ser ut som.

Og hva kommer etter? Når transistorene ikke kan bli stort mindre i flaten, begynner man å stable dem i høyden. Man bygger spesialiserte kretser som gjør én oppgave svært effektivt, i stedet for generelle kretser som gjør alt litt, og det forskes på helt andre materialer og helt andre måter å regne på. Det hører hjemme i kapittel 8, og i appendiks A finner du historien om hvordan vi kom hit.

6.11 Oppsummering

Kapitlet begynte med en varm telefon og endte med en kurve som flater ut i 2004. Underveis fant vi det kapittel 5 ikke kunne vise fram: bryteren uten finger.

6.11.1 Hva som egentlig var nytt

6.11.2 Den ene ideen, fire ganger

Det samme svaret dukket opp fire ganger.

Hvorfor kommer ettallet fra forsyningen og ikke fra inngangen? Fordi transistoren er en bryter, og en bryter slipper gjennom noe annet enn seg selv.

Begrep	Kort sagt
Transistor	Bryter som styres av et elektrisk signal, ikke av en finger.
nMOS	Leder når styresignalet er 1.
pMOS	Leder når styresignalet er 0. Tegnes med boble.
VDD	Det høye punktet i kretsen. Logisk 1. Rundt 0,8 volt i dag.
GND (jord)	Det lave punktet. Logisk 0, og referansen alt måles fra.
CMOS	De to typene brukt sammen. All digital elektronikk i dag.
Inverter	To transistorer. Inngangen velger hvilken vei som åpnes.
Spenningsområde	0 og 1 er intervaller, ikke tall. Imellom ligger et belte der ingenting loves.
Støymargin	Hvor mye juks et signal tåler før det blir misforstått.
Signalgjenoppretting	Hver port lager et nytt, skarpt signal fra forsyningen.
Kapasitans	Badekaret som må fylles. Oppstår av seg selv, ingen vil ha den.
Portforsinkelse	Tiden en port bruker på å skifte. Rundt 5 ps.
Kritisk vei	Den lengste veien gjennom kretsen. Den bestemmer farten.
Klokkeperiode	Tiden mellom to like flanker. Må romme den kritiske veien.
Frekvens	én delt på klokkeperioden. Måles i hertz.
Moores lov	Dobling annethvert år. En bransjeplan, ikke en naturlov.
Lekkasje	Stengt er ikke helt stengt. Koster strøm hele tiden.
Varmeveggen	Spenningen kan ikke senkes mer, så frekvensen kunne ikke økes mer.

Tabell 6.3 Begrepene fra dette kapitlet.

Hvorfor tåler digitale signaler støy? Fordi hver port henter en fersk verdi fra forsyningen og kaster den slitne som kom inn.

Hvorfor tar en port tid? Fordi bryteren må fylle et kar, og kar tar tid å fylle.

Hvorfor blir brikken varm? Fordi hvert kar som fylles og tømmes koster en slurk, og slurken blir til varme.

Alle fire er følger av én setning: det sitter en bryter der, og den flytter ladning.

6.11.3 Videre til neste kapittel

Dette var bunnen. Vi gikk ned fra kodelinja i kapittel 1, gjennom tallene, instruksjonene, arkitekturen og portene, og nå står vi på silisiumet. Under oss ligger halvlederfysikken, som hører hjemme i et annet fag. Herfra går veien oppover igjen.

Vi tar med oss et problem. Du så at en addisjon kan ta 960 pikosekunder, og at klokka må romme den tregeste veien. Da er det naturlig å spørre hvor lang tid det tar å hente et tall fra minnet. Svaret er ubehagelig: rundt hundre ganger lengre enn å regne med det. En prosessor bruker mesteparten av tiden sin på å vente.

Kapittel 7 handler om de to triksene som gjør at den likevel kommer noen vei: å legge det man bruker ofte nærmest, og å begynne på neste oppgave før den forrige er ferdig.

6.12 Videre lesning

Englander dekker ikke dette stoffet. Det er greit å si rett ut, slik vi gjorde i forrige kapittel. Boka nevner transistoren noen få ganger i historiekapitlet og forklarer den aldri. I avsnitt 8.6 sier den selv at hensikten ikke er å diskutere den elektroniske implementasjonen, og henviser leseren videre til en ingeniørlærebok. Det er et poeng i seg selv: Englander trekker abstraksjonsgrensa akkurat der dette kapitlet begynner, så du har vært ett lag lenger ned enn pensumboka går.

Harris & Harris, *Digital Design and Computer Architecture*, kapittel 1. Den beste neste boka, og du ble sendt dit i forrige kapittel også. Denne gangen blar du tilbake til kapittel 1. Avsnitt 1.6 tar spenningsnivåer og støymargin med tall, 1.7 tar CMOS-transistorene og bygger NAND og NOR på transistornivå, og 1.8 tar strømforbruk. Du trenger ingen fysikkbakgrunn.

Patterson & Hennessy, *Computer Organization and Design*, avsnitt 1.5, 1.7 og 1.8. Her står den kanoniske framstillingen av akkurat det Englander mangler: hvordan brikker lages, Moores lov, varmeveggen med den berømte figuren, og hvorfor bransjen gikk over til flere kjerner. Skal du bare lese én ting etter dette kapitlet, les 1.7.

Petzold, *Code*. Den vennligste inngangen hvis dette kapitlet føltes bratt. Petzold bygger hele veien fra brytere og releer til en fungerende datamaskin, uten en eneste formel. Han bruker releer der vi bruker transistorer, og det er en fordel, fordi mekanismen er synlig.

Ben Eater, «Making logic gates from transistors», på nett. Et kvarter der han bygger porter av løse transistorer på et koblingsbrett og måler spenningene underveis. Det er overgangen fra kapittel 5 til dette kapitlet, gjort fysisk.

Chris Mack, «Fifty Years of Moore's Law». En kort og fritt tilgjengelig artikkel om hvorfor Moores lov aldri var en naturlov, og hvilke deler av den som sluttet å gjelde når.

Videre i denne boka: Appendiks D tar fysikken under transistoren: Ohms lov, halvledere, terskelspenning og transistoren brukt som forsterker. Appendiks E tar klokkefordeling på en hel brikke. Ingen av dem er pensum. Appendiks A forteller historien fra radiorør til silisium.

6.13 Refleksjon

1. Vi sa at kapasitansen er noe ingen vil ha, men som ikke kan unngås. Finn et eksempel fra et helt annet felt på en uunngåelig kostnad som følger av at noe er fysisk, og som alle jobber for å gjøre mindre.
2. Klokka får raske kretser til å vente på trege. Det sløser med tid, men gjør alt enklere å konstruere. Hvor ellers i datateknologien betaler man med ytelse for å få enklere regler?
3. Moores lov ble en selvoppfyllende profeti: bransjen planla etter spådommen, og derfor slo den til. Er det en styrke eller en svakhet ved en spådom at den virker på denne måten?
4. Da frekvensen stoppet i 2004, ble regningen sendt fra maskinvare til programvare. Hvem bestemte det? Og hva slags ansvar har en bransje for å fortelle brukerne sine at reglene har endret seg?

6.14 Selv-test

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

1. Hva er en transistor, sagt med én setning?
2. Hvor mange typer transistorer bruker vi i dette kapitlet, og hva skiller dem?
3. Hva betyr VDD, og hva betyr GND?
4. Hvilken logisk verdi svarer VDD til, og hvilken svarer GND til?
5. Hva betyr den lille boblen på styreledningen til en transistor?
6. Hvor mange transistorer er det i en inverter?
7. En inverter får inn 0. Hvilken av de to transistorene leder, og hva blir utgangen?
8. Hva står CMOS for, og hvorfor er navnet beskrivende?
9. Hvorfor går det nesten ingen strøm gjennom en CMOS-port som står i ro?
10. Hvor mange transistorer er det i en NAND-port? Og i en AND-port?
11. Hvorfor koster AND mer enn NAND, når AND ser ut som den enklere porten?
12. I en NAND-port står to transistorer i serie og to i parallell. Hvilke står hvor, og hvorfor gir det riktig oppførsel?
13. Forklar hvorfor 0 og 1 er spenningsområder og ikke enkelttall.
14. Hva er poenget med det udefinerte beltet mellom de to områdene?
15. En port får inn et signal som er slapt og litt rusket, men innenfor 1-området. Hvordan ser utgangen ut, og hvorfor?
16. Hvorfor blir en lydfil kopiert tusen ganger identisk med originalen, mens en kassett kopiert tusen ganger blir ubrukelig?
17. Hva er kapasitansen til en utgang, forklart med badekarbildet? Og hvorfor er det feil å si at noen har satt den inn med vilje?
18. Hva er portforsinkelse?
19. En portforsinkelse er 5 pikosekunder. Hvor mange pikosekunder går det på ett nanosekund, og hvor mange porter rekker å skifte på den tiden?
20. Hva er den kritiske veien i en krets, og hvorfor er den den eneste som betyr noe for farten?
21. En krets har tre veier fra inngang til utgang: én på 2 porter, én på 5 porter og én på 3 porter. Hver port tar 5 ps. Hvor lang tid bruker kretsen?
22. Hvorfor er svaret på en fulladder 15 ps og ikke 25 ps, når den har fem porter?
23. En OR-port har to innganger. Den ene er klar etter 5 ps, den andre etter 10 ps. Når er utgangen klar?
24. En mentekjede på 8 bit bruker 120 ps. Hvor lang tid bruker en på 32 bit, og hvorfor er svaret annerledes enn for de parallelle veiene i oppgave 21?
25. Hva er en klokkeperiode, og hva er en stigende flanke?
26. En prosessor går i 4 GHz. Hva er klokkeperioden i pikosekunder?
27. En klokkeperiode er 400 ps. Hva er frekvensen?
28. Hvorfor må klokkeperioden være minst like lang som den kritiske veien? Hva skjer hvis den ikke er det?

29. En port tar 5 ps, som skulle tilsi en frekvens på 200 GHz. Gi tre grunner til at ingen prosessor går så fort.
30. Hva sier Moores lov, og hvorfor er «lov» et misvisende ord?
31. Hvorfor teller forsyningsspenningen dobbelt i effektregnskapet, mens frekvensen bare teller én gang?
32. En brikke går i 2 GHz. Du dobler frekvensen og lar spenningen stå. Hva skjer med effekten?
33. Hva er lekkasje, og hvorfor er den et større problem i dag enn for tretti år siden?
34. Hvorfor kan man ikke bare fortsette å senke forsyningsspenningen for å spare strøm?
35. Forklar med egne ord hvorfor klokkefrekvensen sluttet å øke rundt 2004, og hvorfor svaret ble flere kjerner.
36. En laptop trekker 60 watt. Hvor mye varmer den rommet, og hvorfor er svaret så enkelt?
37. Hva betyr det egentlig når en produsent sier at en brikke er laget på en «3 nm-prosess»?
38. **Finn feilen.** En student skriver: «Klokka trengs fordi verdiene i kretsen lekker ut i løpet av noen nanosekunder, så vi må rekke å lagre dem før de forsvinner.» Hva er galt, og hva er den riktige grunnen?
39. **Finn feilen.** En annen student skriver: «AND-porter finnes ikke i en prosessor, bare NAND-porter.» Hva er riktig i påstanden, og hva er galt?
40. Kapittel 5 sa at det er verdt å spare porter. Forklar nå, med det du vet fra dette kapitlet, hva man faktisk sparer — og nevn tre ting en spart port koster mindre av.