TFE4101 Krets- og Digitalteknikk

Håvard Krogstie
krogstie.havard@gmail.com

23. august 2020

Dette dokumentet inneholder forklaringer, definisjoner, metoder og formler brukt i TFE4101 Krets- og Digitalteknikk ved NTNU våren 2020 (Den med Corona). Det er en personlig gjennomgang av det som virket relevant, og utfyllende nok til at jeg ikke skal trenge noe annet på hjemmeeksamen. Dokumentet inneholder kretsteori, metoder for regning på kretser, teorien bak kondensatorer og spoler, dioder, transistorer, CMOS, logiske kretser, boolsk algebra, binærtallsregning, timing av digitale kretser og sekvensielle kretser. Seksjonene blir mer utfyllende underveis.

Dokumentet har jevnt over ikke kildehenvisninger, og er ikke vurdert av noen med kunnskap i faget. Du finner heller ikke øvingskok her. Les på eget ansvar. Kildekode og nyeste utgave på github.com/haved/notater. Skrivefeil, faktafeil, mangler eller uklaheter? Opprett en issue eller skriv en e-post!

Innhold

1 Kretser

Kretser er samlinger med komponenter koblet sammen. Kretsen kan modeleres som en graf der koblinger er noder, og komponentene blir kanter mellom nodene. Ofte ser vi ikke på ledninger som komponenter, men bare som ideelle koblinger / noder. Noder med kun ledninger i mellom blir dermed samme node. Det går strøm mellom nodene gjennom komponentene som binder dem sammen. Nodene har også en egenskap kalt potensiale. Vi har lover for å regne på disse.

1.1 Strøm og ladning

Ladning er en elektromagnetisk egenskap ved blant annet elektroner. Den måles i Coulomb (C). Strøm betyr flyt av ladning over tid, og måles i ampere (A) etter formelen

$$1\,\mathrm{A} = 1\,\mathrm{C\,s^{-1}}$$

Strøm går altså gjennom en ledning eller et komponent, og kan måles ved å erstatte ledningen med et amperemeter. Vi kaller ofte strøm for I eller i.

1.2 Spenning

Spenning er en differanse i potensiale mellom to noder i en krets. Spenning måles i Volt (V), og er energi per ladning ($\mathrm{J\,C^{-1}}$). Spenning må altså alltid ses som en potensialforskjell mellom to steder i en krets, så det er ikke egentlig mulig å ha spenning i et bestemt punkt, med mindre et annet punkt er definert til å være nullpunktet, jord (GND). Vi kaller ofte spenning for V eller v. Andre bruker U for spenning.

1.3 Effekt

Observer at

$$1\,\mathrm{V} \cdot 1\,\mathrm{A} = 1\,\mathrm{J\,C^{-1}} \cdot 1\,\mathrm{C\,s^{-1}} = 1\,\mathrm{J\,s^{-1}} = 1\,\mathrm{W}$$

Dette gir oss formelen for effekt i et komponent.

$$P = V \cdot I$$

1.4 Kirchoffs strømlov

Kirchoffs strømlov (KCL) sier at summen av strømmer inn i en node er alltid lik 0. Hvis det kommer 5 A inn i en node, må det altså gå 5 A ut igjen òg.

En node kan ha vilkårlig mange kanter ut av seg, og en denne definisjonen på node kan også inneholde komponenter. Det viktige er at du teller alle ledninger som går inn og ut av noden.

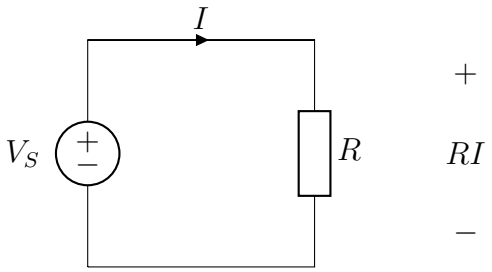
Et resultat av KCL er at det ikke kan gå strøm i en åpen krets, altså en ledning som bare er koblet til noe i éne enden.

1.5 Kirchoffs Spenningslov

Kirchoffs spenningslov (KVL) sier at summen av spenningsfallet over enhver løkke må være lik 0. Spenningsfall over et komponent er potensialforskjellen mellom nodene den er koblet til. En løkke er en sti gjennom kretsen som stopper der den starter, og KVL sikrer at potensialet til startnoden og stoppnoden er det samme, hvilket det burde være siden de er samme node.

1.6 Passiv fortegnskonvensjon

Når man summerer spenningsfall rundt en løkke brukers positive ledd for spenningsfall i referansestrømretningen (I). Dette betyr i praksis at alle motstander har et spenningsfall $R \cdot I$. Spenningskilder får ofte negativt spenningsfall (i strømretningen), siden de som regel bidrar med energi. Husk at spenning er energi per ladning, og hvis hver ladning har høyrere energi etter å ha passert gjennom komponentet betyr det at energi er blitt tilført.



Figur 1: Eksempel på regning med passiv fortegnskonvensjon

Som eksempel bruker KVL på kretsen i Figur 1 for å finne effekten levert av spenningskilden.

$$-V_S + R \cdot I = 0$$

Observer her at V_S er satt inn negativt. Det er fordi V_S er *spenningsstigningen* langs strømretningen. Når vi skal summere sammen *spenningsfall* blir det derfor negativt.

Når vi skal regne effekten gjennom spenningskilden, må vi bruke $P = V \cdot I$ der V er spenningsfallet langs I. Derfor setter vi inn $V = -V_S$

$$P = VI = -V_S I = -RI^2$$

Dette gir oss en negativ effekt på spenningskilden, som betyr at den leverer energi.

Merk. Man kan ha valgt referansestrømretningen "feil vei" slik at I er negativ, men isåfall blir også spenningsfallet $R \cdot I$ over motstanden negativ i den retningen, som er riktig. Resultatet blir uansett at effekten i motstanden, $V \cdot I$, alltid er positiv. Effekten i en kilde som tilfører energi blir alltid negativ.

2 Kilder

Det kan være mange komponenter som tilfører strøm eller spenning. Vi ser på 4 idealiserte utgaver. Uavhengige strøm- og spenningskilder leverer henholdsvis strøm og spenning. Uansett hva som skjer ellers i kretsen garanterer kildene en viss strøm gjennom en ledning, eller en viss potensialforskjell mellom noder. Avhengige kilder gir samme garanti, men hvilken strøm/spenning som leveres er avhengig av noe. For eksempel kan en spenningskilde levere 5 ganger spenningsfallet over en gitt motstand.

På grunn av lovene vil det gå ofte gå strøm gjennom en spenningskilde, og en strømkilde kan ha spenningsfall. På grunn av den passive fortegnskonvensjonen vil ofte spenningskilder ha negativt spenningsfall, men det er ikke sikkert. F. eks. i serie med en motsatt rettet sterkere spenningskilde.

Det finnes også AC-kilder som leverer vekselstrøm med oppgitte frekvenser og bølgeformer.

2.1 Nulle ut

Man kan ønske å nulle ut en kilde slik at den ikke bidrar (f.eks. i superposisjon). En spenningskilde blir en kortslutning for å garantere $V = 0$, mens en strømkilde blir en åpen krets for å garantere $i = 0$.

2.2 Effekt

For å regne effekt levert av en kilde bruker man $P = V \cdot I$. Det er viktig at spenningsfallet over kilden er målt i samme retning som strømmen. Ofte vil

dette gi et negativt spenningsfall, som gir en negativ P. Det betyr at kilden leverer effekt. Hvis P er positiv bruker den effekt (er en last).

3 Motstand

Motstand er en egenskap ved komponenter som begrenser flyten av strøm. Forholdet mellom strøm (I), spenning (V) og motstand (R) er gitt ved Ohms lov.

$$V = R \cdot I$$

Motstanden måles i Ω. En ideell motstand har samme R for alle I. Når vi bruker komponentet vi kaller motsand, regner vi med at den er ideell. $I(V)$ er da en proposjonal funksjon, og grafen er en linje. Andre komponenter, slik som lysdioder, har helt andre grafer for $I(V)$-funksjonen.

3.1 Effekt

Ohms lov kombineres med formelen for effekt.

$$P = V \cdot I = RI^2 = \frac{V^2}{R}$$

Disse formlene gjelder i enhver last. I en vanlig motstand blir effekten om til varmeenergi.

3.2 Seriekobling

Seriekoblede mostander oppfører seg som én motstand R_{eq} der

$$R_{eq} = R_1 + R_2$$

3.3 Parallellkobling

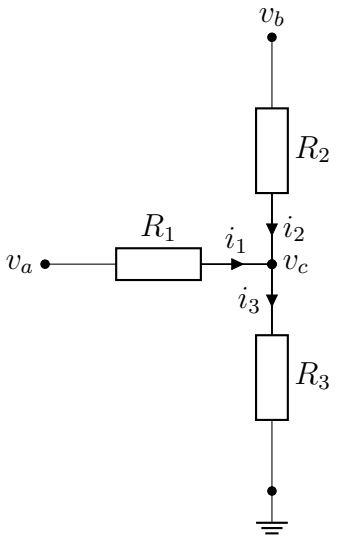
Parallellkoblede motstander må hva samme spenningsfall over hver motstand, men strømmen blir forskjellig. To parallellkoblede motstander R_1 og R_2 vil oppføre seg identisk med én motstand R_{eq} der

$$\begin{aligned} \frac{1}{R_{eq}} &= \frac{1}{R_1} + \frac{1}{R_2} \\ R_{eq} &= \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}} \\ &= \frac{R_1 R_2}{R_1 + R_2} \end{aligned}$$

Strømmen får flere veier å gå, så R_{eq} er mindre enn både R_1 og R_2.

4 Nodespenningsmetoden

Begynn med å definere en node som jord. Dette gir oss et potensiale i alle andre noder relativt til nullspenningen. Bruk Ohms lov og spenningsfallet mellom noder for å regne strømmen som går mellom nodene. For hver node setter du at summen av strømmene inn i noden må være 0 (KCL). Se Figur 2.



Figur 2: Nodespenningsmetoden

Dette utsnittet av en krets gir oss følgende formler:

$$i_1 = \frac{v_a - v_c}{R_1} \quad i_2 = \frac{v_b - v_c}{R_2} \quad i_3 = \frac{v_c - 0\,\mathrm{V}}{R_3}$$

Satt sammen med KCL for noden v_c:

$$i_1 + i_2 - i_3 = 0$$
$$\frac{v_a - v_c}{R_1} + \frac{v_b - v_c}{R_2} - \frac{v_c - 0\,\mathrm{V}}{R_3} = 0$$

5 Superposisjon

I en krets med kun lineære komponenter (motstander, kilder, kondensatorer og spoler) kan man regne utslagene til hver kilde for seg, ved å nulle ut alle

andre kilder og regne strømmer og spenningsfall. Etter å ha regnet for hver kilde kan man addere sammen for å finne totale strømmer og spenningsfall.

6 Thévenin-ekvivalenten

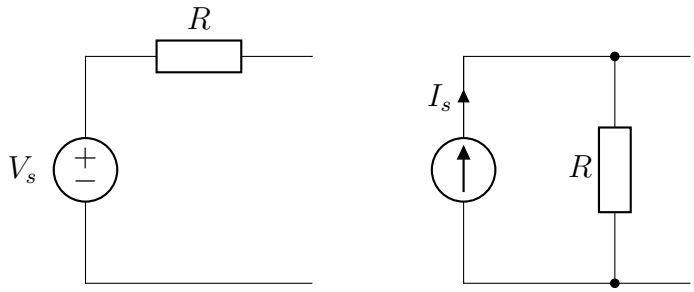
Enhver lineær krets med kun kilder og motstander vil sett fra to terminaler kunne erstattes med en enkel krets med kun en spenningskilde V_{Th} og en motstand R_{Th}. Vi kan finne V_{Th} ved å se på spenningen over den åpne kretsen mellom terminalene i den opprinnelige kretsen. Vi kan finne R_{Th} ved å nulle ut alle kilder og måle R_{eq} over terminalene. Vi kan også regne ut R_{Th} ved å se på strømmen over terminalene ved kortslutning: $R_{Th} = \frac{V_{Th}}{I_{SC}}$

7 Kildetransformering

En spenningskilde V_S i serie med en motstand R oppfører seg akkurat likt som en strømkilde I_S i parallell med like stor motstand R, der forholdet mellom V_S og I_S er gitt ved

$$V_S = R \cdot I_S$$

Merk at retningen bevares, se Figur 3.



Figur 3: Kildetransformering

8 Kondensatorer

Kondensatorer består av to ledende plater (elektroder) separert av en isolator, ofte laget av et dielektrisk element. Kapasitansen C måles i farad (F) og er gitt ved arealet til platene A, avstanden mellom platene d, og isolatorens permittivitet ϵ.

$$C = \frac{A\epsilon}{d}$$

Det går ikke strøm mellom platene, men platene kan lades opp med en spenningsforskjell v. Da får platene en ladning på $+Q$ og $-Q$, gitt ved

$$Q = C \cdot v$$

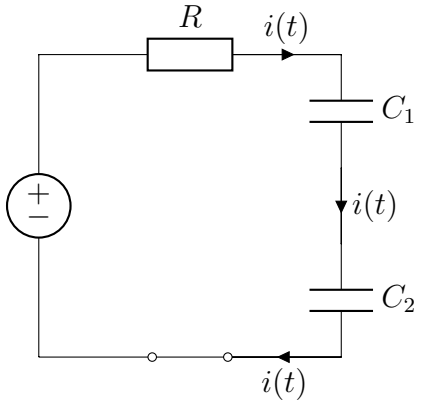
Dette gir oss at $1\,\mathrm{F} = 1\,\mathrm{C}\,\mathrm{V}^{-1}$. Dette er svært mye, så kondensatorer opererer gjerne med µF. Den deriverte av ladning med hensyn på tid $\frac{dQ}{dt} = i(t)$, så vi kondensatorligningen på differensialform:

$$i(t) = C \cdot \frac{dv}{dt}$$

Dette betyr at strømmen er proposjonal med forandring i spenning. I en krets vil ikke spenningen over en kondensator kunne endres momentant, siden det ville gitt uendelig høy strøm. Spenningen endrer seg gradvis, og når den har nådd målspenningen vil $i(t) = 0$. Dette kalles *steady state*.

8.1 Regning på kondensatorer

I en krets vil platene i en kondensator alltid være motsatt ladet, altså $+Q$ og $-Q$. Selv om kondensatoren ikke lar strøm passere, vil det derfor likevel virke slik når den lades opp eller ut. Kirchovs strømlov gjelder fortsatt, så man kan ikke tilføre strøm til én plate uten at det kommer like mye strøm ut fra den andre platen. Dette høres kanskje trivielt ut, men det er lett å tro at strømmen mellom to kondensatorer lever sitt eget liv, og at ladning kan bevege seg fritt mellom platene på innsiden. Figur 4 viser at $i(t)$ oppfører seg helt vanlig sett utenifra.



Figur 4: Kirchovs strømlov gjelder fortsatt for kondensatorer

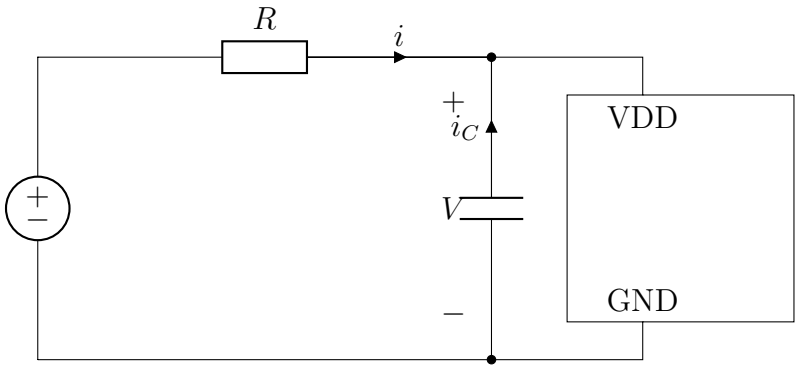
I en AC-krets vil det se ut som strøm passerer gjennom kondensatoren.

8.2 Eksempler på kondensatorer

Én type er små gule/brune kjeramiske kondensatorer. De har ikke polaritet. Det finnes også sylinderformede elektrolyttkondensatorer som har polaritet. I tillegg er det verdt å merke seg at alle ledninger med isolasjon mellom seg vil ha en viss kapasitans. Dette betyr at nærhet mellom ledninger kan påvirke hvor raskt signaler (spenning) i ledningene klarer å endre seg. En ledning med kontinuerlig strøm vil også indusere et magnetfelt rundt seg, se seksjon 9 om spoler.

8.3 Eksempelbruk av kondensator

Kapasitans kan tenkes på som en treghet i spenningsfall. Et eksempel på bruk er mellom VDD og jord på en chip. Se Figur 5. Chippen kan gjøre mye forskjellig, og mengden strøm som brukes kan variere. Når strømforbruket i øker vil spenningsfallet over R øke, men chippen har lyst på så jevn spenning som mulig. Derfor har vi en kondensator som motvirker forandring i V. Kondensatoren tilfører ekstra strøm i_C når chippen trenger det, slik at i blir så jevn som mulig. Spenningsfallet over R blir dermed mer stabilt. Når chippen bruker mindre strøm vil kondensatoren ta til seg strøm for å holde i stabil.



Figur 5: Kondensator som spenningsstabisator

Kondensatoren tar på ingen måte bevisste valg om å gi og ta strøm. Det er et resultat av at spenningsfallet over kondensatoren er proposjonalt med ladningen den har. For å endre spenning må den strømme inn eller ut ladning. Kondensatoren "prøver" altså å tilpasse seg forandringer i spenningen rundt, men må først kvitte seg med ladning. Denne strømmen motvirker forandringene, men kondensatoren lades samtidig opp eller ut. Dersom chippen over tid bruker mer strøm, vil spenningen over kondensatoren etterhvert

stabiliere seg ved den nye standaren. Kondensatoren motvirker altså raske forandringer, men er ikke en spenningsregulator.

8.4 Energi

Mengden energi i en kondensator regnes med formelen

$$E_C = \frac{1}{2}Cv^2$$

8.5 Seriekobling

Seriekoblede kondensatorer oppfører seg som én kondensator der avstanden mellom platene summeres, så kapasitansen blir lavere.

$$C_{eq} = \frac{1}{\frac{1}{C_1} + \frac{1}{C_2}} = \frac{C_1 C_2}{C_1 + C_2}$$

8.6 Parallellkobling

Parallellkobling av kondensatorer gir mer plate å lade. Kapasitansen blir større.

$$C_{eq} = C_1 + C_2$$

9 Spoler

TODO: Skriv

9.1 Seriekobling

TODO: Skriv

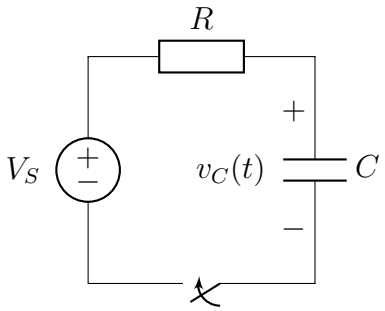
9.2 Parallellkobling

TODO: Skriv

10 RC-krets

En RC-krets er en enkel krets med en spenningskilde, en motstand og en kondensator i serie. Andre kretser kan omdannes til RC-kretser for å enklere

kunne regne på spenning over kondensatoren over tid. Ofte vil vi kjenne til spenningen over kondensatoren $v_C(t)$ ved en tid $t = 0$. I det øyeblikket vil en bryter flippes, og kretsen endres. Det enkleste eksempelet er en krets der vi begynner med en spenning $v_C(0) = 0$, og en spenningskilde V_S kobles på ved $t = 0$. Se Figur 6.



Figur 6: En enkel RC-krets

Ved bruk av Kirchovs lover kan vi finne en differensiallgining for v_C. Den spesifikke løsningen blir.

$$v_C(t) = V_s(1 - e^{-\frac{t}{\tau}})$$

Der $\tau = RC$. Detter kaller vi tidskonstanten. Når $t = \tau$ vil $v_C(\tau) = V_s(1 - \frac{1}{e})$. Etter uendelig lang tid vil $v_c(t)$ gå mot V_S.

Mer generelt dersom $v_c(0) = V_0$ og spenningskilden byttes til V_s på ved $t = 0$.

$$v_c(t) = V_s + (V_0 - V_s)e^{-\frac{t}{\tau}}$$

11 RL-krets

$\tau = \frac{L}{R}$ TODO

Digitalteknikk

12 Dioder

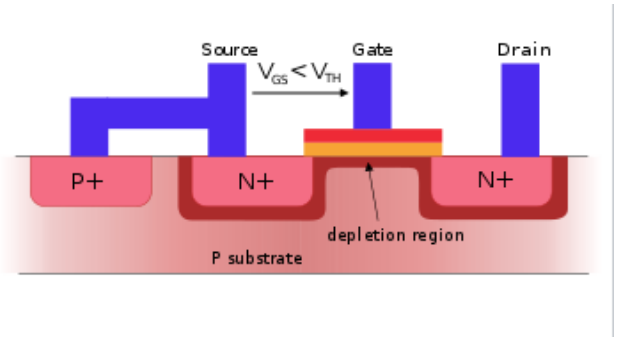
TODO:Skrive om doping, hull og spenningsbarriærer.

13 Transistorer

Transistorer er komponenter som bruker en strøm eller spenning til å regulere flyten av strøm eller spenning. De kan brukes både som forsterkere og brytere.

13.1 MOSFET

MOSFET (Metal-oxide-semiconductor field-effect transistor) er en type transistor som bruker dopede halvledere for å kontrollere strøm. Se seksjon 12 om dioder. MOSFET tegnes ofte som et vindu med gardiner, der vinduet er p/n-dopet, og gardinene er motsatt n/p-dopet. Det er dopingen på gardinene som bestemmer om det er pMOS eller nMOS. Se figur 7.



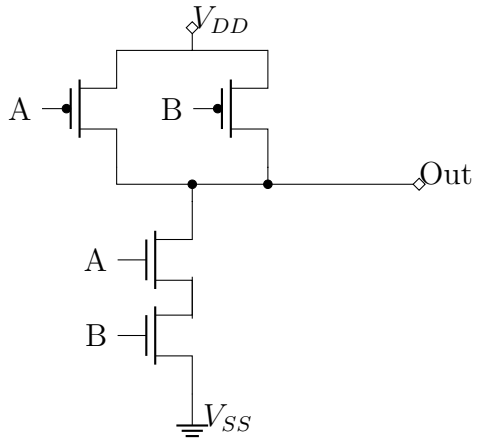
Figur 7: Tverrsnitt av nMOS fra Wikipedia

Til den éne gardinen kobles terminalen *source* (S). Til den andre gardinen *drain* (D). Strøm klarer ikke å passere mellom dem til vanlig, siden man får spenningsbarriærer mellom viduet og gardiene. En tredje terminal, *gate* (G) er koblet til en **metall**plate med et isolerende **oksid**lag, som "kobler" source og drain. Gate er isolert, så det går ikke faktisk strøm, men platen kan lades opp likt en kondensator, der vinduet er den andre platen. Viduet må da være koblet opp til en fjerde terminal, *body* (B), slik at gate har en spenning å være relativ til. Når gate og body har en spenningsforskjell riktig vei over terskelspenningen V_{TH}, dannes en bru av positiv eller negativ ladning mellom

gardinene. Avhengig av dopingen lar én av disse strøm passere mellom source og drain. Ofte har transistorer bare tre terminaler. Det er fordi body kobles opp til source. Da måles gatespenning V_{GS} relativt til sourcespenning V_S:
pMOS lukkes mellom *source* og *drain* når $V_{GS} \ll V_S$
nMOS lukkes mellom *source* og *drain* når $V_{GS} \gg V_S$
Her betyr $\ll$ en differanse på mer enn terskelspenningen V_{TH}. Eksempel på terskelspenning: 0.45 V.

13.2 CMOS

CMOS (Complementary Metal-oxide-semiconductor) er et system for design av logiske kretser med MOSFET. Det baserer seg på symmetriske par av pMOS og nMOS koblet opp til V_{DD}, V_{SS}, inputs og hverandre. V_{DD} er drain, og er høy (f.eks. 5 V). V_{SS} er source, som er lav, altså jord.



Figur 8: En NAND-port i CMOS

CMOS følger regler for oppsett. En **pMOS** sin *source* er alltid koblet til V_{DD} eller til en annen pMOS sin *drain*. En **nMOS** sin *source* er alltid koblet til V_{SS} eller til en annen nMOS sin *drain*. pMOS brukes altså til opptrekk av output, og nMOS brukes til nedtrekk av output. *Complementary* betyr at kun én av trekkene på output kan være lukket. Se figur 8.

13.2.1 Viktig å huske om CMOS

- CMOS bruker kun strøm på å endre ladning i gate i MOSFETene. Det betyr at det kun går strøm til input når input endrer seg.

- V_{DD} er drain, men er altså høy. V_{SS} er source, men er altså lav/jord.

- En pMOS sin *source*-terminal kan aldri kobles til V_{SS}, men kan kobles til V_{DD}. En nMOS er omvendt. Dette gir mening med tanke på at *gate* skal sammenlignes med *source*.

14 Boolsk algebra

I boolsk algebra er alle verdier enten 0 eller 1. Et boolsk uttrykk består av slike boolske verdier og boolske operatorer. En boolsk funksjon tar parametere og evaluerer til enten 0 eller 1. Funksjoner kan representeres både som et boolsk uttrykk der parameterne forekommer som literaler ($x+yz$), og som en sannhetstabell. Operatorer er i seg selv boolske funksjoner.

14.1 Unary operatorer

Unary betyr at operatoren er en funksjon med ett parameter. Det finnes kun 4 unary operatorer $F(x)$ i boolsk algebra:

Navn	Symbol	F for $x=$		Uttrykk	Kommentar
		0	1		
Zero		0	0	$F_0 = 0$	Konstant 0
Ident.	x	0	1	$F_1 = x$	Identitet
Compl.	$\bar{x}/x'$	1	0	$F_2 = \bar{x}$	Komplement
One		1	1	$F_3 = 1$	Konstant 1

14.2 Binære operatorer

Binære operatorer kalles binære fordi de tar inn to operander (ikke fordi operandene er binære verdier). Det finnes $2^{2^2} = 16$ forskjellige boolske funksjoner $F(x, y)$, men vi bryr oss ikke om alle.

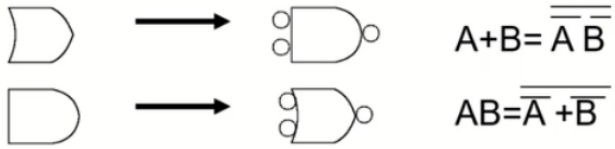
Navn	Symbol	F for $x, y=$				Uttrykk	Kommentar
		$0,0$	$0,1$	$1,0$	$1,1$		
Zero		0	0	0	0	$F_0 = 0$	Konstant 0
AND	$x \cdot y$	0	0	0	1	$F_1 = x\,y$	x og y
XOR	$x \oplus y$	0	1	1	0	$F_6 = x\,\bar{y} + \bar{x}\,y$	enten x eller y
OR	$x + y$	0	1	1	1	$F_7 = x + y$	x eller y
NOR	$x \downarrow y$	1	0	0	0	$F_8 = \overline{x + y}$	Not-OR
Equiv.	$x \odot y$	1	0	0	1	$F_9 = x\,y + \bar{x}\,\bar{y}$	x = y
NAND	$x \uparrow y$	1	1	1	0	$F_{14} = \overline{x\,y}$	Not-AND
One		1	1	1	1	$F_{15} = 1$	Konstant 1

14.3 Regneregler

Vi har en rekke regneregler på operatorene i boolsk algebra. Med disse kan vi omskrive boolske uttrykk til enklere eller tydeligere uttrykk. Alle boolske uttrykk kan skrives som

14.3.1 De Morgans teoremer

De Morgans teoremer gir oss en sammenheng mellom AND og OR, ved å invertere både input og output.



Hvis vi starter med komplementet gir dette oss følgende

$$\bar{x}\,\bar{y} = \overline{x + y}$$

$$\bar{x} + \bar{y} = \overline{x\,y}$$

Vi kan også gjøre flere i slengen:

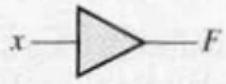
$$\overline{x + y + z} = \bar{x}\,\bar{y}\,\bar{z}$$

14.4 Forenkling av uttrykk

Vi ser på representasjonformer og måter å forenkle boolske uttrykk i seksjon 20 om boolske funksjoner.

15 Logiske porter

Med CMOS (Seksjon 13.2) kan man lage porter som utfører logiske operasjoner. Se tabell 1. Portene bruker et visst antall transistorer og har en forplantningsforsinkelse. Se Seksjon 23 om teknologimapping for effektiv bruk av porter i logiske kretser.

NAME	GRAPHIC SYMBOL	FUNCTIONAL EXPRESSION	COST (NUMBER OF TRANSISTORS)	GATE DELAY (NS)
Inverter		$F = x'$	2	1
Driver		$F = x$	4	2
AND		$F = xy$	6	2.4
OR		$F = x + y$	6	2.4
NAND		$F = (xy)'$	4	1.4
NOR		$F = (x + y)'$	4	1.4
XOR		$F = x \oplus y$	14	4.2
XNOR		$F = x \odot y$	12	3.2

Tabell 1: Logiske porter, Figur 3.14 i Gajski

16 Forsinkelser

Komponenter har alltid en viss kapasitans som gjør at spenningsendriger ikke kan skje momentant. Derfor må vi ta hensyn til forsinkelser, også i digitale

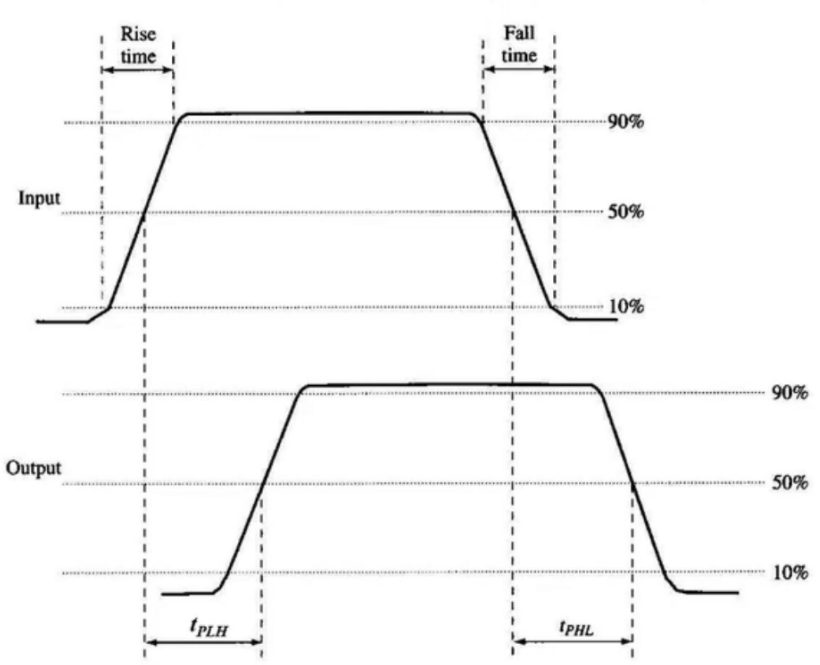
kretser, der signaler til tider vil befinne seg "mellom" 0 og 1. Vi opererer med flere typer forsinkelser. For å finne verdiene for et gitt komponent må vi se på komponentets datablad. Se seksjon 21.

16.1 Eksempel med CMOS-inverter

TODO: Sett inn eksempelet med CMOS-inverter og kondensatorer.

16.2 Forplantningstid

Forplantningstid (Propagation delay) er tiden det tar fra input har endret seg, til output endrer seg. Oftest ser man på tiden det tar fra input har passert 50% til output passerer 50%. Man deler forplantningstiden inn i t_{PLH} (Output går lav til høy) og t_{PHL} (Output går høy til lav). Se figur 9. Vi skiller mellom t_{PHL} og t_{PLH} siden de kan være forskjellige, men vi har også $t_P = \frac{t_{PHL}+t_{PLH}}{2}$ som vi bare kaller "forsinkelse gjennom porten".



Figur 9: Tider målt på en enkel port med input og output

Forplantningstid måles gjerne i nanosekunder. En kobberledning har en typisk forplantningstid rundt $\frac{1}{15}\frac{\text{ns}}{\text{cm}}$. En enkeltsående IC-chip med XOR ved $V_{DD} = 5\,\text{V}$ kan ha $t_{PLH} = 140\,\text{ns}$. Merk at porter i logiske kretser oppererer rundt 1–4ns. Se seksjon 15.

16.3 Stigetid

Stigetid (Rise time) er hvor lang tid det tar fra signalet er gått fra gyldig low-verdi til gyldig high-verdi. Verdien benevnes med t_{TLH}. I forelesning har vi opperert med at 10% er terskelen for lav, og 90% er terskelen for høy. Se figur 9.

16.4 Falltid

Falltid (Fall time) er hvor land tid det motsatte tar, altså høy til lav. Benevnes t_{THL}. Se figur 9.

16.5 Kritisk sti

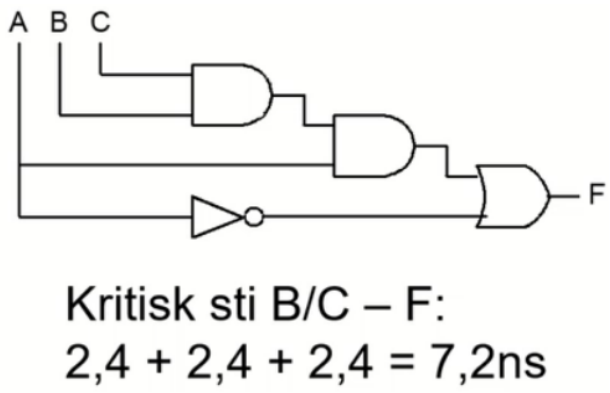
Når vi skal finne forplantningstid i en krets sammensatt av flere komponenter, må vi først finne en kritisk sti, og to input-states som, når de veksles mellom, forursaker at et signal må propageres gjennom hele den kritiske stien før output endrer seg. Stien er valgt slik at tiden det tar før signalet når output er størst mulig. For å finne forplantningstiden til den kritiske stien kan vi måle input og output, og ta tiden fra input endres til output endres. Husk at forplantningsstid måles fra signalet passerer 50% på input til det passerer 50% av sluttoutput. Ofte vil å endre kun én input-bit gi kritisk sti, så fremt de andre er satt riktig.

For logiske kretser med porter har vi fått oppgitt en tabell med *gate delay* for hver port. Se seksjon 15. Da slipper vi å forholde oss til t_{PLH} og t_{PHL}, og kan bare addere sammen forsinkelsene til hvert komponent på den tregeste veien fra en input til en output. Se figur 10. Gjennom boolsk algebra er det kanskje mulig å forbedre tiden til kritisk sti. Se seksjon 23 om Teknologimapping.

Husk at datablader for chipper gjerne opererer både med Typ, Min og Max-verdier for forplantningstid. Variasjon i tid kan skyldes flere ting, slik som temperatur og kapasitans i komponenter koblet til output. Når vi ser på portkretser i dette faget har vi forenklet regningen, og forholder oss kun til ett tall for forplantningstid.

16.6 Maksimal frekvens

Gitt en krets med en kritisk sti på 24 ns kan man regne ut maksimal frekvens ved å se hvor mange signaler som kan ferdigpropageres gjennom kretsen per



Figur 10: Regning på kritisk sti i enkel logisk krets

sekund.

$$\frac{1\,\mathrm{s}}{24\,\mathrm{ns}} = 41.67\,\mathrm{MHz}$$

I klokkestyrte kretser vil resulatet av den kombinatoriske kretsen ofte mates inn i et komponent med en setuptid. Den må da plusses sammen med kritisk sti slik at den kombinatoriske kretsen er stabil i en periode før neste klokkeflanke. I tillegg kan man måtte legge til den største av stigetider eller falltid.

17 Binærtall

Binærtall er tall i 2-tallsystemet, der hvert siffer enten er 0 eller 1, og plassene har verdier 2^n istedenfor 10^n slik vi er vant til. Hvert siffer kalles en bit, og sifferet lengst til venstre kalles mest signifikante bit. Sifferet lengst til høyre kalles minst signifikante bit, og svarer til $2^0 = 1$.

17.1 Konvertere fra binærtall

For å gå fra totallsystemet til 10-tallssystemet er det letteste å plusse sammen polynomet. Eksempel:

$$\begin{aligned} 1100101_2 &= 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 64 + 32 + 4 + 1 \\ &= 101_{10} \end{aligned}$$

17.2 Konvertere til binærtall

For å konvertere fra 10-tallsystemet til binærtall kan man enten begynne fra MSB eller LSB. Vi ser på sistnevnte. LSB er 1 hvis of bare hvis tallet er et oddetall. Etter å ha funnet LSB heltallsdividerer vi på 2 og fortsetter på samme måte for å finne neste bit. Eksempel med 73_{10}:

$$
\begin{aligned}
73 \text{ er et oddetall} &\rightarrow 1\textit{(LSB)} \\
\left\lfloor \frac{73}{2} \right\rfloor = 36 \text{ er et partall} &\rightarrow 0 \\
\left\lfloor \frac{36}{2} \right\rfloor = 18 \text{ er et partall} &\rightarrow 0 \\
\left\lfloor \frac{18}{2} \right\rfloor = 9 \text{ er et oddetall} &\rightarrow 1 \\
\left\lfloor \frac{9}{2} \right\rfloor = 4 \text{ er et partall} &\rightarrow 0 \\
\left\lfloor \frac{4}{2} \right\rfloor = 2 \text{ er et partall} &\rightarrow 0 \\
\left\lfloor \frac{2}{2} \right\rfloor = 1 \text{ er et oddetall} &\rightarrow 1\textit{(MSB)}
\end{aligned}
$$

17.3 Desimaltall og andre tallsystemer

TODO

17.4 Graykoding

Graykoding, også kjent som "reflected binary", er en binær koding av tall der alle nabotall har nøyaktig én bit forskjell. Man kan omdanne binærtall til og fra Graykoding. Huskeregel: $G = B \oplus (B >> 1)$.

$$
\begin{aligned}
G_7 &= B_7 \\
G_6 &= B_7 \oplus B_6 \\
&\vdots \\
G_0 &= B_1 \oplus B_0
\end{aligned}
$$

(a) Graykoding fra binær

$$
\begin{aligned}
B_7 &= G_7 \\
B_6 &= B_7 \oplus G_6 \\
&\vdots \\
B_0 &= B_1 \oplus G_0
\end{aligned}
$$

(b) Binær fra graykoding

Figur 11: Omgjøring av 8-bit til og fra graykoding

18 Negative binærtall

Med binærtall kan vi representere tall som strenger av 0 og 1, som er veldig praktisk i digitale kretser. I denne seksjonen skal vi vise hvordan også negative tall kan representeres.

18.1 Signed magnitude

Dersom vi i det daglige vil representere negative tall, pleier vi å legge på et $-$-tegn foran tallet. For at dette skal passe i en digital krets, kan vi si at MSB bestemmer fortegnet til tallet, og resten av bit-ene fungerer som normalt. Dette kalles signed magnitude. Konvensjonen er at 1 er **-** og 0 er $+$.

Eksempler på 8-bits signed magnitude

Signed magnitude	Tallverdi
00000101	5
10000101	-5
01111111	127
11111111	-127
00000000	0
10000000	-0

8-bits signed magnitude kan altså representere tall i intervallet $[-127, 127]$. Merk at $+0$ og -0 har ulike representasjoner, selv om de egentlig er samme tall.

18.2 Toerkomplement

Toerkomplement er veldig likt som vanlig binærtall, bortsett fra at mest signifikante bit (MSB) ikke representerer 2^{n-1}, men -2^{n-1}, der n er antall bits i representasjonen. Dersom MSB er 0 vil det representerte tallet være akkurat det samme som i vanlig binærtall. Hvis MSB er 1 er tallet negativt. Eksempel:

$$\begin{aligned} 10010110 &= 1 \cdot -2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\ &= -128 + 16 + 4 + 2 \\ &= -106 \end{aligned}$$

Fordelen med toerkomplementsrepresentasjon er at mange operasjoner støtter representasjonen uten vidre. Vi vet fra før at en f.eks. 16-bits-representasjon er en ekvivalensklasse modulo 2^{16}, siden all eventuell overflow forsvinner fra representasjonen. Observer at

$$2^{15} \equiv -2^{15} \mod 2^{16}$$

Endringen vi har gjort til betydningen av MSB har altså ingenting å si modulo 2^{16}. Det betyr at addisjon mellom 5 og -2 gir oss 3, uten at addisjonskretsen trenger å vite om toerkomplement. Toerkomplementet er det man automatisk får hvis man subtraherer et tall fra 0 og ignorerer overflow.

Eksempler på 8-bits toerkomplementsrepresentasjon

Toerkomplement	Tallverdi
00000010	2
11111110	-2
11111111	-1
00010100	20
11101100	-20
00010101	21
11101011	-21
01111111	127
10000001	-127
10000000	-128

8-bits toerkomplement kan altså representere tall i invervallet $[-128, 127]$, som er én mer enn signed magnitude. Dette er fordi vi ikke lenger har ulike representasjoner for -0 og $+0$.

18.2.1 Toerkomplement-operasjonen

Egentlig er toerkomplement navnet på en operasjon. Operasjonen er definert slik at et binærtall på n bits og dets toerkomplement blir 2^n i sum. Merk at $2^n \equiv 0 \mod 2^n$, så vi kan også si at et tall og dets toerkomplement blir 0 i sum når overflow forkastes.

Denne operasjonen vil derfor i praksis være å omgjøre et tall på toerkomplementsform til motsatt fortegn. 4 blir -4 og omvendt. Denne operasjonen er ikke bare å flippe MSB, slik som i signed magnitude. Alle bits må flippes, og deretter må 1 legges til. Overflow ignoreres.

Merk. 10000000 (-128) blir til seg selv når man tar 8-bits toerkomplement. Dette er fordi $+128$ ikke kan representeres i 8-bits toerkomplementform.

En mer mennesketilpasset måte: Begynn med minst signifikante bit (LSB) og beveg deg oppover til du møter en bit som er 1. Alle påfølgende bits (opp til og med MSB) flippes. Eksempel: Hva er 11010100 i 10-tallsystemet?

$$
\begin{aligned}
& 11010100 \\
\Rightarrow\ & -00101100_2 && \text{Toerkomplementet} \\
\Rightarrow\ & -(32_{10} + 8_{10} + 4_{10}) \\
\Rightarrow\ & -44_{10}
\end{aligned}
$$

18.2.2 Endring av antall bits

Gitt et tall på n-bits toerkomplementsform, vil vi ofte kunne ønske samme tall representert med en annen mengde bits. For å øke antall bits med k kopierer vi MSB k ganger og legger dem på venstresiden.

$$
\begin{aligned}
1101 &\Rightarrow 11111101 \\
0110 &\Rightarrow 00000110
\end{aligned}
$$

For å senke antall bits fjerner vi *MSB*. Vi kan kun fjerne *MSB* så lenge den nye *MSB* er det samme som den vi fjernet. Hvis vi fjerner flere bits enn det har vi ikke nok bits til å representere tallet.

$$
\begin{aligned}
11110101 &\Rightarrow 10101 \\
00001101 &\Rightarrow 01101
\end{aligned}
$$

19 Regning med binærtall

Binærtall er kun en representasjonsform av tall, så $6+7$ er 13 uansett hvilket tallsystem man bruker. Likevel er det kjekt å kunne regne med tallene direkte i binærtall på toerkomplementsform, siden det er slik de gjerne representeres i kretser. I tillegg har vi en endelig mengde bits, så det er ikke alle tall som kan representeres. Hvis vi har for få bits blir kanskje $6 + 7 = -3$.

19.1 Absoluttverdi

Dersom *MSB* er 1, gjør toerkomplementoperasjonen. Se Seksjon 18.2.1.

19.2 Addisjon

Addisjon fungerer veldig likt som man ville gjort på papir i 10-tallssystemet. Vi skriver tallene under hverandre, og adderer bortover fra LSB til MSB. Hvis begge tallene er 1 får vi 1 i mente, og må ta den med vidre. Eksempel med 8-bits tall:

	11010100	-44
+	01110110	118
=	01001010	74

I dette tilfellet ble carry ut av MSB 1. Dette betyr ikke at vi overflowet. Addisjonen har overflowet hvis to positive tall ble negative i sum, eller to negative ble positive i sum. Summen av et positivt og negativt tall kan aldri overflowe.

19.3 Subtraksjon

Subtraksjon gjøres gjerne ved å ta toerkomplementet til tallet som skal subtraheres, og gjøre addisjon.

19.4 Multiplikasjon

Multiplikasjon kan gjøres ved å først ta absoluttverdien av faktorene, multiplisere, og til slutt sette riktig fortegn på resultatet. I dette faget lærer vi en metode som fungerer generelt både på positive og negative tall på toerkomplementsform. Vi bruker $-14 \times (-5)$ som eksempel, og bruker 5-bits input.

$$10010 \times 11011$$

Vi kaller den første faktoren multiplikand, og den andre multiplikator. Når vi multipliserer et m-bits tall med et n-bits tall blir produktet et $m+n$-bits tall. Det første vi gjør er å utvide multiplikanden til like mange bits som resultatet, ved å kopiere MSB n ganger.

$$1111110010 \times 11011$$

Multiplikasjon er gjentatt addisjon. For å holde orden på resultatet underveis bruker vi en variabel vi kaller *partielt produkt*. Til å begynne med er den $m+n$ bits med 0.

$$0000000000$$

Deretter går vi gjennom hver bit i multiplikatoren, og for hver 1-bit adderer vi multiplikanden til partielt produkt. Vi skifter multiplikanden til venstre for hver bit vi går opp i multiplikatoren. Dette er fordi bit k i multiplikatoren representerer 2^k. MSB i multiplikatoren er spesiell i toerkomplementsrepresentasjon, så den representerer -2^4. Dette gjør vi i praksis ved å ta toerkomplementet til multiplikanden i siste rad.

1111110010×11011	$(-14) \times (-5)$
0000000000	partielt produkt
1111110010	multiplikand $\times\ 2^0 \times 1$
1111110010	partielt produkt
1111100100	multiplikand $\times\ 2^1 \times 1$
1111010110	partielt produkt
0000000000	multiplikand $\times\ 2^2 \times 0$
1111010110	partielt produkt
1110010000	multiplikand $\times\ 2^3 \times 1$
1101100110	partielt produkt
0011100000	multiplikand $\times\ -2^4 \times 1$
0001000110	70_{10}

Vi ignorerer all overflow på partielt produkt underveis, siden vi vet at svaret blir et 10-bits tall.

19.5 Divisjon

Gjøres som vanlig, bruk fantasien TODO.

20 Boolske funksjoner

Denne seksjonen inneholder måter å representere boolske funksjoner på, og algebra for å forenkle boolske uttrykk med flere variabler. Husk De Morgans teoremer og slikt fra seksjon 14.

20.1 Kanoniske former

Et boolsk uttrykk på kanonisk form består enten av OR av ANDer, eller som AND av ORer. Alle variabler må forekomme i hvert "ledd", enten som ikke-komplementær eller komplementær. Vi kan bruke boolsk algebra for å omdanne et vilkårlig uttrykk til kanonisk form:

$$\bar{x}\,\bar{y} + x\,y\,z = \bar{x}\,\bar{y}(z + \bar{z}) + x\,y\,z = \bar{x}\,\bar{y}\,z + \bar{x}\,\bar{y}\,\bar{z} + x\,y\,z$$

Vi har da gjort uttrykket om til OR av ledd der hvert ledd er ANDet og inneholder alle variabler. Vi kaller dette oppsettet SOP, *Sum of Products*. For en gitt boolsk funksjon finnes det nøyaktig én kanonisk SOP og én kanonisk POS, *Product of sum*. Disse leddene kalles henholdsvis mintermer og makstermer.

20.1.1 Minterm

AND av alle variabler på enten komplementær eller ikke-komplementær form. Indeksen til mintermen forteller hvilke variabler som er på ikke-komplementær form. MSB av indeks svarer til første variabel, og LSB til siste. Hvis bit-en er 0 blir variablen på komplementær form. Se tabell 2.

Minterm	Indeks	Uttrykk
m_0	$000_{(2)}$	$\bar{x}\,\bar{y}\,\bar{z}$
m_1	$001_{(2)}$	$\bar{x}\,\bar{y}\,z$
m_2	$010_{(2)}$	$\bar{x}\,y\,\bar{z}$
$\vdots$	$\vdots$	$\vdots$
m_7	$111_{(2)}$	$x\,y\,z$

Tabell 2: Eksempel med tre variabler x, y og z

Vi kan da omdanne en sannehetstabell til kanonisk form ved å se på hvilke rader som gir 1, og ORe sammen mintermene til de radene. Hvis ingen av

mintermene er perfekt tilfredsstilt, blir ingen ledd 1, og uttrykket blir 0. Mintermen m_6 er altså tilfredsstilt hvis og bare hvis $(x, y, z) = (1, 1, 0)$.

indeks	x	y	z	$F(x, y, z)$
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Tabell 3: Sannhetstabell for F(x,y,z)

Vi kan nå skrive F som summen av mintermene som representerer radene der $F = 1$

$$F = \bar{x}\,\bar{y}\,\bar{z} + \bar{x}\,\bar{y}\,z + x\,\bar{y}\,z + x\,y\,z = m_0 + m_1 + m_5 + m_7 = \Sigma(0, 1, 5, 7)$$

Σ betyr altså OR av mintermene med gitt indeks.

20.1.2 Maxterm

OR av alle variabler på enten ikke-komplementær eller komplementær form.

$$\begin{array}{lll} M_0 & 000_{(2)} & x + y + z \\ M_1 & 001_{(2)} & x + y + \bar{z} \\ M_2 & 010_{(2)} & x + \bar{y} + z \\ \vdots & \vdots & \vdots \\ M_7 & 111_{(2)} & \bar{x} + \bar{y} + \bar{z} \end{array}$$

Vi kan da omdanne en sannehetstabell til kanonisk form ved å se på hvilke rader som gir 0, og ANDe sammen maxtermene for de radene. For at en maxterm skal være 0 må den passe med raden på en prikk. Hvis noen av radene passer, blir produktet av makstermene 0, ellers 1.

$$F = (x + \bar{y} + z)(\bar{x} + y + \bar{z})(\bar{x} + \bar{y} + z) = \Pi(2, 5, 6)$$

Π betyr altså AND av maxtermene med gitt indeks.

20.1.3 Regler

Mintermen $m_3 = \bar{x}\,y\,z$ er kun 1 når $(x, y, z) = (0, 1, 1)$
Maxtermen $M_3 = x + \bar{y} + \bar{z}$ er kun 0 når $(x, y, z) = (0, 1, 1)$ Min tar laveste tall, er detfor AND. Minterm er derfor kun 1 under riktig omstendighet. Max tar høyeste tall, er derfor OR. Maxterm derfor kun 0 under riktig omstendighet.

Algebra med Σ og Π
For en funksjon $F(x, y, z)$ (8 forskjellige mintermer og maxtermer)

$$\begin{aligned}\Sigma(1,3,5) &= \overline{\Sigma(0,2,4,6,7)} \\ &= \Pi(0,2,4,6,7) \\ &= \overline{\Pi(1,3,5)}\end{aligned}$$

For eksempel på bruk av minterm, se seksjon **??** om å lage funskoner og krets for fulladderer.

20.2 Standardform

Standardform er ikke like streng som kanonisk form, og vi trenger ikke nevne alle variabler i hvert ledd. Vi må likevel forholde oss til sum av produkt (SOP) eller produkt av sum (POS).

$$F = x\,y + x\,\bar{y}\,z + \bar{x}\,y\,z$$

Hvert produkt her kalles en implikant.

20.2.1 Literalreduksjon

Vi kan redusere antall literaler ved å gjøre algebra.

$$\begin{aligned}F &= x\,y + x\,\bar{y}\,z + \bar{x}\,y\,z \\ &= x\,y\,z + x\,y\,\bar{z} + x\,\bar{y}\,z + \bar{x}\,y\,z \\ &= x\,y\,z + x\,y\,\bar{z} + x\,y\,z + x\,\bar{y}\,z + x\,y\,z + \bar{x}\,y\,z \\ &= x\,y(z + \bar{z}) + x(y + \bar{y})z + (x + \bar{x})yz \\ &= x\,y + x\,z + y\,z\end{aligned}$$

Vi har nå redusert antall literaler ned til en enklere SOP, men den kan reduseres ytterligere.

$$x\,y + x\,z + y\,z = x(y + z) + y\,z$$

Dette er derimot ikke lenger en SOP, men en Sum av Produkt av Sum. Det er derfor ingen grunn til å tro at kretsen blir noe raskere (snarere tvert imot), selv om man har redusert antall literaler.

20.2.2 Bytte mellom POS og SOP

Akkurat som med mintermer og maxtermer kan vi bytte mellom SOP og POS ved å ta inversen til uttrykket og bruke DeMorgans.

$$\begin{aligned} F &= x\,y + x\,\bar{y}\,z + \bar{x}\,y\,z \\ \bar{F} &= \overline{x\,y + x\,\bar{y}\,z + \bar{x}\,y\,z} \\ &= (\overline{x\,y})(\overline{x\,\bar{y}\,z})(\overline{\bar{x}\,y\,z}) \\ &= (\bar{x} + \bar{y})(\bar{x} + y + \bar{z})(x + \bar{y} + \bar{z}) \end{aligned}$$

Dette bruker vi i PAL. Se Seksjon 24.

20.3 Karnaughdiagram

TODO:Skriv

20.4 Tabellmetoden (Quine-McCluskey)

TODO:Skriv. Her er engelsk Wikipedia egentlig veldig god.

20.4.1 Elementære primledd

Dette er ledd som garantert burde være med i SOP av utrykket.

21 Datablader

Datablader forteller oss egenskapene til komponenter når de skal brukes i kretser. Tallene i databladene forutsetter visse omstendigheter, gjerne definert i toppen av tabellen. Dette kan være temperatur T_A, lastkapasitans C_L og gitt resistiv last R_L. Databladene forutsetter også at innsignaler innfrir noen oppgitte krav til stigetid og falltid (t_{TLH}, t_{THL}), og at spenninger er innenfor tilatte skranker.

21.1 Oppgitte verdier

Forplantingstid, **Stigetid**, og **Falltid**
Disse forsinkelsene kalles henholdsvis t_{PLH}/t_{PHL}, t_{TLH} og t_{THL}. Se seksjon 16 om forsinkelser.

Inputkapasitans
Input har en viss kapasitans C_{in}, altså må en viss mengde strøm tilføres for å endre spenningen opp til høy, eller motsatt for høy til lav.

Gyldige spenninger
V_{DD} er driftspenningen, og andre egenskaper påvirkes av driftsspenningen. V_{IL} er hvilken skranke en input-spenning må ligge i for å anerkjennes som lav. V_{IH} er skranken for høy input-spenning.

22 Logiske kretser

Nå skal vi bruke det vi vet om sannhetstabeller, algebra, porter og forsinkelse til å lage kretser som utfører boolske funskjoner raskt og med få transistorer.

22.1 1-bit fulladder

TODO: Skriv.

22.2 Ripple-carry fulladder

Én måte å addere sammen to n-bits tall er å bruke n 1-bits full-addere, med carry propagert fra LSB oppover til MSB. Svaret blir et n-bits tall med carry. Dette ligner måten addisjon gjøres for hånd. Problemet med denne adderen er at carry må propageres helt fra LSB til MSB, og kritisk sti blir veldig lang. (Hint: $0b11111111 + 1$)

22.3 Carry-lookahead adder

TODO: Skriv

22.4 Adder-subtractor

For å regne $x - y$ gjør vi istedet $x + (-y)$. Vi tar altså toskomplementet til y og gjør en vanlig addisjon. Vi kan lett lage en krets som støtter både

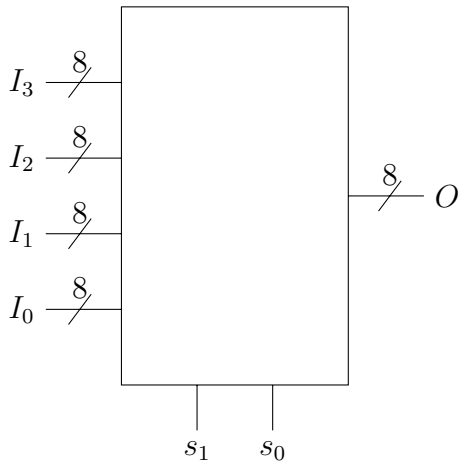
addisjon og subtraksjon, ved å utvide en adder med en ekstra kontrollinput SUB. Dette kalles en *Adder-subtractor*

Å ta toskomplementet er i praksis åinvertere alle bits i y og legge til 1. For å selektivt invertere XORer vi SUB med alle bits i y. For å legge til 1 trenger vi ikke en ekstra adder, men kan bruke carry-in i addisjonskretsen vi allerede har. Se Figur 12.

Figur 12: 16-bits Adder-subtractor

22.5 Multiplekser

En multiplekser er et kombinatorisk komponent som kan velge mellom flere inputsignaler å sende vidre som outputsignal. En kontrollinput brukes for å velge mellom inputene. Dataen kan være enkeltsignaler eller fler-bits data. For blokkdiagramform og sannhetstabell, se Figur 13 og Tabell 4.



s_1	s_0	O
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

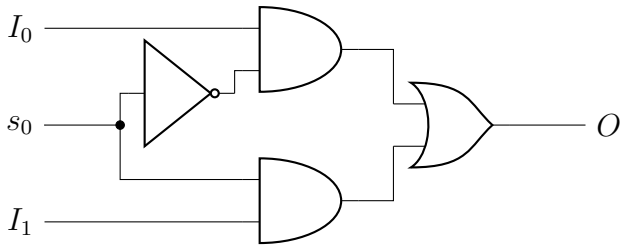
Tabell 4: Output fra 4-kanals multiplekser

Figur 13: 8-bits 4-kanal multiplekser

Vi kaller en 4-kanals multiplekser for en 4x1. Dersom vi vil lage en 8x1 kan vi koble output fra to 4x1-multipleksere sammen som hver sin input til en 2x1-multiplekser, og fordele s_0, s_1 og s_2. Slik kan vi velge mellom alle 8 inputs.

22.5.1 Logisk krets for 1-bits 2x1

For å gjøre det enkelt viser vi kun portdiagrammet for en enkel 2x1-multiplekser med enkle 1-bits inputs. Se Figur 14.



Figur 14: Portdiagram for 2x1-multiplekser

Vi bruker denne kretsen som eksempel på teknologimapping i Seksjon 23.

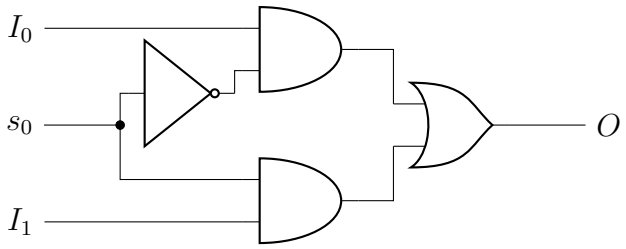
23 Teknologimapping

NAND er en universell port, som betyr at alle boolske funksjoner kan implementeres som kun NAND. Det samme gjelder NOR. I tillegg har NAND og NOR en forplantningstid på 1.4 ns, og bruker kun 4 transistorer hver. Teknologimapping handler om å modifisere en logisk krets slik at den kun bruker et sett tilgjengelige porter, og samtidig prøve å minimere kritisk sti og antall transistorer. Dette gjøres gjennom å dekomponere porter med flere innganger, og konvertering mellom porter med boolsk algebra. Siden AND er assosiativ kan en 3-inputs AND xyz omdannes til to 2-inputs AND $(xy)z$. Vidre kan vi bruke boolsk algebra for å omdanne porter.

$$
\begin{aligned}
x\,y &= \overline{\overline{x\,y}} && \text{AND = NOT NAND} \\
x + y &= \overline{\overline{x + y}} && \text{OR = NOT NOR} \\
x + y &= \overline{\bar{x}\,\bar{y}} && \text{OR = NAND av NOT} \\
x\,y &= \overline{\bar{x} + \bar{y}} && \text{AND = NOR av NOT} \\
\overline{\bar{x}} &= x && \text{NOT NOT kanselleres}
\end{aligned}
$$

23.1 2x1 med kun NAND og NOT

Vi henter frem den logiske kretsen for en 2x1-multiplekser fra Seksjon 22.5.1. Ingen porter har mer enn 2 inputs, så vi trenger ikke dekomponere.

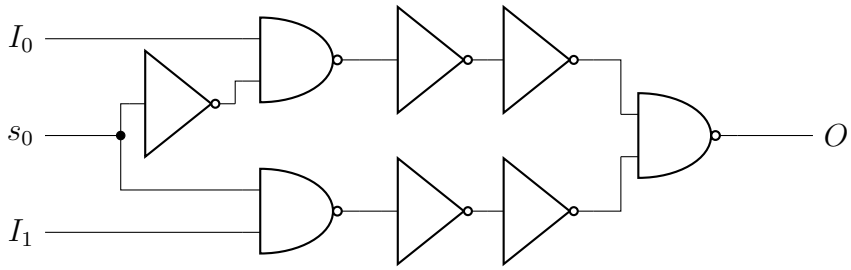


Vi bruker tabell 1 for å regne ut antall transistorer og propageringsforsinkelse gjennom kritisk sti. Før teknologimapping bruker vi $2 + 6 + 6 + 6 = 20$ transistorer og har en kritisk sti på $1 + 2, 4 + 2, 4 = 5.8\,\text{ns}$.

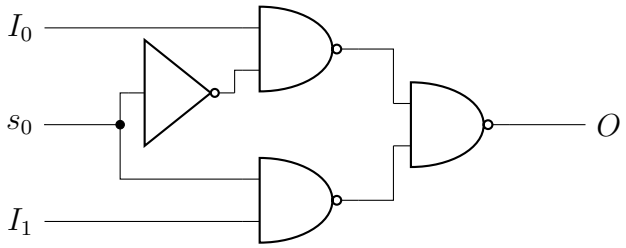
Vi starter med å omdanne AND til NOT NAND, og bruker DeMorgan på OR-porten.

$$x + y = \overline{\overline{x}\,\overline{y}}$$

Dette gir oss følgende krets:



Deretter fjerner vi doble invertere, siden $\overline{\overline{x}} = x$.



Den endelige kretsen har $2 + 4 + 4 + 4 = 14$ transistorer, og kritisk sti med $1 + 1, 4 + 1, 4 = 3.8\,\text{ns}$ propageringstid.

23.1.1 Input-delay

Dersom vi vet at en av inputene kommer fra en annen krets med en ekstra forsinkelse, kan vi legge opp kretsen vår på en annen måte, for å minimere

total kritisk sti.

23.2 Eksempel med kun NOR

Vi ønsker å lage en logisk krets for den boolske funksjonen

$$T = A\bar{C}\bar{D} + AB + B\bar{C}D$$

TODO

24 PAL

Gitt en boolsk funksjon på standardform, enten *Sum of product* eller *Product of sum*, kan vi med PAL lage en logisk krets for funksjonen. Som eksempel bruker vi funksjonene

$$\begin{aligned} F_1 &= x\bar{y} + y\bar{z}w + xy\bar{w} \\ F_2 &= (x + y + \bar{w})(y + \bar{z} + \bar{w})(x + y + w)(x + \bar{z}) \end{aligned}$$

Det første vi gjør er å forenkle F_2 med boolske regneregler.

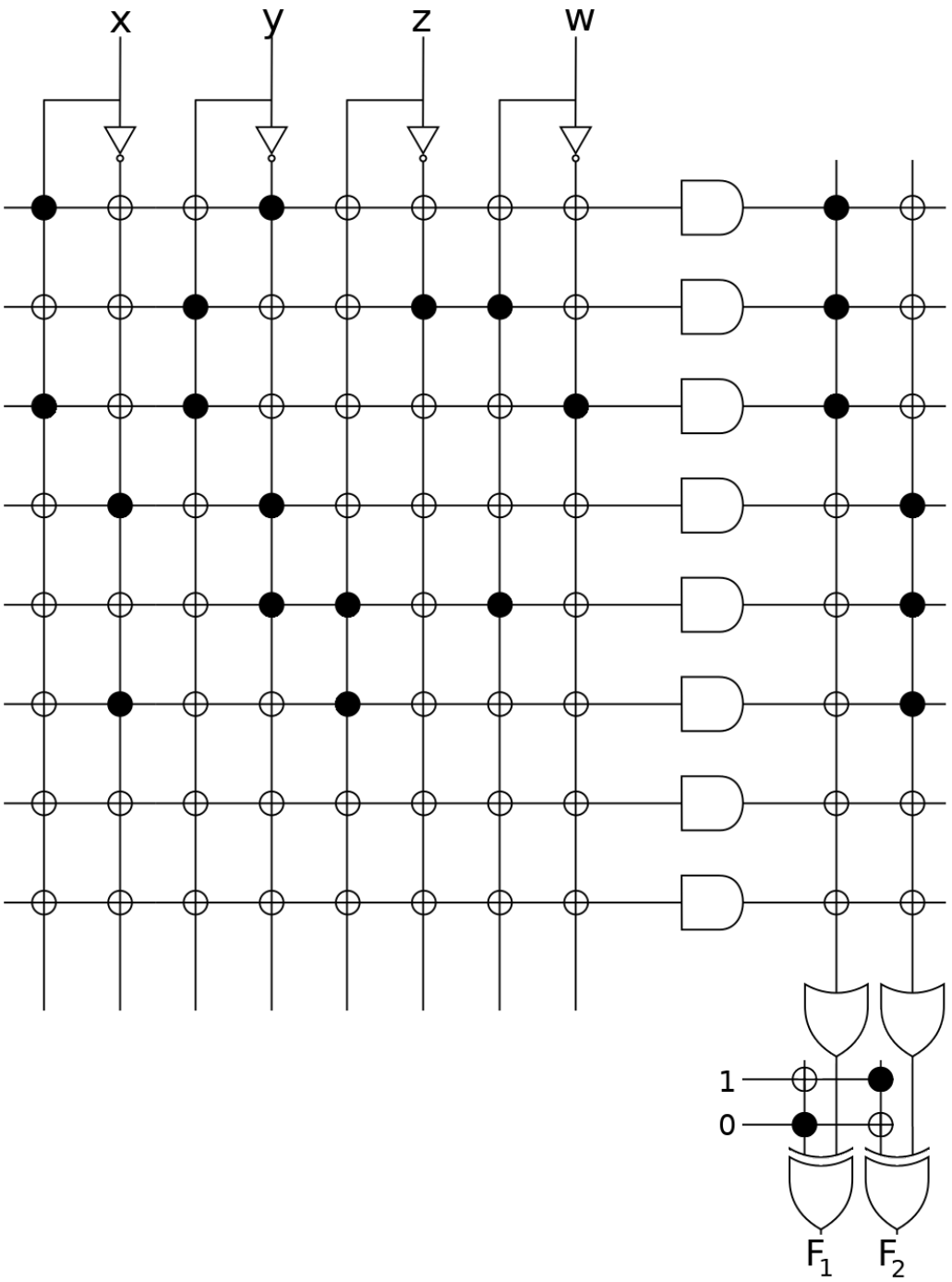
$$(x + y + \bar{w})(x + y + w) = (x + y)(\bar{w} + w) = (x + y)$$

Vi har dermed spart oss for et ledd i standardformen.

Det neste vi gjør er å gjøre begge om til *SOP*, ved å heller bry oss om $\overline{F_2}$. Vi bruker DeMorgan for alt den er verdt.

$$\begin{aligned} \overline{F_2} &= \overline{(x + y)(y + \bar{z} + \bar{w})(x + \bar{z})} \\ &= \overline{(x + y)} + \overline{(y + \bar{z} + \bar{w})} + \overline{(x + \bar{z})} \\ &= \bar{x}\bar{y} + \bar{y}zw + \bar{x}z \end{aligned}$$

Funksjonene kan nå implementeres på PAL. For hvert produkt lager vi en rad med variablene på komplementær eller ikke-komplementær form i AND-matrisen. Produktene for hver funksjon blir OR-et sammen i OR-matrisen. Til slutt bruker vi XOR-matrisen til å invertere F_2, siden *SOP*-uttrykket vårt er for $\overline{F_2}$. Se Figur 15.



Figur 15: PAL for F_1 og F_2

25 Sekvensielle kretser

Tidligere har vi forholdt oss til kombinatoriske kretser, der et sett med input passerer gjennom én eller flere boolske funksjoner og blir output. En sekvensiell krets har tilstand, som betyr at input til kretsen ikke bare kommer utenfra, men også fra inni kretsen selv. Sannhetstabeller inneholder derfor nåværende tilstand som input, og neste tilstand som output. Hvis man ikke er forsiktig kan dette gi ustabile løkker, f.eks. en inverter som mater inn i seg selv og flyter mellom 0 og 1.

25.1 Sekvensielle sannhetstabeller

Sannhetstabellen til en sekvensiell krets kan ofte forenkles slik at man ikke trenger å ha forrige output som input. Dette kan gjøres i tilfeller der neste tilstand er uavhengig av forrige tilstand, eller i tilfeller der man med ord kan beskrive hva som skjer med tilstanden. Se Tabell 5 og 6.

S	R	Q	$\bar{Q}$	Q_{next}	$\bar{Q}_{next}$
0	0	0	1	0	1
0	0	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0

Tabell 5: Utdrag av sannehetstabell for SR-lås

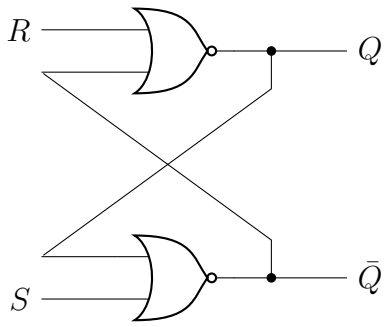
S	R	Q	$\bar{Q}$
0	0	Uendret	
1	0	1	0

Tabell 6: Forenklet versjon

Låser vs. vipper
Det later til å være variasjon i navnsetting av komponentene som følger. I dette dokumentet har jeg valgt å bruke *lås* om alt som er nivåstyrt (eller ikke har noe styring), og *vippe* om det som er flankestyrt. Dette er basert på diverse engelske definisjoner av henholdsvis *latch* og *flip-flop*.

25.2 SR-lås

En SR-lås er en sekvensiell krets bestående av to NOR-porter koblet i "kryss". I tillegg kommer input S og R som står for *Set* og *Reset*. Hver NOR-port har output ut av kretsen i tillegg til inn igjen i motsatt NOR. Disse blir SR-låsens outputs, Q og $\bar{Q}$. Se Figur 16.



Figur 16: Portdiagram for SR-lås

S	R	Q	$\bar{Q}$
0	0	Uendret	
0	1	0	1
1	0	1	0
1	1	0	0

Tabell 7: Sannhetstabell for SR-lås

SR-låsen fungerer ved at én av NOR-portene gir høy output, som får den andre til å ha lav output. Q og $\bar{Q}$ er altså motsatte under vanlig drift. Med S og R kan man styre hvilke NOR-porter som gjør hva. Når S blir satt høy vil NOR-porten til $\bar{Q}$ bli lav, som setter NOR-porten til Q høy, som igjen går inn i $\bar{Q}$ sin NOR-port og holder den lav. Motsatte skjer for R. Resultatet er at en høy input på *Set* setter Q høy helt til *Reset* blir satt høy, som resetter Q til lav. Se Tabell 7.

Merk: Før S eller R har vært høye er det umulig å vite tilstanden til bryteren. Dersom både S og R er høye samtidig blir både Q og $\bar{Q}$ 0, og hvis begge inputs går lave samtidig blir tilstanden igjen ubestemt.

25.3 Eksitasjonstabell

En eksitasjonstabell viser hvilke inputs som forursaker en ønsket tilstandforandring. Venstresiden har nåværende og ønsket tilstand, og høyresiden har hva input må være for å . Dersom en av inputenes tilstand ikke er vesentlig brukes X. For et eksempel med SR-lås, se Tabell 8.

Q	Q_{next}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Tabell 8: Eksitasjonstabell for SR-lås

Det er verdt å merke at styring ofte ignoreres i eksitasjonstabeller, siden et eventuelt *enable*-signal uansett må være høyt for at en tilstandsforanring skal

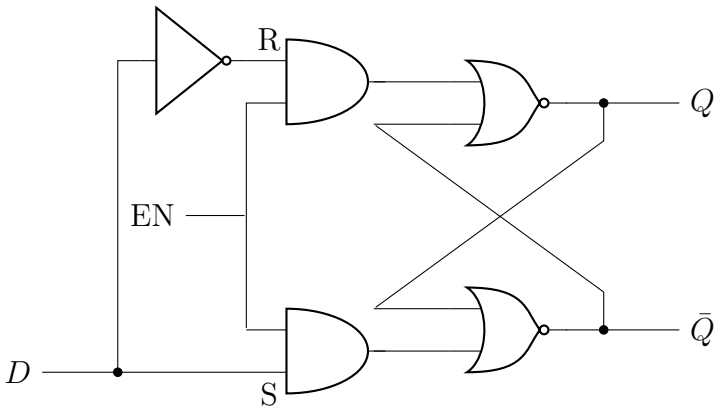
kunne skje. Dette betyr at eksitasjonstabellen for en SR-lås, en styrt SR-lås og en SR-vippe er identiske, selv om de har ulike tilleggskrav til forandringer.

25.4 Styrt SR-lås

En styrt SR-lås er en SR-lås der S og R kun har en virkning når en tredje input EN er høy. Dette ser man i blant annet en D-lås.

25.5 D-lås

En D-lås er en styrt SR-lås med kun én data-inngang D og enable-inngangen EN. SR-låsens input S kommer fra D, og R kommer fra $\bar{D}$. Se Figur 17.



Figur 17: Portdiagram for D-lås

Når EN er høy vil enten S eller R være høy, og SET/RESET-tilstanden blir lagret. Når EN blir lav vil tilstanden holdes, uavhengig av D. En D-lås kan ses på som én bit med hukommelse, som husker hva D var sist gang EN var høy. Se Tabell 9.

26 Tidsdiagrammer

TODO

27 Klokkestyring

Vi instroduserer et klokkesignal i kretsen for å kunne synkronisere dataflyt mellom flere komponenter. Som oftest ser klokkesignalet ut som en firkantpuls med 50% høyt og 50% lavt signal. Øyeblikket lav går til høy kalles stigende

D	EN	Q	Q_{next}
X	0	0	0
X	0	1	1
1	1	X	1
0	1	X	0

Tabell 9: Sannhetstabell for D-lås

Q	Q_{next}	D
0	0	0
0	1	1
1	0	0
1	1	1

Tabell 10: Eksitasjonstabell for D-lås

flanke, og motsatt kalles synkedne flanke. Frekvensen blir antall stigende flanker per sekund.

27.1 Klokkestyrt D-lås

En D-lås er allerede styrt med EN-inputen. For å gjøre den klokkestyrt kobler man klokken til EN. Dette blir da en nivåstyrt lås, som henter inn data når klokken er høy, og holder på dataen når klokken er lav.

27.2 Oppsett- og holdetid

t_{setup} er tiden datasignalet må være stabilt før en endring i klokkesignalet kan skje. t_{hold} er tiden datasignalet må være stabilt etter en endring i klokkesignal. Dette er fordi forsinkelser i porter gjør at signalforandringer bruker en viss tid på å propagere ferdig. Hvis et annet signal endres undeveis vil propageringen kunne skje kun delvis, og sluttilstanden blir udefinert. For nivåstyrte kretser gjelder t_{hold} fra nivået går ned. For flankestyrte kretser er t_{hold} tiden man må holde input etter trigger-flanken.

28 Flankestyring

Forrige seksjon brukte klokkesignalet for å bestemme når data kunne endres, og når data var låst fast. Dette kalles nivåstyring, siden nivåene på klokkesignalet har betydning. I denne seksjonen er det ikke nivået som er viktig, men øyeblikket nivået endres. Når klokke går fra lav til høy kaller vi det stigende klokkeflanke (⌊⌐), og motsatt høy til lav kalles synkende klokkeflanke (¬⌋). Vi opererer gjerne med at endringen skjer momentant, og at klokken er en perfekt firkantbølge. Et flankestyrt komponent bryr seg ofte kun om én av flankene.

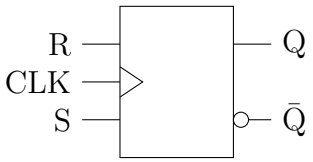
28.1 Triggring

Når et flankestyrt komponent mottar riktig flanke kaller vi det triggring. Da hentes data inn fra input, og behandles av kretsen. Verdien på input er kun interresant i trigger-øyeblikket, og frem til neste triggring vil kretsen være helt likegyldig til input. Så lenge input er stabil i t_{setup} før trigging, og t_{hold} etter, kan input være hva enn den bare vil resten av tiden og ikke ha noe å si.

I blokkdiagrammer har man et symbol for å vise at triggring skjer på stigende klokkeflanke, se Figur 18. Hvis triggring skjer på synkende flanke vil klokkeinputen ha en sirkel foran seg, likt en pMOS.

28.2 SR-vippe

Aller først gjør vi en SR-lås til en SR-vippe, bare for å vise blokkdiagramformen, sannhetstabellen, og timingdiagrammet (TODO). Se Figur 18 og Tabell 11.



Figur 18: Blokkdiagramform av SR-vippe

S	R	CLK	Q	$\bar{Q}$
0	0	⎍	Uendret	
1	0	⎍	1	0
0	1	⎍	0	1
1	1	⎍	0	0
X	X	X	Uendret	

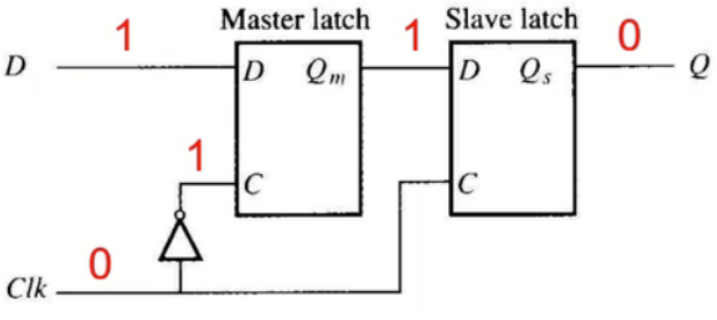
Tabell 11: Sannhetstabell for SR-vippe

28.3 D-vippe (Master-slave-vippe)

To D-vipper i serie, der den éne mater den andre, men med omvendt klokkesignal. Se figur 19. Dette gjøres slik at å lese input D og sende output Q er uavhengige. I vårt eksempel vil D leses inn så lenge klokken er lav, og i øyeblikket klokken går høy "trigger" vi vippen, og signalet sendes ut Q. Endringer på D vil ikke påvirke Q før neste triggring.

28.4 Shift-register

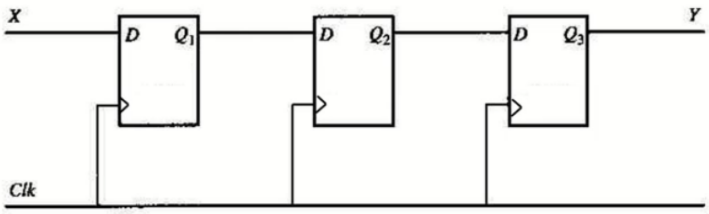
Ved å legge flere flankestyrte D-vipper i serie kan vi lage et shift-register der flere bits er lagret i en slags kø, og kan shiftes bortover samtidig. Se Figur 20.



Figur 19: D-vippe - to D-låser i master-slave-oppsett

D	CLK	Q	$\bar{Q}$
0	⎍	0	1
1	⎍	1	0
X	X	Uendret	

Tabell 12: Sannhetstabell for D-vippe



Figur 20: Et shiftregister laget av flere master-slave-vipper i serie med samme trigger

Hver av disse master-slave-vippene holder en verdi som de sender ut av hendholdsvis Q_1, Q_2, Q_3. Ved stigende klokkeflanke vil hver vippe overskrives av det den fikk inn i akkurat det øyeblikket. Resultatet er at køen går fremover. Verdien i Q_3 forsvinner, og de andre verdiene rykker én plass frem. Verdien X stiller seg bakerst i køa.

$$Q_3 \leftarrow Q_2 \qquad Q_2 \leftarrow Q_1 \qquad Q_1 \leftarrow X$$

28.5 JK-vippe

JK-vippen fikser problemet med SR-vippen. Den har to inputs J og K, men i motsetning til SR-vippen vil ikke tilstanden bli ubsestemt hvis begge inpu-

tene er høye samtidig. I stedet vil output-tilstanden Q flippe på hver triggerflanke der både J og K er høy.

28.6 T-vippe

En T-vippe har bare én input T, som når den er høy vil få output-tilstanden Q til å flippe på hver triggerflanke. Når T er lav vil vippen holde på output. Det er altså det samme som en JK-vippe, men der T går inn i både J og K. Siden tilstanden er ukjent når kretsen starter, har T-vippen en reset-input $RSTN$ som setter output lav. Merk at $RSTN$ kan være *aktiv lav*, altså at output tvinges lav når $RSTN$ er lav. Merk også at reset kan være *asynkron*, altså at T-vippen kan resettes også utenom triggerflanker.

28.7 Registere

Et register er flere d-vipper i parallell, med parallell input og output. Registeret brukes for å lagre hele tall, for eksempel i mellomregninger.

29 Databuss

I digitale kretser der flere komponenter som skal kunne sende data til hverandre er det upraktisk å ha koblinger mellom alle par. Derfor har man en felles databuss, som alle komponentene kan bruke til å dele data. Tradisjonelt er bussen mange parallelle ledninger. Buss-bredden er hvor mange bits som sendes i parallell. Kun étt komponent kan sende data ut på bussen samtidig, så alle komponentene har en kontroll-input som styrer skriving til bus. Dataen er trolig også kun ment for ett annen komponent, så komponentene har også et lesestyresignal. Alle komponentene er koblet på samme klokke, og på motsatte flanker skrives og leses data fra bussen.

I en datamaskin med en CPU er det vanlig med tre busser, og CPUen er "sjef" for alle. Adresse-bussen styrer hvilken addresse i minne CPUen sikter til. Data-bussen inneholder dataen som skal skrives til eller leses fra minne, og en kontroll-bus holder kontroll på hvilke komponenter som skal lese og skrive.

Det finnes også serielle buser, der bits sendes i serie, f.eks. USB, som står for Universal Serial Bus. Dette tillater faktisk raskere datastrømmer, siden parallelle buser kan få problemer med ulik forplantningstid i ledningene, pluss høyere induktans og kapasitans mellom ledningene.