

En Inngang til Digitalteknikk

Aria Alinejad

25. februar 2024

Innhold

1	Innledning	2
1.1	Abstraksjonsnivåer	2
1.2	Transistoranalogien	3
2	Logiske porter	4
3	Digital adderer	8

1 Innledning

Forskjellen mellom analog og digital elektronikk er ikke så stor som man kanskje skulle tro. I en viss forstand er hovedforskjellen bare hvilket abstraksjonsnivå vi arbeider på. I denne korte teksten vil jeg introdusere hovedkonseptene i digital elektronikk og vise hvordan de kan implementeres for å bygge en logisk krets.

Datamaskiner er ganske mystiske, og du lurer kanskje på hvordan man kan gå fra strøm og spenning (elektronikken du kjenner) til å bygge noe som kan løse problemer. Men jeg lover at når du er ferdig med dette dokumentet så vil du være i stand til å lage en logisk krets som kan gjøre enkle beregninger (en addisjonskrets). Jeg vil også gi et innblikk i hvordan nøkkelkonseptene kan brukes videre til å utvikle en datamaskin.

1.1 Abstraksjonsnivåer

Et viktig konsept innen digital elektronikk og ingeniørfag generelt er abstraksjonsnivåer. Du er kanskje ikke klar over det, men du er allerede kjent med dette konseptet fra hverdagen. Akkurat som i elektronikken finnes det flere nivåer i det virkelige liv.

Abstraksjonsnivåer

For å bestå masterstudiet må du:

bestå alle dine 5 år,
for å bestå et år må du bestå alle fagene dine,
for å bestå fagene dine må du bestå x antall prøver,
for å bestå prøven din må du få x% rett på spørsmålene.

I kretsverden:

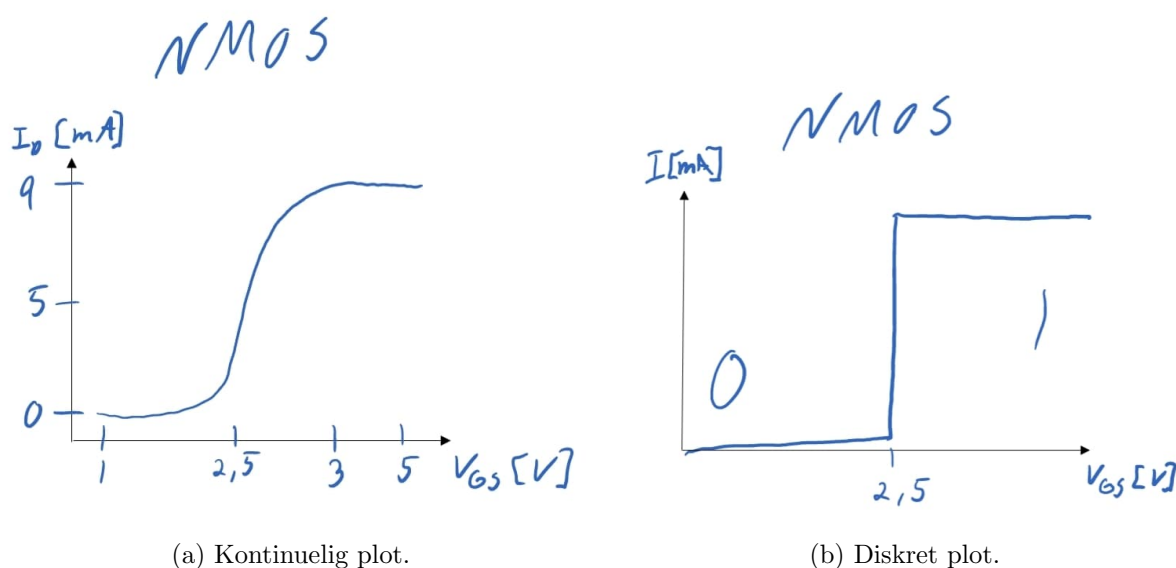
Blokknivå,
Logisk portnivå,
Transistornivå,
Spenningsnivå.

Dette er abstraksjonsnivåer. Når du prøver å finne ut om du har bestått masteren din eller ikke, kan du ikke sette deg ned og tenke på hvert eneste spørsmål du har svart på hver eneste prøve. På samme måte kan vi ikke begynne å beregne spenningen i hver enkelt ledning for å finne ut om datamaskinen vil fungere korrekt eller ikke. Når læreren bedømmer prøven som bestått eller ikke bestått (1 eller 0), bryr han seg bare om poengsummen din er over eller under 40%. På samme måte bryr transistoren seg bare om spenningen er over eller under et visst nivå (høy eller lav spenning) når den bestemmer seg for om strømmen skal passere eller ikke.

Når du vil vite resultatet av et fag, ser du bare på hvilke prøver som er bestått og ikke bestått. På samme måte, så ser vi bare på hvor det er lave eller høye spenninger (0 eller 1) når vi vil vite resultatet av en logisk krets med transistorer. Dermed er detaljene abstrahert bort.

Du er sikkert allerede kjent med transistoren og hvordan den fungerer i en krets. Forholdet mellom strøm og spenning i NMOS-transistoren er vist i figur 1a.¹ Hvis vi forenkler dette, kan vi se på det som noe som slipper gjennom all strømmen, eller stopper all strømmen basert på spenningen, se figur 1b.

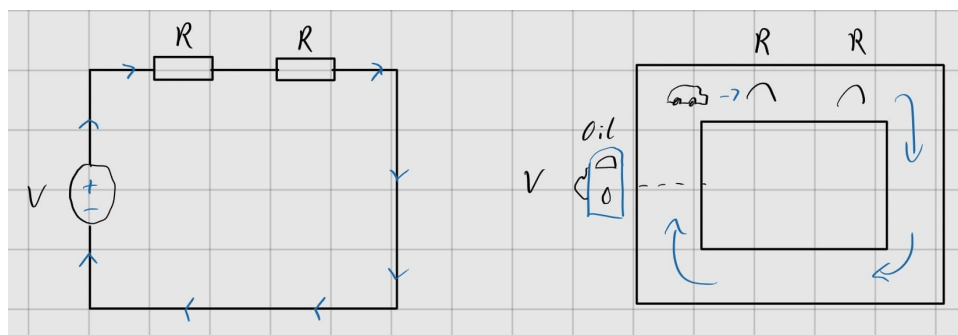
¹Du tenker kanskje at transistoren er ganske magisk sammenlignet med en motstand, men det som definerer en motstand, er at den har et lineært forhold mellom strøm og spenning. Det som definerer en transistor, er ikke noe annet enn et forhold mellom strøm og spenning som vist i figuren. Dette forholdet mellom strøm og spenning er tilfeldigvis veldig praktisk for mange formål.



Figur 1: Spenning plottet mot strøm for en vilkårlig NMOS transistor.

1.2 Transistoranalogien

Året er 1895. La oss nå tenke på en elektrisk krets som veier i byen, se figur 2. Foreløpig har vi bare motstander, altså fartsdumper. Men dette kan bare kontrollere hastigheten på bilene. Vi ønsker å kontrollere trafikkflyten, og til det trenger vi logikk. I bilverdenen trenger vi politibetjenter.



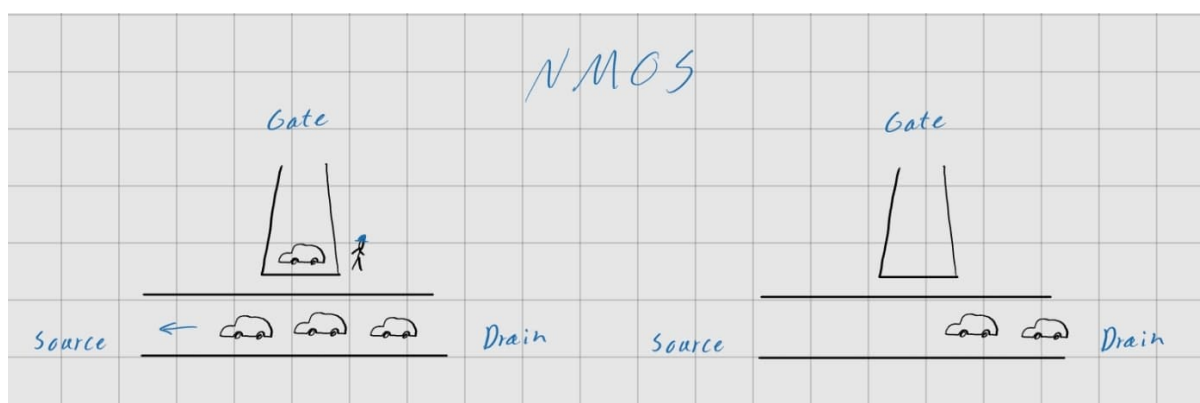
Figur 2: Figur av en enkel krets ved bruk av bil analogien.

Tenk på NMOS-transistoren som en vei som styres av en *Nice* politibetjent, se figur 3. Hvis det er en politimann til stede (høy spenning, **1**), kan du passere. Men hvis det ikke er noen politimann til stede (lav spenning, **0**), stoppes all trafikk. Vi kaller dette NMOS-veien. På PMOS-veien er det en *Påtrengende* politimann, se figur 4. Hvis denne betjenten er til stede (**1**), kan ingen passere, så trafikken flyter bare hvis det ikke er noen betjent (**0**) til stede.

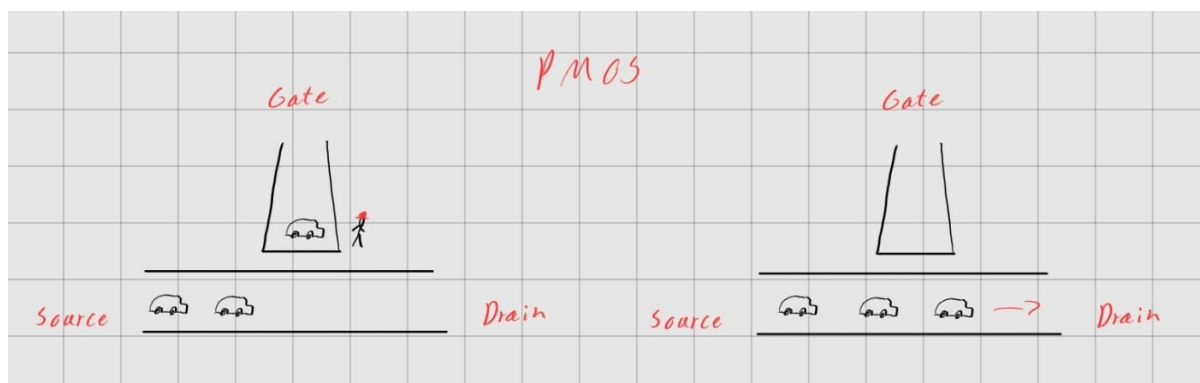
Politifolkene kan bare komme seg til posten sin hvis de har drivstoff (det vil si spenning), men på posten sin kan de ikke bevege seg, de har ingen strøm. Det er altså politiets spenning og ikke strømmen som avgjør om du kan kjøre eller ikke. Videre trenger bilene på hovedveien naturligvis drivstoff, spenning, for å kunne kjøre, strøm. Ingen spenning betyr ingen strøm, men ingen strøm betyr ikke ingen spenning (tenk drivstoff og kjøring ;).

Hva du burde sitte igjen med:

- Vi trenger transistorer for å lage logikk, for å kontrollere trafikkflyten.
- NMOS-veien leder strøm når det er høy spenning ved porten(1).
- PMOS-veien leder strøm når det er lav spenning ved porten(0).
- Det går ingen strøm mellom porten (gate) og de andre kanalene.
- Det er spenningen, ikke strømmen, ved porten (gate), som bestemmer trafikkflyten.
- Ingen spenning \rightarrow Ingen strøm, Ingen strøm \nrightarrow Ingen spenning.



Figur 3: Figur av NMOS transistor ved bruk av bil analogien.



Figur 4: Figur av PMOS transistor ved bruk av bil analogien.

2 Logiske porter

Nå har vi en fin transistor som vi kan leke oss med. Vi har en måte å kontrollere strømmen på, men hvordan lager vi logikk? Logikk handler om inngang og utgang. Språket i digitale kretser er spenning, ² 0 og 1. Uansett hvilken krets vi har, vil vi altså ha noe som kobler utgangen til høy eller lav spenning avhengig av inngangene (hvis de er koblet til høy eller lav spenning). 0 og 1 ut basert på 0 og 1 inn.

²Hvorfor er det slik? Vel, det er spenningen ved transistorens port (gate) som bestemmer strømmen. Transistorene styrer strømmen og skaper logikken. Det som betyr noe i en logisk krets, er altså høy og lav spenning, 0 og 1.

La oss begynne enkelt med en inverter (NOT port). Den gjør akkurat det den høres ut som, hvis den tar inn **0**, gir den ut **1**, og omvendt. La oss bryte det ned, **0** \rightarrow 1, og **1** \rightarrow **0**, se tabell 1. Vi starter med det vi vet må være der, en inngang og en utgang:

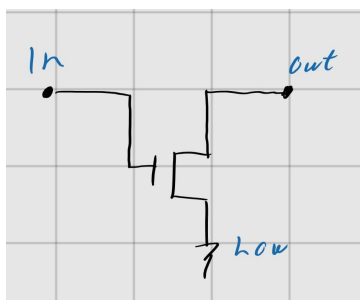
Tabell 1: Sannhetstabell for NOT port.

In	Out
0	1
1	0



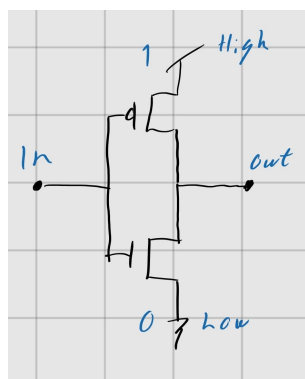
Figur 5: NOT port, første steg.

Vi ønsker at *out* skal være koblet til lav spenning (**0**) når inngangen har høy spenning (**1**). Derfor legger vi til en NMOS transistor (komponenten som tillater strøm ved høy inngang):



Figur 6: NOT port, andre steg.

Videre ønsker vi at *out* skal være koblet til høy spenning (**1**) når inngangen har lav spenning (**0**). Derfor bruker vi en PMOS transistor (komponenten som tillater strøm ved lav inngang):



Figur 7: Fullført NOT port.

Her er det viktig å legge merke til at vi må koble til en lav spenning for å få en lav spenning ut. Hvis vi ikke kobler utgangen til noe, får den ikke lav spenning, men forblir uendret. Vi må koble til lav spenninbg for å få lav spenning.

Hva er 0 volt?

Tenk på spenningen som høyden på et fjell. Høyden beregnes som meter over et gitt utgangspunkt. Dette utgangspunktet er definert av oss, og kan være havnivå eller kanskje foten av fjellet. Det som kan være 0 meter, kan like gjerne være 200 meter basert på hva vi definerer som nullpunktet vårt. Spenning er akkurat slik, null har ingen annen betydning enn at vi sier at det er null. Fysisk sett kan 0V og 5V være nøyaktig det samme, bare med ulike definisjoner av null, akkurat som høyden på et fjell. Det er naturlig at vi må koble et punkt til 5V hvis vi vil at det skal ha 5V, og dermed burde det være like intuitivt at vi må koble det til 0V hvis vi vil at det skal ha det.

- Spenning handler om forskjell i energi mellom to punkter, akkurat som høyde handler om forskjell i posisjon.

Deretter kan vi prøve å lage noe som er litt mer komplisert. Vi vil ha noe som kobler utgangen til **1** hvis begge inngangene, *A* og *B*, er koblet til **0** (AND port). La oss bryte det ned, **00** → **1**, **01** → **1**, **10** → **1** og **11** → **0**. Vi kan skrive dette i en tabell for å gjøre det mer lesbart, se tabell 2. Vi starter igjen en inngang og en utgang:

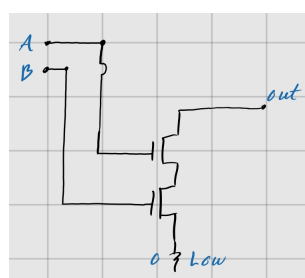
Tabell 2: Sannhetstabell for NAND port.

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0



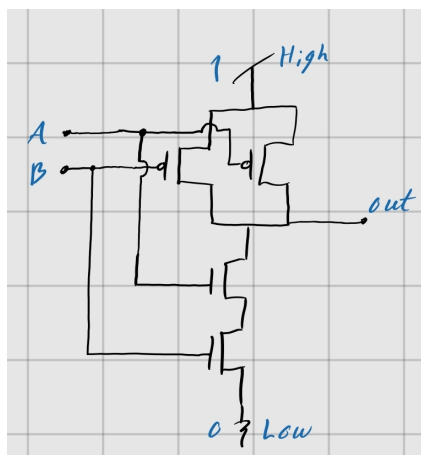
Figur 8: NAND port, første steg.

Videre ønsker du at *out* skal være koblet til lav spenning når begge inngangene er høye. Det er NMOS transistorens oppgave å koble når inngangen er høy. Hvis vi tenker på politimennene igjen, ønsker vi å lage en vei som bare kobles når begge *nice* politimennene er til stede. Dermed vil to NMOS-transistorer i serie gjøre jobben:



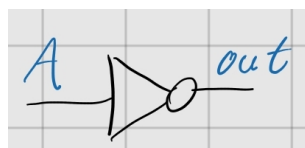
Figur 9: NAND port, andre steg.

Videre ønsker vi at *out* skal kobles til høy spenning hvis noen av inngangene er lav. Siden vi ønsker en tilkobling på lav inngang, må vi bruke en PMOS (*Påtrengende politibetjent*). Og vi ønsker å kunne passere selv om en av veiene er blokkert. Å ha muligheten til å velge flere veier gjør jobben. Vi kobler PMOS transistorene i parallel:

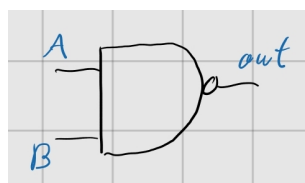


Figur 10: Fullført NAND port.

Dette er nå en port som bare gir oss **0** hvis begge inngangene er **1**, det er det jeg kaller logikk! NAND porten og inverteren er viktige byggesteiner i logikk. Når vi lager større logiske kretser med mange forskjellige porter, er det vanlig å abstrahere bort detaljene om hvordan transistorene er plassert. Figur 11 og 12 viser hvordan NOT og NAND portene kan tegnes på portnivå. I digital design er det et begrenset antall byggeklosser, akkurat som i LEGO. Figur 13 viser alle portene med sine sannhetstabeller som beskriver hvordan de fungerer, se om du kan lage dem av transistorer med samme metode som du brukte til å tegne NOT og NAND portene.³



Figur 11: Tegning av NOT port på portnivå.



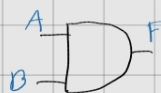
Figur 12: Tegning av NAND port på portnivå.

³AND porten kan faktisk lages ved å rett og slett sette en NOT port etter en NAND port, ta en titt på sannhetstabellene og se om dette gir mening.

AND

$$A \cdot B = F$$

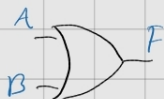
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



OR

$$A + B = F$$

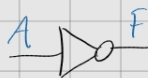
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1



NOT

$$\overline{A} = F$$

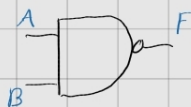
A	F
0	1
1	0



NAND

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

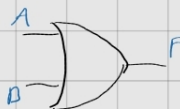
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



NOR

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

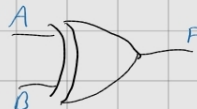
A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



XOR

$$A \oplus B = F$$

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



For å hjelpe deg med å huske portene og hvorfor de er nyttige:

- **AND:**
Når gir det 1? Når både dette OG det er 1.
- **OR:**
Når gir det 1? Når dette ELLER det er 1.
- **NOT:**
Når gir det 1? Når dette IKKE er 1.
- **NAND:**
Når gir det 1? Når AND IKKE gjør det.
- **NOR:**
Når gir den 1? Når OR IKKE gjør det.
- **XOR:**
Når gir den 1? Når det er et oddetall antall 1'ere.

Figur 13: Navn, ligning, sannhetstabell og symbol for alle logiske porter.

3 Digital adderer

Datamaskiner handler om å automatisere oppgaver, og noen av de første datamaskinene ble brukt til å automatisere utregninger (kalkulatorer).⁴ En adderer er en utrolig viktig komponent i mesteparten av digital elektronikk, og ved å bruke komponenter du kjenner til, får du nå muligheten til å desige en krets som bruker høye og lave spenninger til å beregne summen av to tall! Dette er det første åpenbare skrittet mot å lage datamaskiner som kan løse problemer for oss.

Vi skal prøve å konstruere noe som kalles en halv-adder. Vi begynner med å klargjøre for oss selv hva vi ønsker at systemet skal gjøre. Gitt to binære tall A og B ønsker vi at kretsen skal gi ut summen.

$$A + B = out \quad (1)$$

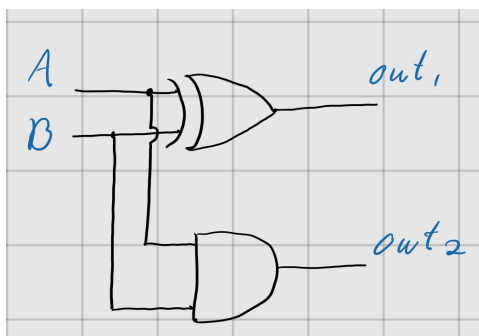
⁴Faktisk var ordet datamaskin tidligere stillingsbetegnelsen til en person som jobbet med å gjøre mange utregninger for hånd!

Her indikerer "+"-symbolet *addisjon* og ikke logisk *OR*. Siden vi arbeider binært, skal resultatet være: $0 + 0 = 00$, $1 + 0 = 01$, $0 + 1 = 01$, $1 + 1 = 10$. Vi kan igjen skrive dette i en tabell for å gjøre det mer oversiktlig, se tabell 3.

Tabell 3: Sannhetstabell for 2-bit adderer.

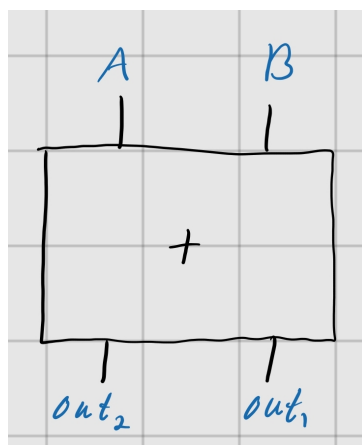
A	B	out ₂	out ₁
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Siden vi har to utganger, trenger vi nå to logiske porter. Hvis vi ser på out₂, ser vi at sannhetstabellverdiene er de samme som for en AND porten. Mens out₁ har de samme verdiene som en XOR port. Det betyr at vi kan realisere denne 2-bit addereren ved hjelp av en XOR port og en AND port. Figur 14 viser hvordan dette kan tegnes ved hjelp av symbolene for portene (som vist i figur 13).



Figur 14: Realisering av 2-bit adderer ved hjelp av logiske porter.

På samme måte som disse symbolene abstraherer bort detaljene i hvordan transistorene er plassert, kan vi abstrahere enda mer ved å bare tegne en boks med inngangene og utgangene til denne 2-bit addereren. Vi kaller dette abstraksjonsnivået for blokknivå, se figur 15.



Figur 15: Figur på blokknivå av 2-bit adderer.

Vi har nå designet en digital krets som kan addere to tall! Designet av denne kan utvides, ved hjelp av samme framgangsmåte, for å lage en blokk som kan addere binære tall av vilkårlig lengde, som er starten på konstruksjonen av en kalkulator, eller en enkel datamaskin om du vil. Og til syvende og sist handler det hele bare om høye og lave spenninger.

Å abstrahere bort detaljene er det som får digitale kretser til å virke så magiske, på samme måte som for ett trylletriks