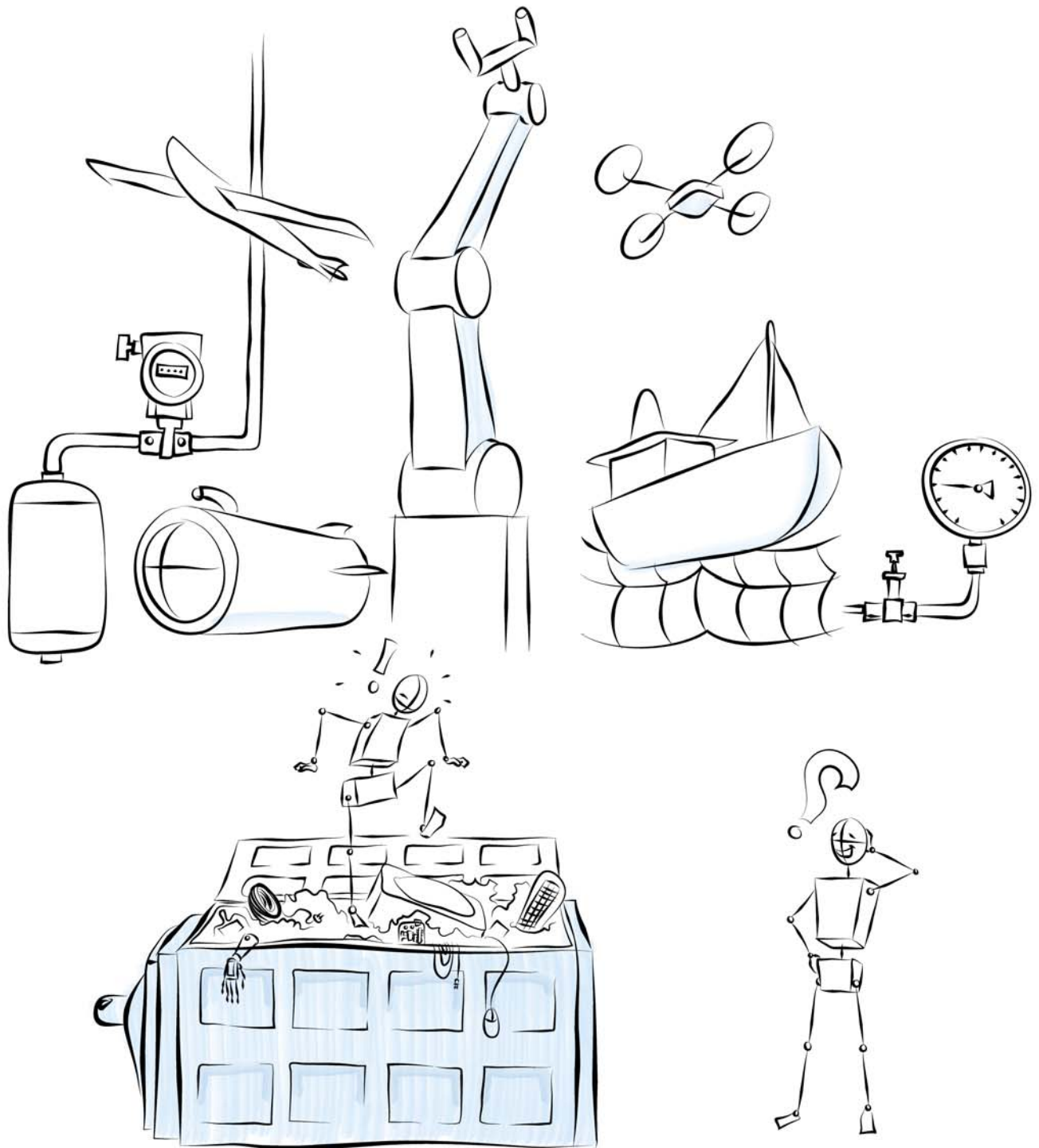


Kompendium for emne TTK4235 og TTK4101

# TILPASSEDE DATASYSTEMER

3. utgave



ØYVIND STAVDAHL M.FL.  
INSTITUTT FOR TEKNISK KYBERNETIKK, NTNU  
2017



# TILPASSEDE DATASYSTEMER

Øyvind Stavdahl m.fl.

---

© Øyvind Stavdahl, Trondheim 2015,  
2016, 2017, samt andre bidragsyttere (se  
forordet).

*Kopiering tillates bare i tilknytning til un-  
dervisning ved Institutt for teknisk kyber-  
netikk, NTNU, eller i henhold til skriftlig  
avtale med forfatteren.*

Forsideillustrasjon: Erlend Stavdahl

Øyvind Stavdahl  
Institutt for teknisk kybernetikk, NTNU  
7491 Trondheim

Tlf. 73 59 43 77  
E-post: ostavdahl@itk.ntnu.no

# Forord

## Forord til 1.utgave:

Dette kompendiet er utarbeidet for bruk i emnet TTK4235 Tilpassede datasystemer som undervises ved Institutt for teknisk kybernetikk, NTNU. Emnet inngår våren 2015 i første årskurs ved masterstudiet i Kybernetikk og robotikk. Kompendiet er ment å fylle igjen noen “hull” som ikke dekkes av den øvrige pensumlitteaturen i emnet, og er derfor ikke å betrakte som et komplett bokverk i seg selv.

Mye av stoffet bygger på tidligere kompendier som i noen tilfeller har blitt revidert mange ganger gjennom mange år. Det er derfor ikke i alle tilfeller mulig å identifisere nøyaktig hvilke bidrag ulike forfattere har gitt, ei heller alle som har bidratt. Foruten undertegnede er de kjente bidragsyterne (kjente bidrag angitt i parentes):

Kjell E. Malvig (kap. 3, 4, 5 og 6),  
Odd Pettersen (kap. 1 i sin helhet, kap. 4),  
Jan T. Gravdahl (kap. 4),  
Irja Gravdahl (figur 4.1 og 4.12),  
Pål From (figur 4.11) og  
Arnfinn Aa. Eielsen (kap. 3).

Takk til Åsmund Stavdahl og årets studentkull for korrekturlesning, og Erlend Stavdahl for forsideillustrasjonen.

Trondheim, 2015  
Øyvind Stavdahl

## Forord til 2.utgave:

En håndfull trykkfeil er rettet, noen figurer og forklaringer er forbedret. Takk til årets studentkull for konstruktive innspill.

Trondheim, 2016  
Øyvind Stavdahl

## Forord til 3. utgave:

Noen flere trykkfeil er rettet, takk til Konstanze Kölle for grundig korrekturlesning. Forsiden er oppdatert for å gjenspeile at kompendiet i år også brukes i emnet TTK4101 Instrumentering og måleteknikk.

Trondheim, 2017  
Øyvind Stavdahl



---

# Innhold

<b>Forord</b>	<b>iii</b>
<b>Innhold</b>	<b>v</b>
<b>1 Logikkstyring</b>	<b>1</b>
1.1 Innledning .....	1
1.2 Grunnleggende om kombinatoriske funksjoner .....	3
1.2.1 Svitsjer og svitsj-teori .....	3
1.2.2 Kort om Boolsk algebra .....	3
1.2.3 Svitsj-funksjoner .....	5
1.3 Logikk-kretser .....	6
1.3.1 Krets-symboler .....	6
1.3.2 Elektroniske kretselementer .....	7
1.3.3 Intern realisering av elektroniske svitsj-kretser .....	8
1.4 Regneregler innen boolsk algebra .....	11
1.4.1 Postulater for boolsk algebra .....	11
1.4.2 Utlede lover eller teoremer .....	12
1.4.3 Konsekvenser av De Morgan's teorem .....	13
1.4.4 Høy og lav representasjon .....	14
1.5 Vippen eller "Flip-flop" .....	16
1.5.1 SR (SETT-RESETT)-vippen .....	16
1.5.2 D-vippen (lås) .....	19
1.5.3 Flanke-triggete flip-flop'er .....	19
1.6 Sekvensielle funksjoner .....	23
1.6.1 Tids-usikkerhet .....	23
1.6.2 Sekvenstyper .....	26
1.6.3 Generell formulering av sekvens-forløp .....	27
1.6.4 Analyse av en sekvensiell krets .....	30
1.7 Tilstandstabeller for sekvensstyring .....	33
1.7.1 Huffman-tabell .....	33
1.7.2 Beslutningstabell .....	38
1.8 Eksempel på sekvensstyre-system: Heis-styring .....	45
1.8.1 Heis-systemet .....	45
1.9 IEC-848: Grafisk notasjon for sekvensstyring .....	53
1.9.1 Trinn .....	53
1.9.2 Kommandoer og aksjoner .....	54
1.9.3 Detaljert beskrivelse av kommandoer .....	60

1.9.4	<i>Detaljert beskrivelse av transisjoner</i> .....	65
1.9.5	<i>Gjenbruk av sekvens</i> .....	67
1.9.6	<i>Flere nivåer</i> .....	68
1.9.7	<i>Eksempler</i> .....	68
1.10	Metoder og utstyr for realisering av logikkstyring.....	72
1.10.1	<i>Oversikt</i> .....	72
1.10.2	<i>Elektromekaniske releer</i> .....	74
1.10.3	<i>Parallelle og sekvensielle funksjoner</i> .....	79
1.10.4	<i>Standarden IEC 1131-3</i> .....	79
<b>2</b>	<b>Prosesorarkitektur</b> .....	<b>85</b>
2.1	Innledning.....	85
2.1.1	<i>Historisk perspektiv</i> .....	85
2.1.2	<i>Digitalteknikkens tre kompleksitetsnivåer</i> .....	86
2.2	Z80: En klassisk prosessorarkitektur.....	88
2.2.1	<i>Overordnet arkitekturbeskrivelse</i> .....	88
2.2.2	<i>Registrene</i> .....	89
2.2.3	<i>Aritmetisk-logisk enhet (ALU)</i> .....	90
2.2.4	<i>Styringsenheten (Control Section)</i> .....	90
2.2.5	<i>Eksempel på utføring av en instruksjon</i> .....	91
2.3	AVR: Et moderne eksempel.....	92
2.3.1	<i>Mikroprosessor vs. mikrokontroller</i> .....	92
<b>3</b>	<b>Transmisjonslinjer og bølgefenomener</b> .....	<b>95</b>
3.1	Innledning.....	95
3.1.1	<i>Terminologi og begrepsavklaring</i> .....	95
3.2	Telegraflikningene.....	96
3.2.1	<i>Likningene</i> .....	97
3.2.2	<i>Løsningen</i> .....	99
3.2.3	<i>Karakteristisk impedans</i> .....	100
3.2.4	<i>Implikasjonene</i> .....	101
3.3	Oppsummering og konklusjoner .....	103
3.4	Eksempler .....	104
3.5	Relasjonen linjelengde og signalfrekvens .....	106
3.6	Pulser .....	107
3.7	Terminering av linjer.....	112
<b>4</b>	<b>Signaler og signalomsetning</b> .....	<b>117</b>
4.1	Signalers rolle i reguleringssystemer.....	117
4.2	Nettverkstyper for datainnsamling .....	118
4.3	Analoge signaler .....	119
4.3.1	<i>Enpolede signaler</i> .....	120
4.3.2	<i>Topolede (differensielle) signaler</i> .....	120
4.3.3	<i>Analoge signalformater</i> .....	122
4.4	Digitalisering av analoge signaler .....	123
4.4.1	<i>Filter mot nedfolding og overspenning</i> .....	123
4.4.2	<i>Analog multiplekser</i> .....	123
4.4.3	<i>Tast-og-hold-krets</i> .....	125
4.4.4	<i>Flying Capacitor</i> .....	127
4.4.5	<i>Selvsjekk</i> .....	127



4.4.6	A/D-omsetterens arbeidsområde .....	127
4.4.7	Samplingsteoremet og nedfolding .....	128
4.4.8	Tasting gir tidsforsinkelse .....	130
4.4.9	A/D- og D/A-omsettere – sentrale begreper.....	130
4.5	Kretser for D/A-omsetting .....	131
4.5.1	Binærvektet stigenettverk .....	132
4.5.2	R–2R-stigenettverk .....	133
4.6	Kretser for A/D-omsetting .....	134
4.6.1	Tilbakekoplet A/D-omsetter (servoomsetter).....	134
4.6.2	Dobbel-rampe-omsetter.....	135
4.6.3	Parallellomsetter.....	138
4.7	Skalering og kalibrering.....	139
4.7.1	Skalering, lineære elementer.....	140
4.7.2	Skalering, ulineært måleelement .....	143
4.7.3	Skalering av ut-variable .....	144
<b>5</b>	<b>Modulasjon .....</b>	<b>147</b>
5.1	Innledning og definisjoner .....	147
5.2	Amplitudemodulasjon med undertrykt bærebølge.....	149
5.2.1	Modulasjon.....	149
5.2.2	Demodulasjon.....	150
5.3	Amplitudemodulasjon med bærebølge .....	153
5.3.1	Modulasjon.....	153
5.3.2	Demodulasjon.....	153
5.4	Frekvensmodulasjon .....	153
5.5	Pulsbreddemodulasjon .....	155
<b>6</b>	<b>Strømsløyfe .....</b>	<b>157</b>
6.1	Hvorfor konverterer vi signaler til 4-20 mA? .....	157
6.1.1	Robusthet mot støy.....	157
6.1.2	Signal og kraftforsyning på samme linje .....	159
6.2	Strømsløyfens oppbygning og virkemåte .....	159
6.2.1	Strømforsyningen.....	159
6.2.2	Sløyfens spenningsfall .....	160
6.2.3	Transmitterens ytelse.....	161
6.2.4	Ledningsmotstand.....	162
6.3	Regneeksempel .....	162



---

# KAPITTEL 1

## *Logikkstyring*

### 1.1 Innledning

Logikkstyring er den del av et regulerings- eller styringssystem som tar seg av logiske funksjoner, som for eksempel Av/På, sekvensiell oppstart og nedkjøring, betingelses-kontroll på videre progresjon i en trinnvis opp-starting (forrigling), osv.

Det er påstått fra industri-hold at rundt 80 % av et regulerings- og styringssystem er logikkstyring, mens bare 20 % er kontinuerlig eller analog regulering. Det finnes knapt noe styringssystem som *ikke* inneholder logikkstyring, mens det er svært mange systemer som bare inneholder det, dvs. de inneholder *ikke* kontinuerlig regulering.

Når vi i dag i utstrakt grad realiserer styrings- og reguleringssystemer ved hjelp av datamaskiner eller datateknisk baserte styringsenheter, ligger dette spesielt naturlig til rette for logikkstyring. For en digital datamaskin er logikkstyring mye mer naturlig enn kontinuerlig regulering. Kontinuerlig regulering må behandle systemet som samplet for å kunne realisere reguleringen på en datamaskin, mens logikkstyring benytter logiske funksjoner, som er direkte naturlig for datamaskinen.

Logiske funksjoner kan inndeles i to grupper:

- Kombinatoriske
- Sekvensielle

I det etterfølgende vil vi først se på hva som er særtrekkene for disse to grupper. Deretter vil vi gjennom fire kapitler gi en konsentrert oversikt over grunnlaget for kombinatoriske funksjoner, svitsj-teori og boolsk algebra samt kretser for realisering av kombinatoriske funksjoner og sekvenskretser. I resten av kompendiet blir kombinatoriske og sekvensielle funksjoner for en stor del behandlet sammen. Man kan nemlig vanskelig tenke seg den ene gruppen uten at den er sterkt knyttet til den andre, og kombinatoriske og sekvensielle funksjoner er gjerne innvevet i hverandre.

### Kombinatoriske funksjoner:

Når vi skal studere kombinatoriske funksjoner, er det enklest å ta utgangspunkt i elektriske brytere eller *svitsjer*.

Svitsjer og svitsjing opptrer i mange fysiske former: elektriske brytere og vendere, releer, signal eller ikke signal i en elektrisk krets, AV og PÅ, osv. Det dreier seg altså om noe som har én av to tilstander, og datateknikken er som kjent basert på kretser som opptrer på denne måten. I styringssystemer manifesterer dette “noe” seg som *variable* som kan ha én av to tilstander. Slike variable er altså *binære variable*, og i styringssystemer direkte realisert med elektronikk er de gjerne representert av et elektrisk signal som kan anta én av to distinkte spenningsnivåer.

I kombinatoriske funksjoner genereres en ut-variabel entydig av funksjonens øyeblikkelige inn-variable (fri variable). De fri variable er enten binære (boolske<sup>1</sup>) variable eller relasjoner med binært utkomme (verdi), som f.eks.  $x > y$ . Generelt er dette boolske *uttrykk* (eng. *expression*). Den genererte utvariabelen (avhengige variable) er boolsk, dvs. med verdi SANN eller FALSK, 1 eller 0, osv.

### Sekvensielle funksjoner:

Sekvensielle funksjoner bygger på kombinatoriske, men innfører noe om det som har foregått før, dvs. *historien*; det huskes hvilken *tilstand* systemet har vært i. Vi vil se mer grundig på dette i Kap. 1.6.

---

1. Hvorfor det kalles *boolsk* blir forklart i kapittel 1.2.

## 1.2 Grunnleggende om kombinatoriske funksjoner

### 1.2.1 Svitsjer og svitsj-teori

For å uttrykke relasjoner mellom størrelser som hver kan anta en av to verdier, må vi ha et sett av binære variable, et sett av svitsjefunksjoner av de binære variablene, og en algebra til behandling av disse funksjonene av binære variable. Dette er nært knyttet til **logikk**. I logikk har vi *påstander*, og en påstand er enten sann eller falsk og kan representeres ved en binær variabel. I logikken har vi en grunnleggende unær operasjon *negasjon*, to binære operasjoner *disjunksjon* (logisk addisjon) og *konjunksjon* (logisk multiplikasjon) og en algebra som kan formuleres helt basert på disse elementæroperasjonene.

I svitsj-teori har vi liknende fenomener: Vi har en unær operasjon komplementering svarende til negasjon, to binære operasjoner addisjon og multiplikasjon, gjerne kalt henholdsvis ELLER og OG og svarende til disjunksjon og konjunksjon, og vi har en teori som binder disse sammen, *boolsk algebra*, oppkalt etter George Boole som først formulerte denne i et banebrytende dokument publisert i 1854. Overensstemmelsen med logikk er grunnen til at vi ofte omtaler svitsjteori som svitsjlogikk, og elektroniske kretser som realiserer svitsjer som *logiske kretser*. Dette innebærer at svitsjkretser kan betraktes som realiseringer av logiske relasjoner, dvs. som anvendelse av logikk.

Fordi svært komplekse logiske kretser kan realiseres, og fordi dette inngår i en lang rekke forskjellige systemer i styrings- og reguleringsteknikk, telefoni, datateknikk osv., har svitsjteorien blitt utviklet til en uavhengig og vel etablert disiplin. Det er også verdt å merke seg at svitsjteorien også er nært beslektet med mengdelære. Operasjonene ELLER og OG svarer til *union* og *snitt* i mengdelæren.

### 1.2.2 Kort om Boolsk algebra

En **binær variabel**  $x$  er en variabel som kan anta én av to mulige verdier. Hva verdiene er, er egentlig av underordnet interesse når bare operasjonene på verdiene er definert. Således kan settet av de to verdiene eksempelvis være {NEI, JA}, {FALSK, SANN}, {FALSE, TRUE}, {0, 1}, {AV, PÅ},

{OFF, ON}, {LAV, HØY}, eller det generiske {a, b}. I det følgende vil vi bruke settet (mengden)  $V = \{0, 1\}$  hvis det ikke uttrykkelig er sagt noe annet.

For de to elementene i  $V$  definerer vi først to binære<sup>1</sup> operasjoner

1. Addisjon eller ELLER (OR), med symbol  $+$  eller  $\vee$  :

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 1$$

I tabellform:

	0	1
0	0	1
1	1	1

To elementer behandlet med en addisjonsoperator blir kalt en *boolsk sum*, eller ganske enkelt en *sum*, av elementene.

En elektronisk krets som utfører denne operasjonen kalles en *ELLER-krets* (eng. *OR-circuit*).

2. Multiplikasjon eller OG (AND), med symbol  $\bullet$  eller  $\wedge$  :

$$0 \bullet 0 = 0$$

$$0 \bullet 1 = 0$$

$$1 \bullet 0 = 0$$

$$1 \bullet 1 = 1$$

I tabellform:

	0	1
0	0	0
1	0	1

To elementer behandlet med en multiplikasjonsoperator blir kalt et *boolsk produkt*, eller ganske enkelt et *produkt*, av elementene.

En elektronisk krets som utfører denne operasjonen kalles en *OG-krets* (eng. *AND-circuit*).

---

1. *binær* i denne sammenheng betyr at operasjonen virker på *to objekter*

Den tredje viktige operasjonen er *unær*, dvs. den opererer på bare én variabel:

3. *Komplementering*, ofte også kalt *negasjon*:

$$\bar{0} = 1$$

$$\bar{1} = 0$$

En elektronisk krets som utfører denne operasjonen kalles en *inverter*.

### 1.2.3 Svitsj-funksjoner

I likhet med vanlig algebra, har vi også i boolsk algebra funksjoner av en eller flere variable. I boolsk algebra må alle variable som inngår i en funksjon være binære variable. I tråd med vår anvendelse kaller vi disse funksjonene svitsj-funksjoner.

En svitsjfunksjon kan formelt uttrykkes:

$$f = \text{func}(x_1, x_2, x_3, \dots)$$

hvor *func* er en nærmere definert funksjon, *f* er boolsk (dvs. ender opp som en verdi 0 eller 1), og alle argumentene *x* er boolske variabler, dvs. enten fri variable eller boolske funksjoner. Eksempel på en enkel funksjon:

$$f = a \cdot c + b \cdot d$$

Hvis f.eks.  $c = 1$  og  $d = 0$ , vil dette gi  $f = a$ .

Vi vil gi mange flere eksempler etter hvert, men først skal vi se litt på praktiske muligheter for realisering, og deretter på egenskaper og “regneregler” ved boolsk algebra.

Realisering skjer enten i programmer som kjører på datamaskin, eller direkte i fysisk utstyr, “hardware”, som ofte er elektroniske kretser. Siden elektroniske kretser kan realisere boolske funksjoner på en meget direkte måte, egner disse seg for å *beskrive* og *forklare* svitsjefunksjoner, selv om man til syvende og sist kanskje velger endelig realisering i datamaskinprogram.

## 1.3 Logikk-kretser

### 1.3.1 Krets-symboler

Kretselementer som realiserer elementære svitsjoperasjoner symboliseres:

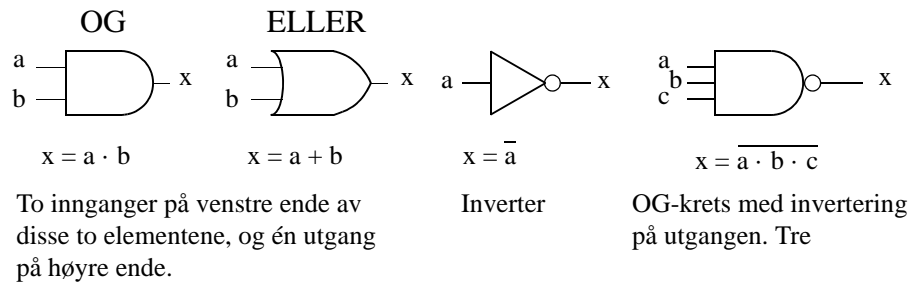


Fig. 1 Basis kretselementer

Basis-kretsene er OG, ELLER og inverter, og OG- og ELLER-kretsene har to eller flere innganger. Alle kretser kan kombineres med invertering, som symboliseres ved en ring. Invertert utgang er vist på kretselementet til høyre, men både innganger og utganger kan kombineres med invertering, og vi skal se at dette gir mulighet for svært fleksible løsninger.

Faktisk brukes oftere OG og ELLER-kretser *med* invertert utgang enn *uten*. På engelsk kalles disse henholdsvis NAND (Negated AND) og NOR (Negated OR). På norsk har vi ingen tilsvarende innarbeidete uttrykk<sup>1</sup>. De to vanligste grunnelementene er altså:

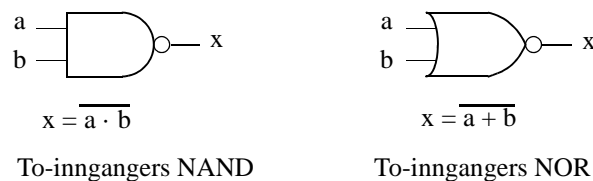


Fig. 2 Basis-elementene NAND og NOR

Som vi skal se senere, gir disse større fleksibilitet til kombinasjoner, og de er også litt enklere å realisere, dvs. de kan realiseres med litt færre elementære komponenter på den integrerte brikken.

1. Man kunne tenke seg "NOG" og "NELLER" eller "OG-IKK" og "ELLER-IKK", men slike uttrykk har ikke slått an.



### 1.3.2 Elektroniske kretselementer

Elektroniske kretselementer som realiserer svitsjfunksjoner er i dag vanligvis integrerte kretselementer, eller de inngår i integrerte kretser. De logiske variable er oftest elektriske likespenningssignaler med to distinkte spenningsnivåer til å representere **0** og **1**. I en vanlig tradisjonell klasse slike elementer er gjerne boolsk **0** representert ved  $u_L = 0$  volt (egentlig  $0\text{ V} < u_L < 0,4\text{ V}$ ) og boolsk **1** representert ved  $u_H = +5$  volt (egentlig  $2,4\text{ V} < u_H < V_{DD}$ , der  $V_{DD}$  er forsyningsspenningen, nominelt 5 volt).

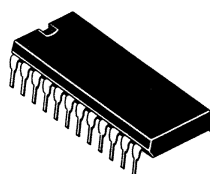


Fig. 3 Integrert krets, brikke (“chip”)

Vi kan også ha representasjon motsatt av det som ble forklart foran, dvs. boolsk 0 representert ved nominelt +5 volt og boolsk 1 representert ved 0 volt. I sistnevnte tilfelle snakker vi om “Lav” representasjon, og i det første, om “Høy” representasjon. Det er oftest “Høy” representasjon som brukes når kretser og logikkfunksjoner beskrives, og hvis intet annet sies, mener vi “høy” representasjon. Siden et signal (her boolsk) vanligvis varierer med tiden, dvs. det er en tidsfunksjon, avbilder vi et slikt tidsforløp gjerne slik:

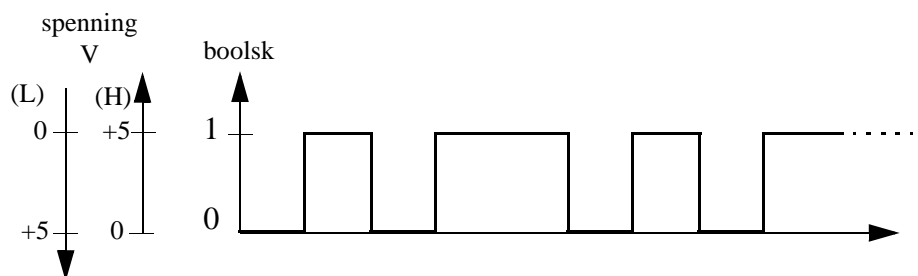


Fig. 4 Tidsdiagram for binær variabel

I stedet for Høy og Lav representasjon finner man i litteraturen også “positiv logikk” og “negativ -”, men dette er egnet til misforståelse, for i begge tilfeller er det vanligvis positiv spenning eller 0. I tilfellet “negativ logikk” er det spennings-svinget som går i negativ retning når signalet går fra 0 til 1 boolsk.

### 1.3.3 Intern realisering av elektroniske svitsj-kretser

Dette avsnittet er egentlig overflødig, for intet i det vi vil behandle i de etterfølgende kapitler avhenger av hva som her skal beskrives. Ikke desto mindre, så kan det hjelpe på forståelsen å vite hvordan de integrerte krets-elementene virker internt, derfor skal vi ta en kort gjennomgang.

Moderne svitsjkretser er oftest utført i teknologien CMOS (Complementary Metal-Oxide Semiconductor), og grunnkretsen er inverteren:

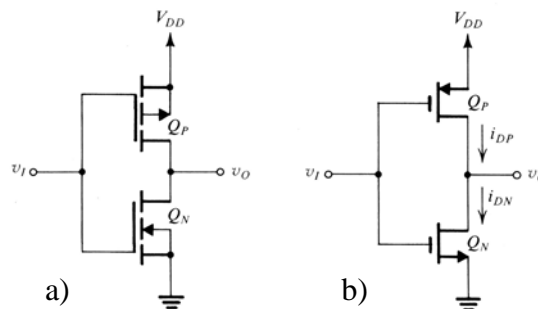


Fig. 5 Inverter realisert i CMOS: a) Detaljer b) Forenklet kretsskjema

Virkemåte for inverteren:

Inverteren består av to CMOS-transistorer, én P-kanal ( $Q_P$ ) og én N-kanal ( $Q_N$ ). Figur 6 viser inverteren med et ekvivalentskjema. Med “høy” spenning  $v_I$  på gate-inngangen blir  $Q_N$  ledende (har liten motstand source-drain), svarende til bryteren  $S_N$  lukket. Øverste transistor  $Q_P$  blir som en åpen bryter (høy motstand), dvs.  $S_P$  åpen. Dermed blir utgangen  $v_O$  lav. Motsatt med “lav” spenning på inngangen: Da blir  $Q_N$  (ekvivalent  $S_N$ ) åpen og  $Q_P$  ledende ( $S_P$  lukket), og  $v_O$  blir høy.

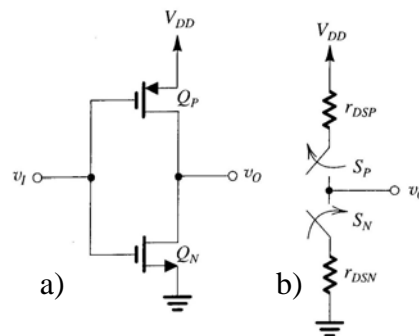


Fig. 6 CMOS-inverteren (a) og dens ekvivalent-skjema (b)

La oss nå utvide koplingen i Figur 6 med å dublere transistoren  $Q_N$  med en maken i parallell. Vi doublerer også transistoren  $Q_P$  med en maken, men her plassert i serie:

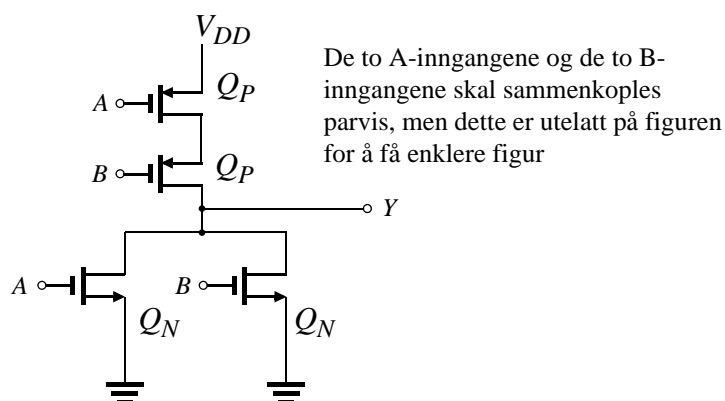


Fig. 7 To-inngangs NOR-port

På samme måte som i Figur 6 vil høy inngang gjøre  $Q_N$ -transistoren ledende, og dermed vil punktet  $Y$  trekkes *ned* hvis *enten*  $A$ - eller  $B$ -inngangen blir høy. Tilsvarende må *begge*  $Q_P$ -transistorene, som står i seriekopling, bringes ledende for at utgangen  $Y$  skal trekkes *opp*, dvs. at inngangene på begge  $Q_P$ -transistorene må være lav for at dette skal skje. Og motsatt, for at  $Y$  ikke skal trekkes opp, er det tilstrekkelig at én av  $Q_P$ -inngangene er høy. Når vi så danner par av én  $Q_P$ - og én  $Q_N$ -transistor ved å sammenkople deres innganger, vil det for at  $Y$  skal trekkes lav være tilstrekkelig at enten  $A$ - eller  $B$ -inngangene er høy. Vi har altså fått en krets som gir den boolske funksjonen

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

altså en NOR-krets, dvs. ELLER med invertert utgang.

Hvis vi trenger flere innganger, legger vi bare til flere  $Q_N$ - $Q_P$  transistorpar, en  $Q_N$  i parallell med de andre  $Q_N$ , og en  $Q_P$  innskutt i serie med de opprinnelige  $Q_P$ . Jeg sier “vi”, men det er riktignok fabrikanten av integrertkretsen som gjør dette, vi “brukere” kan bare velge mellom forskjellige kretser med gitte antall innganger.

Forresten kaller vi ofte disse kretsene *porter*, idet de virker som porter eller dører som åpnes og lukkes. “Gate” på engelsk.

En NAND-krets kan tilsvarende realiseres ved å bytte om parallell- og seriekoplingene, altså seriekople  $Q_N$ -transistorene og parallellkople  $Q_P$ -transistorene:

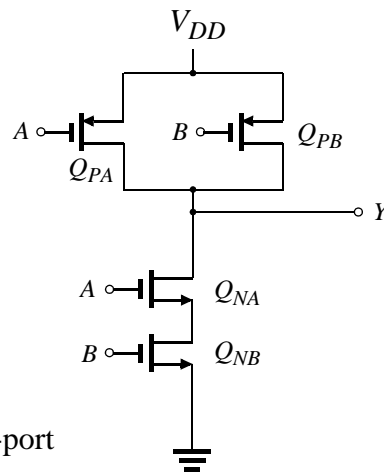


Fig. 8 To-inngangs NAND-port

Nå når vi har sett litt på hvordan elementære svitsjkretser eller porter virker og er oppbygget, er vi godt rustet til å bygge disse videre ut til å løse mer sammensatte boolske funksjoner. Da må vi først se litt mer på boolsk algebra, som vi så vidt introduserte i Kap. 1.2 side 3.

## 1.4 Regneregler innen boolsk algebra

### 1.4.1 Postulater for boolsk algebra

La oss ta for oss en mengde variable  $B = \{a, b, c, \dots\}$  med ekvivalens-relasjonen  $=$  og de to binære operasjonene  $+$  og  $\cdot$ , samt den unære komplement-operasjonen.

Boolsk algebra baserer seg på et sett postulater, og på grunnlag av disse kan vi deretter utlede en del matematiske lover. Vi kan lett konstatere at reglene i postulatene samsvarer med talleksemplene med de to mulige verdiene 0 og 1 vist i innledningen.

P1: Lukket system: Resultatet av en boolsk operasjon er en verdi innen det lukkede sett av verdier  $\{0, 1\}$ :

$$\text{For alle } a \in B, \quad a + 1 = 1, \quad a \cdot 0 = 0$$

P2: Assosiativ: Operasjonene  $+$  og  $\cdot$  er assosiative:

$$(a + b) + c = a + (b + c) = a + b + c$$

$$(a \cdot b) \cdot c = a \cdot (b \cdot c) = a \cdot b \cdot c$$

P3: Kommutativ: Operasjonene  $+$  og  $\cdot$  er kommutative:

$$a + b = b + a$$

$$a \cdot b = b \cdot a$$

P4: Distributiv: De to operasjonene er distributive overfor hverandre:

$$a + b \cdot c = (a + b) \cdot (a + c)$$

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

P5: Identitetselementer: Det finnes et *identitetselement* for operasjonen  $+$ , betegnet 0 og kalt *null*, og et annet for operasjonen  $\cdot$ , betegnet 1 og kalt *én* eller *enhet* (unity) innenfor mengden  $B$  slik at

$$a + 0 = a$$

$$a \cdot 1 = a$$

P6: Komplement: Hvert element i  $B$  har et *komplement* innenfor  $B$  slik at hvis  $\bar{a}$  er komplementet av  $a$ , vil:

$$a + \bar{a} = 1$$

$$a \cdot \bar{a} = 0$$

Bemerk at 0 og 1, som betegner de to identitetslementene i boolsk algebra, ikke må forveksles med tallene 0 og 1 i vanlig algebra. Samtidig gjør vi oppmerksom på at i boolsk algebra, likesom i vanlig algebra, utføres operasjonen  $\cdot$  før operasjonen  $+$ . Altså:

$$a + b \cdot c = a + (b \cdot c) \neq (a + b) \cdot c$$

Komplementoperasjonen innebærer altså, at hvis  $x = 0$ , så vil  $\bar{x} = 1$ , og hvis  $x = 1$ , vil  $\bar{x} = 0$ .

### 1.4.2 Utlede lover eller teoremer

Fra postulatene foran kan vi utvikle følgende lover:

L1: Likhet: For alle

$a, b, c \in B$ , hvis  $a + b = a + c$  og  $a \cdot b = a \cdot c$ ,  
da er  $b = c$ .

L2: Komplementær: For alle

$a, b \in B$ , hvis  $a + b = 1$  og  $a \cdot b = 0$ ,  
da er  $a = \bar{b}$  og  $b = \bar{a}$

L3: Identitetslementene 0 og 1 er komplement av hverandre:

På grunnlag av P4 har vi, siden 0 og 1 er  $\in B$ ,  
 $1 + 0 = 1$  og  $0 \cdot 1 = 0$ . Dermed gir L2:  
 $0 = \bar{1}$  og  $1 = \bar{0}$ .

L4: Identitetslementene 0 og 1 er unike:

Hvis dette ikke hadde vært tilfelle, ville det vært to eller flere 0'er, la oss anta de to  $0_1$  og  $0_2$ . Siden  $0_1 \in B$  og  $0_2$  er et identitetslement, vil P4 gi  $0_1 + 0_2 = 0_1$ . Samme resonnerment for  $0_2$  gir  $0_2 + 0_1 = 0_2$ , og dermed vil  
 $0_1 = 0_1 + 0_2 = 0_2 + 0_1 = 0_2$ .  
På samme måten kan lett vises at  
 $1_1 = 1_1 \cdot 1_2 = 1_2 \cdot 1_1 = 1_2$ .

L5: Komplementet er unikt:

Hvis dette ikke hadde vært tilfelle, ville det vært (minst)

to komplementer av en variabel  $a$ , la oss kalle dem  $a_1$  og  $a_2$ . P5 ville da gi  $a + a_1 = 1$  og  $a + a_2 = 1$ . Vi ville også ha  $a \cdot a_1 = 0$  og  $a \cdot a_2 = 0$ . Altså vil  $a + a_1 = a + a_2$  og  $a \cdot a_1 = a \cdot a_2$ . Dermed får vi, fra L2:  $a_1 = a_2$ . Bemerk: L5 kunne vært utviklet fra dette.

- L6: Involusjon: For alle  $a \in B$  er  $\overline{\overline{a}} = a$ .  
 Dette lar seg lett vise, f.eks. ut fra P5 og L2.
- L7: Idempotent: Et boolsk element er *idempotent* overfor de binære operasjonene  $+$  og  $\cdot$ , dvs. en operasjon utført gjentatte ganger på samme variabel endrer ikke variabelens verdi.  
 D.v.s.  $x + x = x$ ,  $x \cdot x = x$ . Hvis  $x = 0$ , så vil  $x + x = 0 + 0 = 0$ ,  $x \cdot x = 0 \cdot 0 = 0$ , og hvis  $x = 1$ , vil  $x + x = 1 + 1 = 1$ ,  $x \cdot x = 1 \cdot 1 = 1$ .
- L8: Absorpsjon: For alle  $a, b \in B$ ,  $a + ab = a$  og  $a \cdot (a + b) = a$ .
- L9: DeMorgan's teorem: Dette er et viktig teorem for realisering av porter.  
 $\overline{(a + b)} = \bar{a} \cdot \bar{b}$  og  $\overline{(a \cdot b)} = \bar{a} + \bar{b}$ .

Disse teoremene kan se "tørre og kjedelige" ut, men de danner et uunnværlig fundament for videreutvikling og sammenstilling av elementære logikkretser. De forteller oss blant annet at om vi utfører en rekke operasjoner etter hverandre på boolske variable, holder resultatet seg alltid innenfor det lukkede settet  $\{0, 1\}$ . Hvis vi komplementerer en variabel  $a$  flere ganger etter hverandre, veksler resultatet bare mellom  $a$  og  $\bar{a}$ . Altså  $\overline{\bar{a}} = a$ .

### 1.4.3 Konsekvenser av De Morgan's teorem

En svært nyttig lov er De Morgan's teorem. Under gjennomgangen foran om hvordan elementære kretselementer er realisert, ble det nevnt at porter med invertert utgang er vanligst. Dette skal vi nå utdype i sammenheng med De Morgan's teorem. NAND-kretsen realiserer  $\overline{(a \cdot b)}$ , og De Morgan's teorem sier oss at dette er det samme som  $\bar{a} + \bar{b}$ . Altså kan den samme elektronikk-kretsen som realiserer NAND også betraktes som en OR med in-

verterte innganger. Likeså vil NOR også være det samme som AND med inverterte innganger:

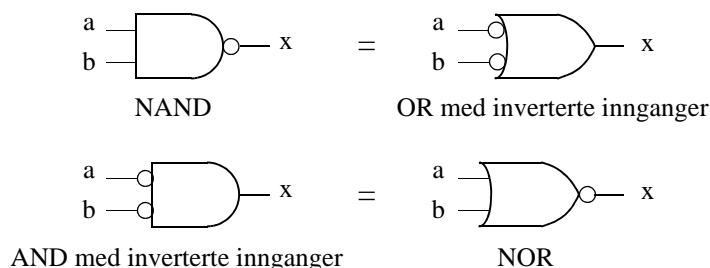


Fig. 9 To sider av samme sak v.hj.a. De Morgan

#### 1.4.4 Høy og lav representasjon

På side 7 ble forklart hva vi mener med “høy representasjon” og “lav representasjon” og forskjellen mellom disse. Et “høyt” signal (nominelt 5 V) uttrykker boolsk 1 i høy representasjon. Men vi ser nå at høy signalverdi like gjerne kan oppfattes som 0 i lav representasjon. Vi kan gjerne blande disse representasjonene hvis vi markerer representasjonen: 1 (H) = 0 (L) og 0 (H) = 1 (L). Dette er det samme som når vi, med høy representasjon, skriver  $1 = \bar{0}$ , og  $0 = \bar{1}$ . Altså er  $a(L) = \bar{a}(H)$  og  $\bar{a}(L) = a(H)$ .

Fordelen som oppnås med dette er at vi i mange tilfeller kan beholde en mer “naturlig” oppfatning av et signal slik det passerer gjennom en rekke trinn av kretser. Hvis et navngitt signal (la oss kalle det  $a$ ) representerer en hendelse, av varighet mye kortere og sjeldnere enn “ikke hendelse”, vil  $a = 1$  uttrykke at hendelsen inntreffer, representert ved en kortvarig positiv puls på en signalledning. Når dette signalet passerer gjennom en NAND-port, med konstant 1 (H) på portens andre inngang, kommer det på utgangen et signal som i hviletilstand ligger høyt, men når hendelsen inntreffer kommer det en puls som svinger negativt, kortvarig til 0 V. Det er da mer naturlig å se dette som bare et skifte av representasjon, ikke en negasjon. Det er fremdeles samme aktive hendelse, det er unaturlig å være tvunget til å betrakte høy signalverdi som aktiv eller “SANN” og et “ikke” eller FALSK som hendelsen.



Med denne betraktningsmåte får vi en notasjon som hjelper oss med å skille mellom logisk verdi og elektrisk representasjon. En logisk variabel har da en øyeblikksverdi  $a$  eller  $\bar{a}$ , uavhengig av om den på et visst sted i kretsskjemaet er i høy eller lav representasjon, (H) eller (L). I kretsskjemaet lar vi en ring symbolisere både boolsk invertering og skifte av elektrisk representasjon.

Eksempel: En kretskombinasjon vi ofte har bruk for er multiplekseren. Dette er en krets hvor vi ved hjelp av et boolsk styresignal velger ut ett av to signaler. Dette svarer til en elektrisk vender med to stillinger, hvor et boolsk styresignal bestemmer venderstillingen.

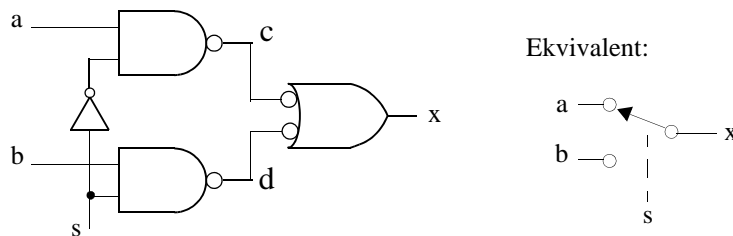


Fig. 10 To-stillings multiplekser

En morsom detalj er at denne koplingen, som benytter fire elementære kretselementer: to NAND, én OR med inverterte innganger og én inverter, kan realiseres med bare én integrert krets, “chip”, som inneholder 4 identiske elementer, i katalogen oppført som fire NAND. En av disse benyttes som inverter ved å sammenkople de to inngangene, og en av NAND-kretsene benyttes som utgangens OR med innganger i lav representasjon.

$$c(L) = a \cdot \bar{s} ; d(L) = b \cdot s ; x(H) = c(L) + d(L) = a \cdot \bar{s} + b \cdot s$$

Konsentrerer vi oss om logikken fremfor elektrisk representasjon, kan vi droppe (L) og (H) og får ganske enkelt

$$c = a \cdot \bar{s} ; d = b \cdot s ; x = c + d = a \cdot \bar{s} + b \cdot s$$

Det er imidlertid viktig å bemerke at det aller meste av litteraturen *ikke* drar nytte av denne notasjon men blander representasjon og logisk tilstand ved å konsekvent betrakte alle signalledninger i høy representasjon. Med denne vanlige uttrykksmåte blir ovenstående likninger:

$$c = \overline{a \cdot \bar{s}}, \quad d = \overline{b \cdot s}, \quad x = \bar{c} + \bar{d} = a \cdot \bar{s} + b \cdot s$$

Begge uttrykksmåter er naturligvis *korrekte*, men jeg vil påstå at den første av disse former viser mest direkte hva vi ønsker å uttrykke.

## 1.5 Vipper eller “Flip-flop”

I de foregående kapitlene har vi behandlet kombinatorisk logikk. I neste kapittel skal vi ta for oss sekvensielle funksjoner, som vi så vidt nevnte helt i begynnelsen, på side 2. For å kunne ha noe konkret å knytte behandlingen av sekvensielle funksjoner til, vil vi i dette kapitlet gi en kort gjennomgang av elektroniske *komponenter* som trengs for å kunne realisere *historie* eller *tilstander*, som **sekvensielle funksjoner** baserer seg på.

“Historie” eller “tilstander” innebærer en form for hukommelse, og for å oppnå det må vi ha noen kretser som kan skiftes mellom forskjellige stabile tilstander. En type grunnelementer for dette er bistabile vipper, ofte kalt “flip-flops”. Det finnes også monostabile og astabile vipper.

Bistabile vipper har to tilstander, SATT og RESATT (eng. SET og RESET), og de kan forbli i disse tilstander ubegrenset i tid.

Vippen er også den elementære byggeblokken for tellere, registre og andre kretser for å styre sekvensiell logikk.

### 1.5.1 SR (SETT-RESETT)-vippen

En aktiv-HØY input SR-vippe kan enkelt realiseres ved å krysskople to NOR-kretser. En aktiv-LAV input SR-vippe realiseres tilsvarende ved å bruke to NAND-kretser:

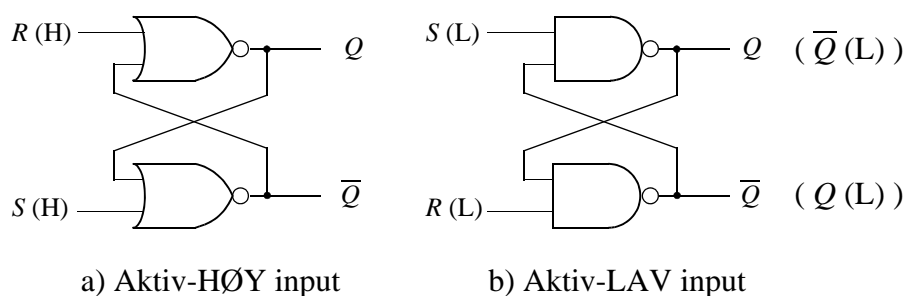


Fig. 11 SR-vippe, realisert med enkle portelementer

For Aktiv-HØY input-kretsen skal begge inngangene normalt (dvs. når de er passive) være LAV. Etter høy puls på S-inngangen vil utgangen  $Q = 1$ ,

uansett hva den var før S-(SETT-)pulsen. Dette innebærer tilstand 1. Tilsvarende vil en HØY puls på R-inngangen tvinge utgangen  $Q$  til LAV, dvs. tilstand 0. Utgangen  $\bar{Q}$  vil alltid være komplementet av  $Q$  når begge inngangene er passive, dvs. lave.

Aktiv-LAV input -kretsen virker tilsvarende, men her er passiv verdi for inngangene HØY, og kretsen settes, henholdsvis resettes, av en LAV inngangspuls.

I overensstemmelse med figur 9 kan kretsene i figur 11 tegnes slik:

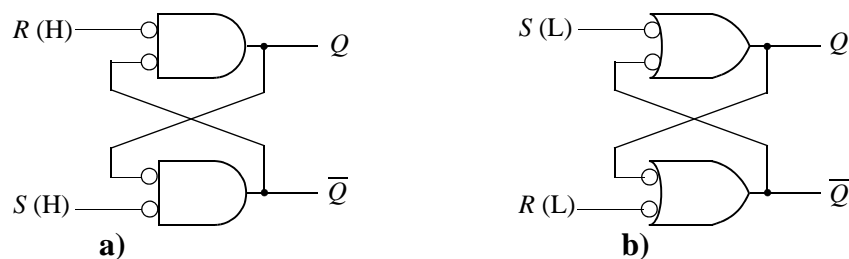


Fig. 12 Alternative tegnemåter for kretsene i figur 11

Vi kan noen ganger ha behov for å styre hvorvidt S eller R-signalene skal få påvirke vippen. Vippen kan da utstyres med OG-porter foran inngangene:

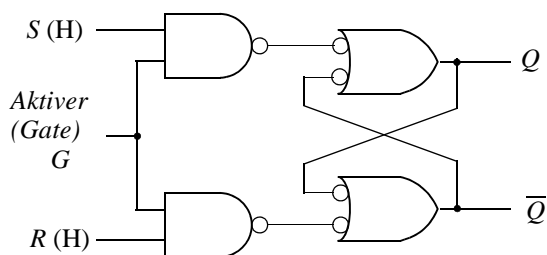


Fig. 13 SR-vippe med aktiveringsinngang

Signalet *Aktiver* må være 1 for at S eller R skal kunne ha noen virkning.

### Tilstandstabell

Det kan være illustrerende med en tabell over tilstandene til innganger og utganger for SR-vippen. Vi tar for oss vippen med Aktiv-LAV innganger:

Innganger		Utganger		Forklaring
S	R	Q	$\overline{Q}$	
H	H	IE	IE	Ingen Endring. Vippen forblir i eksisterende tilstand
L	H	1 (H)	0 (L)	Vippen blir SATT
H	L	0	1	Vippen blir RESATT
L	L	1	1	Ikke tillatt tilstand for inngangene

Hva betyr “Ikke tillatt”?

Hvis vi analyserer virkemåten for kretsen i figur 12-b og undersøker hva som hender hvis begge inngangene er LAV, ser vi at det skjer ikke noe spesielt dramatisk, men utgangene Q og  $\overline{Q}$  er begge HØY. De er altså ikke komplementære, som forutsetningen er. Når inngangene deretter begge går til HØY vil de to utgangene igjen være komplementære, men hvis inngangene skifter nøyaktig samtidig, vil det være tilfeldig hvilken tilstand Q havner i.

Vi vil litt senere i dette kapittelet se hvordan denne anomaliteten unngås.

### Logikksymbol

Vipper er brukt så ofte at vi vanligvis benytter et funksjonelt, dvs. mer overordnet, kretssymbol for dem, i stedet for de detaljerte kretsskjemaene foran:



Fig. 14 SR-vipper, funksjonelt symbol

Vi kan også se dette som et *logikk-symbol*, som fremhever logisk funksjon fremfor elektrisk virkemåte.

### 1.5.2 D-vippen (lås)

D-vippen kan betraktes som en videreutvikling fra SR-vippen med aktiveringsporter:

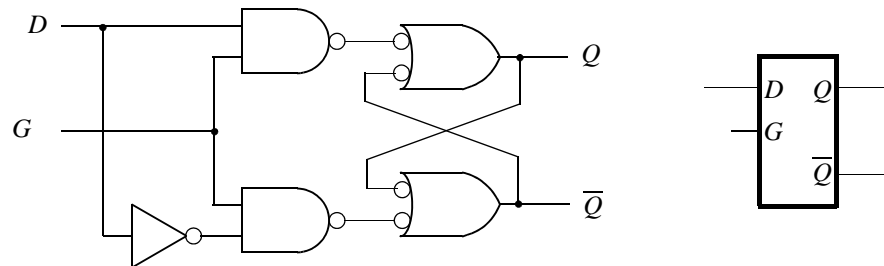


Fig. 15 D-vippe

D-vippen atskiller seg fra SR-vippen ved at den har bare én inngang utenom  $G$ . Denne inngangen kalles  $D$ , som står for Data. Figur 15 viser et logikkdiagram (logisk kretsdiagram) samt logikksymbol for D-vippen. Virkemåten er at når  $G$  kortvarig blir 1, vil verdien av  $D$  innsettes i vippen, dvs.  $Q = D$ . Når  $G$  går tilbake til 0, beholder  $Q$  sin verdi.

Hvis  $G$  blir stående =1 en stund mens  $D$  varierer, vil  $Q$  følge  $D$ 's variasjoner. Idet  $G$  går til 0, vil  $Q$  bli stående slik den var i dette øyeblikk. M.a.o.  $Q$  låses til verdien  $D$  hadde idet  $G$  gikk til 0, dvs. fra HØY til LAV. Av denne grunn kalles D-vippen også en **lås** (eng. latch). Endringer i  $D$  mens  $G=0$  påvirker ikke  $Q$ .

Med denne koplingen unngår vi anomaliteten beskrevet foran, dvs. en ikke-tillatt kombinasjon av inngangene. Siden de to datainngangene til NAND-portene foran vippen alltid er komplementære, vil NAND-portenes utganger aldri kunne være LAV samtidig.

### 1.5.3 Flanke-triggete flip-flop'er

En annen variant får vi hvis  $G$ -inngangen erstattes med en flanketrigget klokkepuls-inngang  $C$ , hvor vippens tilstand (utgang  $Q$ ) bare kan endres idet  $C$  skifter fra 0 til 1, eller fra 1 til 0. I førstnevnte tilfelle har vi en vippe som trigges på positiv flanke, i sistnevnte er det triggering på negativ flanke. En gitt flip-flop er enten positivflanke trigget eller negativflanke trigget.

Den vanligste bruken av disse kretsene er at C er tilknyttet en felles klokkepuls-generator. Dermed vil de forskjellige flipflop’ene skifte tilstand bare ved disse samme tidspunkt. Kretsene opererer altså *synkront*, og vi har å gjøre med *synkron logikk*.

Det finnes flere forskjellige typer flanketriggete flip-flop’er, men her skal vi bare beskrive de to viktigste: D-flipflop og JK-flipflop.

### Synkron D-vippe

Den synkrone D-vippen (også kalt *klokket D-vippe*) likner på D-låsen beskrevet i kapittel 1.5.2, med den forskjell at Q’s tilstand blir bestemt av D idet øyeblikk C skifter fra 0 til 1 (positiv-trigget flipflop) eller fra 1 til 0 (gjelder for negativ-triggete flipflop’er). Utenom dette bestemte øyeblikk, er tilstand Q upåvirket av D. Logikksymbolet for D vippen er:

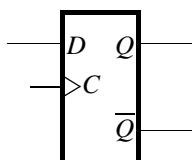


Fig. 16 Logikksymbol for synkron D-vippe

Tilstandstabell:

Innganger		Utganger		Forklaring
D	C	Q	$\bar{Q}$	
1	↑	1	0	SETT (lagrer en 1)
0	↑	0	1	RESETT (lagrer en 0)

### JK-vippe

JK-vippen er allsidig og mye brukt. I tillegg til klokkeinngangen har den to innganger J og K og likner på flanketrigget RS-vippe, men med den forskjell at J- og K-inngangene er fullstendig uavhengige av hverandre. JK-vippen har altså ingen “ikke-tillatt” kombinasjon av inngangenenes tilstander.

Logikksymbol:

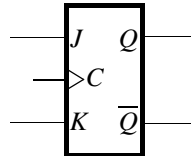


Fig. 17 Logikksymbol for synkron JK-vippe

Tilstandstabell:

Innganger			Utganger		Forklaring
J	K	C	Q	$\bar{Q}$	
0	0	$\uparrow$	$Q_0$	$\bar{Q}_0$	Ingen endring
0	1	$\uparrow$	0	1	RESETT (lagrer en 0)
1	0	$\uparrow$	1	0	SETT (lagrer en 1)
1	1	$\uparrow$	$\bar{Q}_0$	$Q_0$	Skift til motsatt (eng. toggle)

### Asynkron Preset og Clear (Sett og Resett)

I tillegg til de synkrone inngangene, har de synkrone D- og JK-vippene gjerne også to asynkrone innganger. Ved hjelp av disse kan man tvinge vippen til henholdsvis 1 eller 0, uavhengig av D eller JK eller klokke-innganger. Disse spesielle inngangene er vanligvis Aktiv-LAV, dvs. de skal normalt holdes i den inaktive stillingen HØY. Med en LAV puls på en av disse kan man altså tvangssette vippen til 1 eller 0. Det er konstruktørens plikt å påse at disse spesielle inngangene aldri kan bli LAV (dvs. *aktive*) samtidig.

Logikksymboler:

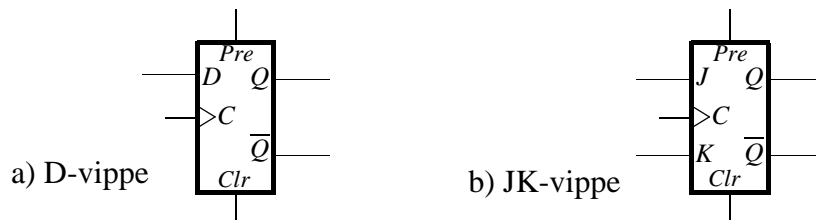


Fig. 18 Asynkrone Sett og Resett-innganger i synkrone vipper

### Eksempel med JK-vippe

En JK-vippe med negativ flanketrigging tilføres klokkepulser som vist i tidsdiagrammet nedenfor. Diagrammet viser også tilført signal til J- og K-inngangene, og vi kan se resultatet som forløpet av utsignalet Q.

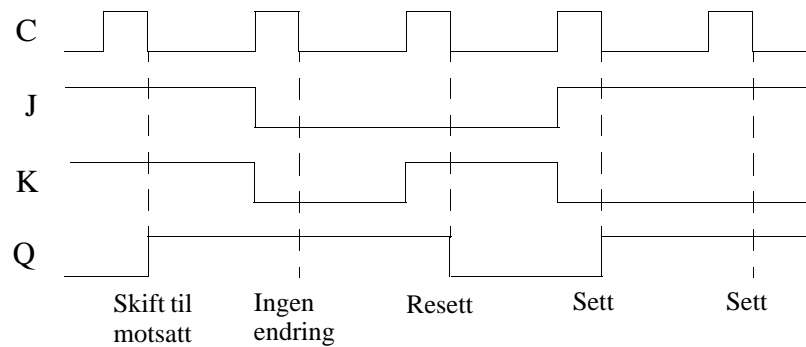


Fig. 19 JK-vippe: Signaler på innganger og derav utgangssignal Q



## 1.6 Sekvensielle funksjoner

### Sekvensielle funksjoner:

Sekvensielle funksjoner avhenger av kombinasjoner av inndata og historie. Historien uttrykker i hvilken grad og på hvilken måte nåværende funksjonsverdi påvirkes av hva som har foregått tidligere. Dette kommer til syne gjennom funksjonens øyeblikkelige *tilstand*, som igjen kan avhenge av *tidligere tilstand*. Hvis nåværende tilstand, sammen med kombinatoriske funksjoner, entydig uttrykker funksjonsverdien, og tilstanden er uavhengig av hvordan systemet kom i denne tilstanden, sier vi at systemet, i denne tilstanden, følger Markov-regler. Hvis dette gjelder alle tilstander, er systemet i helhet et Markov-system.

En systemmodell basert på tilstander illustreres gjerne i form av en *graf*, med distinkte noder som står for tilstandene. En tilstandstransisjon, dvs. overgang fra en tilstand til en annen, forårsakes av en *diskret hendelse* (eng. “discrete event”) og skjer momentant, ideelt betraktet. Eksempler på diskrete hendelser er diskrete forandringer av signalverdier, som forandring av binære verdier, at en kontinuerlig variabel passerer en gitt grense ( $x > x_0$ ), etc. Vi vil ta for oss dette, med prinsipiell angrepsmåte, i kapittel 1.6.3, men først vil vi se litt på et viktig fenomen ved praktiske koblinger.

### 1.6.1 Tids-usikkerhet

I avsnittet over ble *diskret hendelse* introdusert og forklart som momentan, ideelt sett. Praksis er imidlertid ikke ideell, og dette gjelder naturligvis også praktiske realiseringer av logikkretser. En diskret tilstandsændring i elektroniske logikkretser er et plutselig nivåskifte, vanligvis i elektrisk spenning, f.eks. fra 0V til nær 5 volt. Denne “plutselige” nivåendringen skjer ikke med uendelig hastighet, men i løpet av gjerne noen nanosekunder eller mer, altså med en skrå flanke, ikke rett firkant. Dessuten forplanter denne spenningsendringen seg videre i kretskoblingen, gjennom forskjellige kretselementer, og dette tar også litt tid. I løpet av et kort tidsintervall er altså mange av kretselementene i en overgangsfase, og først når det hele har kommet i ro, vil kretsen som helhet være i den nye tilstanden. Hvis man ikke tar spesielle forholdsregler, kan logiske kombinasjoner av forskjellige kretselementer gi helt gale “svar”, og disse kan også være upredikterbare. Et eksempel:

En binær tellerkrets kan lages slik at når vi teller oppover og går fra 011 til neste trinn, så skal dette bli 100. Hvert binært siffer (bit) realiseres med en JK-vippe:

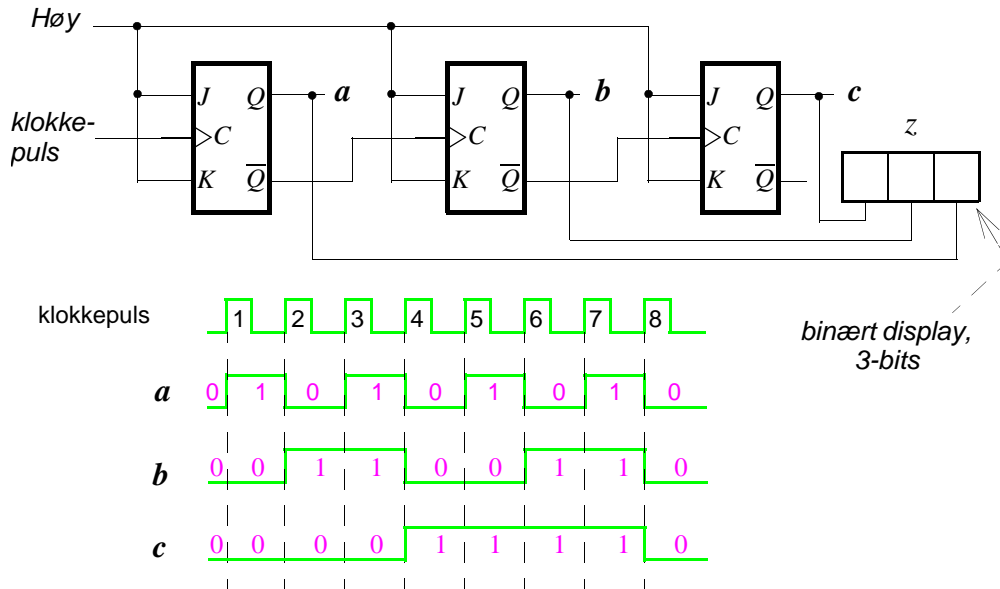


Fig. 20 3-bits asynkron teller og kappløp

Hvert trinn “klokkes” fra foregående trinn i denne koblingen. Den nevnte overgangen realiseres slik at når begge utgangene både  $a=1$  og  $b=1$  og vi får en ny tellepuls, vil  $a \rightarrow 0$  og  $b \rightarrow 0$  samtidig som det går en mente videre til neste vippe ( $c$ ).

Disse hendelsene skjer innenfor et kort øyeblikk i forkanten av puls nr. 4: Først går  $a \rightarrow 0$ , så  $b \rightarrow 0$  og deretter  $c \rightarrow 1$ . I løpet av dette korte overgangsintervallet har vi altså flere gale kombinasjoner, vist i rekkefølge ovenfra - ned:

$c$	$b$	$a$	$z$
0	1	1	3
0	1	0	2
0	0	0	0
1	0	0	4

overgangs-  
faser

Uttrykt som et trebits tall  $z = cba$ , så vil  $z$  i et kort øyeblikk gjennomløpe fra 3 til 2 og så til 0 før det ender på det riktige 4 (100 binært). Denne forplantningen gjennom kretselementer mens de stabiliserer seg kalles på engelsk “ripple” eller “race”, altså rippel eller kappløp. Asynkrontelleren kalles derfor også “rippel-teller”.

For å hindre at logikksystemet gjør gale ting under slike overgangssituasjoner, bruker vi gjerne **synkron** logikk, hvor en *klokkepuls* styrer slik at alle trinnene klokkes samtidig. I følgende kobling kan vi la  $C = Lav$  (dvs.  $C(L)=1$ ) være tidsintervallene når alle tilstandene er stabile og korrekte. Tellerkretsen skifter på positiv flanke av  $C$ , dvs. når  $C$  går fra L til H (logisk 1 til 0). Før de enkelte kretstrinnene vinner å stille seg om som følge av at  $C \rightarrow H$ , har korrektbetingelsen opphørt å være sann, dvs.  $C(L) = 0$ .

Vi kan ta utgangspunkt i den asynkrone telleren i Figur 20 og forandre koblingen litt og får en **synkron**-teller:

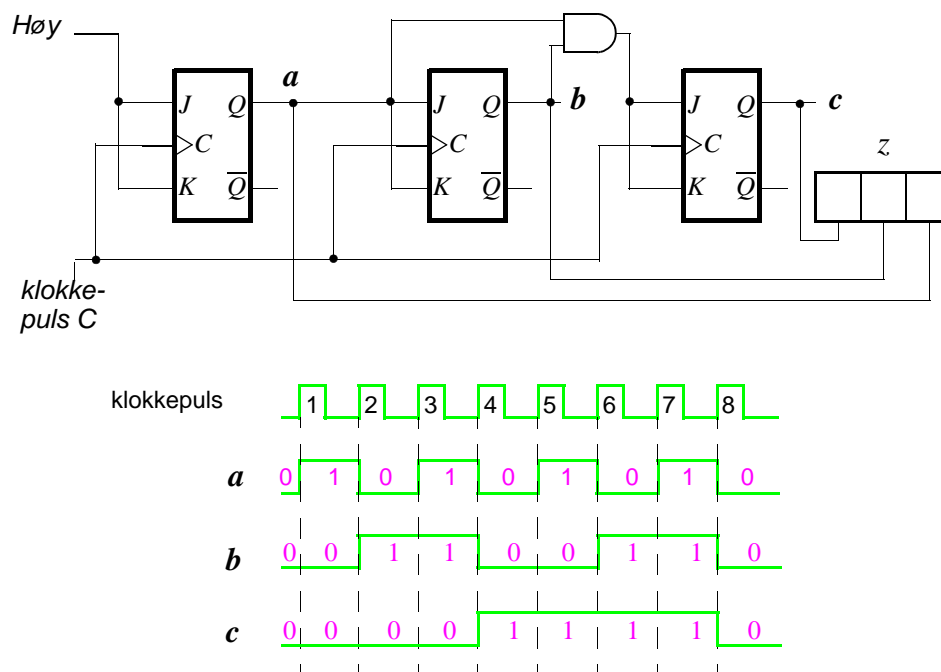


Fig. 21 3-bits synkron teller uten kappløp

Hver enkelt JK-krets stiller seg om samtidig, når  $C \rightarrow H$ , og de gjør dette i henhold til sine J-K -innganger i dette klokkeøyeblikket. Alle J og K er fast koblet til Høy, og dette innebærer at vippene stiller seg om til motsatt tilstand. Se forklaringen om JK-vipper side 20.

I den synkrone telleren skifter alle vippene “samtidig”, i motsetning til den asynkrone telleren, hvor vippe nr.  $n$  skifter som følge av at nr.  $(n-1)$  har skiftet. Likevel setter jeg ordet “samtidig” i anførselstegn, for her har vi å gjøre med en annen type kappløp, som faktisk kan være enda verre enn ved rippeltelleren: Et kappløp som skyldes små ulikheter fra én integrert krets til en annen, og som gjør at to vipper som skulle skifte samtidig endrer seg med kanskje noen få nanosekunders forskjell. Moderne elektroniske kretser er så raske at noen få nanosekunder kan ha betydning, og det farlige med et slikt kappløp er at det er upredikterbart. Denne usikkerheten unngås ved å sørge for at vi ikke aksjonerer i den nye tilstanden før et kort øyeblikk etter klokkepuls-flanken som trigget tilstandsovergangen. Det er en enkel måte å gjøre dette på: Hvis triggering skjer på stigende pulsflanke, lar vi fallende pulsflanke være det tidspunkt hvor vi foretar en (ny) aksjon som følge av den nye tilstanden.

Dermed kan vi vende tilbake til innledningen av dette kapittelet, Sekvensielle funksjoner på side 23: Siden tiden i et digitalt system er diskret og går fremover skrittvis med ett skritt pr. systemklokke-puls, vil **en diskret hendelse skje momentant, og vi trenger ikke ta forbeholdet "ideelt betraktet"**.

### 1.6.2 Sekvenstyper

Sekvensfunksjoner kan være *tidssekvenser*, *beregningsbetingede* eller *prosessbetingede* sekvenser og blandinger av disse. Ved prosessbetingede sekvenser er signaler fra prosessen bestemmende for overgang til en annen tilstand, og dette gjør denne typen *ikke-deterministisk*. Hvis sekvensen bare er bestemt av beregninger og av fastlagte tidspunkt, er sekvensen deterministisk.

Ved tidssekvenser skiftes fra en tilstand til en annen ved forutbestemte tidspunkt, ikke nødvendigvis ekvidistante. Mange sekvensstyringer følger et slikt prinsipp, i hvert fall delvis. Ved beregningsbetingede sekvenser vil beregningsresultater under veis være bestemmende for videre gang, dvs. vi-

dere sekvens av tilstander. Mange styringssystemer følger også delvis et slikt prinsipp. Begge disse typer kan imidlertid karakteriseres som *åpen sløyfe* -styring, siden det ikke skjer noen påvirkning av sekvensen ut fra hvordan prosessen oppfører seg. Styring etter blanding av disse to deterministiske prinsippene kan man betrakte som styring *etter oppskrift*: Oppskriften foreskriver for eksempel: Start motor, vent 2 sekunder, åpne ventil V, vent 30 sekunder mens en tank fylles, sett på oppvarming, varm i 15 minutter, åpne tømmeventil, steng etter 30 sek., osv.

En slik “blind” styring som antydnet over kan være enkel, men den har naturligvis en stor svakhet i at den ikke tar hensyn til hvordan *prosessen* utvikler seg. I de aller fleste tilfeller er derfor sekvensen i betydelig grad bestemt av *målte (innleste) prosess-signaler*, og dermed har vi en **prosess-betinget** sekvens. Som regel har vi altså en blanding av disse tre typer.

Ved blandet sekvenstype kan prosess-signaler bryte inn i en tids-sekvens og endre denne, eller forskjellige undersekvenser kan innkoples ved forskjellige tidspunkt som bestemt av signaler innlest utenfra.

### 1.6.3 Generell formulering av sekvens-forløp

Sekvensielt forløp beskrives generelt gjerne med henvisning til sekvensmodell formulert av Mealy eller Moore. Vi skal her se på Mealys formulering:

En sekvensiell prosess kan beskrives ved:

1. Et endelig sett  $Q$  interne tilstander.
2. Et endelig sett  $P$  innganger.
3. Et endelig sett  $W$  utganger (ut-signaler).
4. Tilstandsforandring  $\tau$  fra  $q_m$  til  $q_n$  hvor  $q \in Q$
5. Utgangstransformasjon  $\omega$  fra  $(q_m, p_i)$  til  $w$  hvor  $q \in Q$ ,  $p \in P$  og  $w \in W$

Begge modellene illustreres ved en rettet graf, med knutepunkter for

tilstandene og linjer med piler for transisjonene mellom tilstandene:

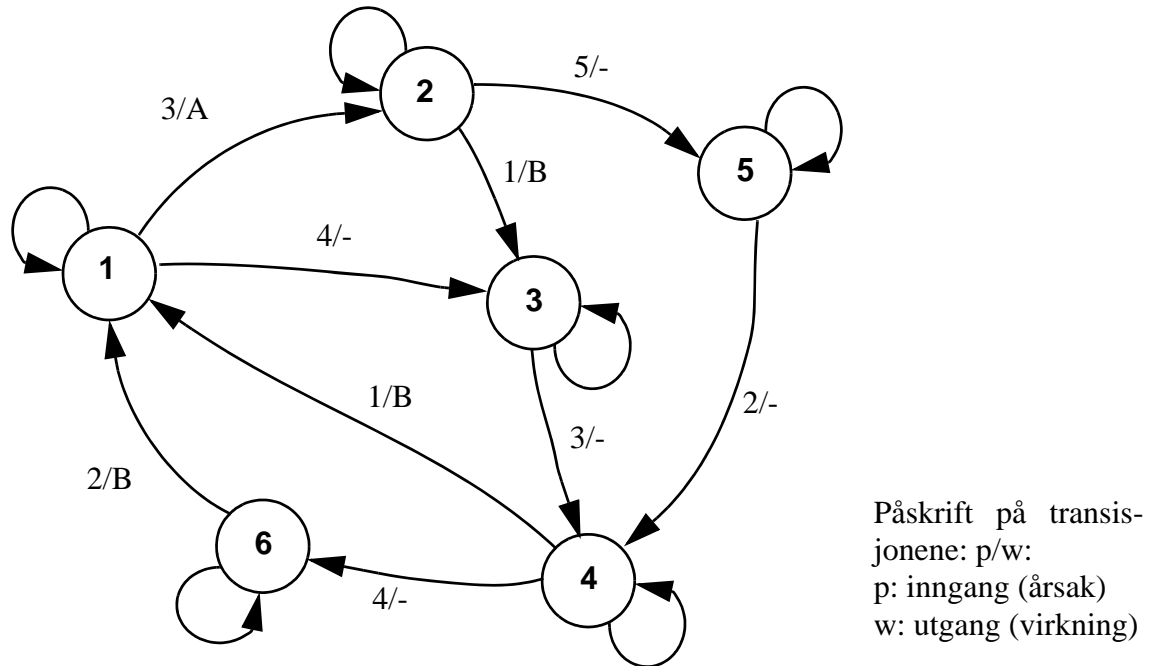


Fig. 22 . Tilstandsforandringer og tilhørende aksjoner

En sekvensiell prosess “er” i en og bare en tilstand til enhver tid. Om denne øyeblikkelige tilstanden kan vi si at den er *aktiv*. Egenskapen *aktiv* forflytter seg blant knutepunktene i den rettede grafen langs transisjonslinjene, det vil si prosessen går over fra en tilstand til en annen, som følge av “innganger” både i Moores og i Mealys modell. Slike “innganger” kan være hvilke som helst fri variable i et program, slik at logiske inn-signaler kan bearbeides numerisk og påvirke modellens “innganger”. Tilstandsoverganger ledsages av en **aksjon**, karakterisert ved dannelsen av “utganger” (Mealy). En “utgang” er generelt altså en aksjon, noe som har en effekt. I vår sammenheng kan dette typisk være generering av et fysisk utgangssignal.

For en gitt tilstand  $q_j \in Q$  og inngang  $p_i \in P$ , vil transformasjonen  $\tau$  definere neste tilstand:

$$q_s = \tau(q_j, p_i) \in Q$$

Transformasjonen  $\omega$  definerer utgangssignalene:

$$w = \omega(q_j, p_i) \in W$$

Tilstander og transformasjonene  $\tau$  og  $\omega$  kan illustreres med figur 22.

Det er altså knyttet en aksjon til *tilstandstransisjonen*, ikke til selve tilstanden. Ved Moores modell er derimot aksjonen knyttet til tilstanden, og det kan teoretisk vises at de to modellene er ekvivalente, dvs. et system fremstillet etter én av modell-typene kan transformeres til den andre modell-typen. Vi kunne derfor gjerne brukt Moores modell, men vi vil her til å begynne med bruke Mealys modell.

Et komplekst sekvenssystem må fremstilles hierarkisk, hvis vi skal kunne både behandle detaljer og samtidig beholde total-oversikten. Dette innebærer at *en enkelt aksjon* på høyt betraktningsnivå ofte må detaljeres ved å beskrives som et undersystem og fremstilles som en egen sekvensiell modell med tilhørende graf. Dette undersystemet trer i funksjon når og bare når aksjonen på nivået over utløses. I Moores modell er aksjonen knyttet til *tilstandene*, og dette betyr i denne sammenheng at den mer detaljerte undersekvensen aktiveres mens tilhørende tilstand på nivået over er *aktiv*.

### 1.6.4 Analyse av en sekvensiell krets

Vi tar for oss en kretskobling som ved første øyekast kan se enkel ut, men som inneholder noen interessante egenskaper:

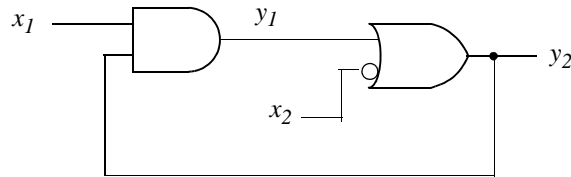


Fig. 23 Eksempel på sekvensiell krets

Når vi skal analysere en slik krets, kan vi starte med å sette opp en tabell over alle kombinasjoner av inngangsverdier ( $x$ ) og tilhørende avhengig variable ( $y$ ), slik at vi kan finne systemets tilstander og tilstandsoverganger. I analysen beveger vi oss stort sett nedover i tabellen, men det er også avmerket en overgang oppover, merket med stiplet pil:

$x_1$	$x_2$	$y_1$	$y_2$	Tilstand
0	0	0	1	A
1	0	1	1	B
0	0	0	1	A
1	0	1	1	B
1	1	1	1	B
0	1	0	0	C
1	1	0	0	C
0	1	0	0	C
0	0	0	1	A
0	1	0	0	C

Eksempel på transisjon uten tilstandsending

Eksempel på mulig race hvis begge inngangene  $x_1$  og  $x_2$  skifter samtidig.



Vi starter med en tilfeldig valgt kombinasjon av inngangene, her  $\langle x_1, x_2 \rangle = 0,0$ , finner verdiene av de avhengig variable, her  $\langle y_1, y_2 \rangle = 0,1$ , og setter et navn på denne tilstanden, her kalt A. Så endrer vi én av inngangene, her  $x_1 \rightarrow 1$ . Av figur 23 ser vi da at  $\langle y_1, y_2 \rangle = 1,1$ , altså har vi en ny tilstand, og vi kaller den B. Vi kan etter hvert tegne et tilstandsdiagram, liknende det som ble gjennomgått på mer prinsipielt vis i kapittel 1.6.3, en rettet graf med tilstandene som noder illustrert som sirkler, og tilstandsoverganger som transisjoner illustrert med piler mellom nodene. Ved siden av pilene kan vi påføre hvilke endringer i inngangene som forårsaker transisjonen. Innen i hver sirkel angir vi tilstandens navn, samt verdien av de avhengig variable, her  $y_1, y_2$ . Etter hvert oppstår diagrammet nedenfor:

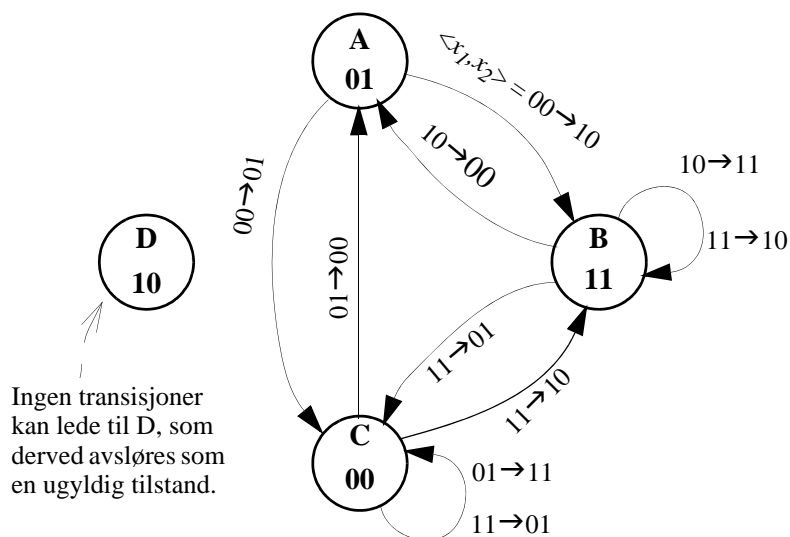


Fig. 24 Tilstandsdiagram, eksempel-krets

Vi har her definert tilstandene ut fra systemets utgangsvariable, og dette er jo naturlig fordi det er disse variablene som systemet genererer og som påvirker omgivelsene. Imidlertid kan vi under analysen få litt problemer fordi inn-variablenes nåtilstand kan være bestemmende for neste systemtilstand. M.a.o.: Noen av de tilstandene vi har definert kan ha indre tilstander. I vårt eksempel gjelder dette tilstandene B og C, så la oss trekke inn disse

interne tilstandene. Da blir diagrammet figur 24 videreutviklet:

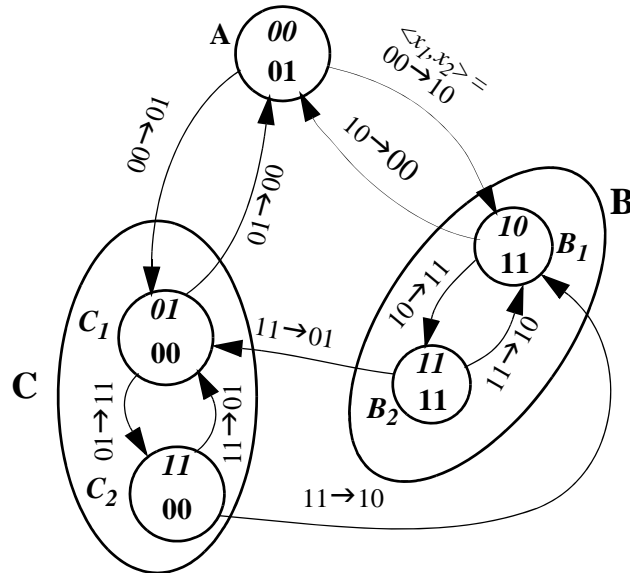


Fig. 25 Tilstandsdiagram med undertilstander

Her er de ovale tilstandsmarkeringene for B og C identiske med figur 24, men inne i både B og C har vi to interne tilstander, henholdsvis  $B_1$ ,  $B_2$  og  $C_1$ ,  $C_2$ . I både disse og i A har vi nå tilføyd inngangsverdiene (skrevet i kursiv for å gi forskjell fra utgangs-tilstandene). Transisjonen i figur 24 som fører tilbake til samme tilstand er nå inne i tilstandene B og C, som overganger mellom de *interne* tilstandene. Et naturlig spørsmål nå: Hvorfor beskrive dette i to trinn, som her først metoden i figur 24 og deretter figur 25? Svaret er at det er førstnevnte metode som er viktig, idet denne beskriver systemets egentlige tilstander. De interne transisjoner skyldes ofte *virkningsløse endringer av inngangsvariable*. Metoden med de "interne" tilstandene er bare noe som i visse tilfeller kan være nyttig for å hjelpe oss med å sikre at vi har dekket alle mulige transisjoner: Vi har her to ( $n$ ) uavhengig variable, og i prinsippet skal det da kunne gå  $2^n = 4$  transisjoner ut fra hver node. Hvis bare én av dem endrer seg av gangen (se avsnitt 1.6.1), reduseres antall transisjoner til  $n$ . Med kjennskap til kretsen og tidsforholdene, vil dette diagrammet hjelpe oss til å ha kontroll med at vi har fått med oss alle transisjonene. I figur 25 er det eksakt 2 *transisjoner* ut fra *hver* av tilstandene A,  $B_1$ ,  $B_2$ ,  $C_1$ ,  $C_2$ , og det samme også i de ytre tilstandene A, B og C.

## 1.7 Tilstandstabeller for sekvensstyring

Overgang mellom de aktuelle tilstander, med derav følgende utlesning av utgangssignaler, kan oppstilles i forskjellige former for *tilstandstabeller*. Tilstandstabeller er hensiktsmessige for systematisk og oversiktlig behandling under programmeringen, og programmet kan hensiktsmessig arbeide direkte på disse tabellene. Vi skal nedenfor se på noen viktige former for tilstandstabeller.

### 1.7.1 Huffman-tabell

En av de viktigste typer sekvenstabeller er Huffman-tabellen. Huffman-tabellen uttrykker meget direkte teorien for tilstandstransformasjon, gitt foran. Forutsatt at Huffman-tabellen holdes på moderat størrelse, vil den også være hensiktsmessig som datastruktur for sekvensstyring innen sann-tidsprogram. Dette kan nødvendiggjøre tilstrekkelig system-oppdeling.

En Huffman-tabell fremstilles som en tre-dimensjonal matrise med  $n \times m \times 2$  elementer, hvor  $n$ =antall tilstander og  $m$ =antall innganger. De to elementer  $k = 1, 2$  for hver verdi  $i, j$  ( $i \leq m, j \leq n$ ) inneholder informasjon om henholdsvis **neste tilstand** og **aksjon** eller **utgang**. De  $m$  innganger i tabellen representerer alle forekommende "inngangene" i Mealys modell. Disse kan være, og er ofte, avledet av *inn-signalene*, som en kombinatorisk funksjon. Inngangene i modellen, og dermed kolonnene i Huffmantabellen, representerer altså de *diskrete hendelsene* som ble omtalt i innledningen.

Huffman-tabellen kan oppstilles slik:



Tilstander $q$	Innganger					
	$p_1$	$p_2$	....	$p_i$	....	$p_m$
$q_1$	-/-	-/-		-/-		-/-
$q_2$	-/-	-/-		-/-		-/-
$q_3$	-/-	-/-		-/-		-/-
...						
$q_j$	-/-	-/-		$q_s/w$		-/-
...						
$q_n$	-/-	-/-		-/-		-/-

Fig. 26 . Huffman-tabell, skjematisk.

Element (i,j) inneholder to komponenter: neste tilstand  $q_s$  og utgang (aksjon)  $w$ .

### Eksempel: Gardintrekk-mekanisme.

Som et meget enkelt eksempel på bruk av Huffman-tabeller skal vi betrakte en “gardintrekk”-mekanisme, av den typen som brukes for gardiner i auditorier, hvor motorstyring skjer fra en trykknappsats med tre knapper:

1. Gå mot venstre (  ) (lukk)
2. Gå mot høyre (  ) (åpne)
3. Stopp

Videre er mekanismen forsynt med en endebryter i hver ende av gardinføringen, slik at motoren stanser når gardinen er helt lukket, henholdsvis helt åpen. Disse gir altså to signaler:

4. Venstre ende (dvs. helt lukket)
5. Høyre ende (dvs. helt åpen)

Proessen representerer en rent Prosessbetinget sekvens. Systemet har i alt følgende fem mulige tilstander:

1. I ro ved venstre ende
2. I ro ved høyre ende
3. I ro i en mellomstilling
4. Underveis mot venstre
5. Underveis mot høyre

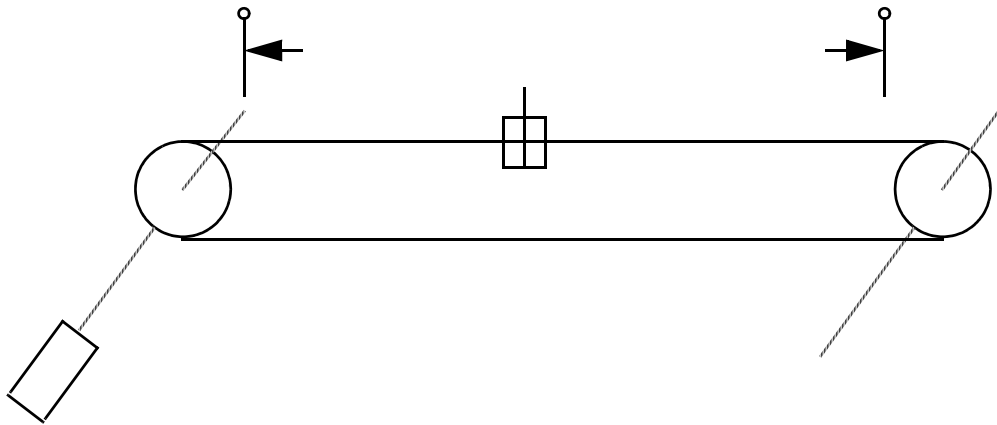


Fig. 27 . Gardintrekk-mekanisme, skjematisk

Gardintrekk-mekanismen kan beskrives med følgende Huffman-tabell, som uttrykker overgangen  $\tau$  fra en tilstand til en annen som følge av forandringer i inngangene. Samtidig med hver overgang angis tilhørende utgangsfunksjon  $w$ .

Parentesmerkete situasjoner forekommer normalt ikke, bortsett fra eventuelt transient. Disse “situasjoner” bør likevel gis fornuftig aksjon, da de uttrykker feil-tilfeller, først og fremst utstyrsfeil.

Tegnforklaring for utganger:

V: Start mot venstre

H: Start mot høyre

S: Stopp motor

-: Ingen aksjon



Tilstander <b>q</b>	Innganger				
	Trykknapper			Endebrytere	
			STOPP	venstre	høyre
	1	2	3	4	5
1. I ro venstre	1/-	5/H	1/-	1/-	(1/-)
2. I ro høyre	4/V	2/-	2/-	(2/-)	2/-
3. I ro mellom	4/V	5/H	3/-	1/-	2/-
4. Mot venstre	4/-	4/-	3/S	1/S	(3/S)
5. Mot høyre	5/-	5/-	3/S	(3/S)	2/S

Fig. 28 Huffman-tabell for gardintrekk-mekanismen

For store systemer blir det gjerne stort antall tabell-innganger og dermed stor og uoversiktlig Huffman-tabell. Det kan da være hensiktsmessig å oppdele tabellen i flere mindre Huffman-tabeller. Dette innebærer imidlertid at modellen oppdeles i flere deler, som igjen er en forbundet graf, på et høyere hierarkisk nivå. En slik hierarkisk inndeling må være mulig ut fra systemets (prosessens) egenskaper, hvis en oppdeling av modellen skal være gjennomførbar. Hvis en slik inndeling synes vanskelig, henger det sammen med sterk og kompleks kopling innen systemets logiske struktur. Prøv å omarbeide og forenkle denne.

Det er viktig å merke seg forskjellen mellom innleste prosess-signaler og tabellinnganger. Selv om binære inn-signaler ofte opptrer direkte som tabellinnganger, som i eksempelet ovenfor om gardintrekk-mekanisme, er

dette slett ingen regel. Tabellinnganger svarer direkte til Mealy-modellens innganger, og disse kan prinsipielt like gjerne være boolske program-variable, beregnet i en eller annen program-del. I praksis har tabellinngangene dog ofte en forholdsvis nær sammenheng med innleste prosess-variable.

Forskjellen mellom Mealys og Moores modeller er som nevnt utgangene. Mealys modell knytter utgangene til transisjonene, mens Moore knytter dem til tilstandene. I eksempelet med gardintrekk-mekanismen vil etter Mealys modell aksjonene være å *starte* drivmotor til høyre eller til venstre, samt *stopp* av motor. Ved Moores modell ville tilsvarende aksjoner være *kjør* motoren. Ingen stopp er nødvendig som egen aksjon, for man forlater bare tilstanden hvor det kjøres. Realiseringen etter Mealys modell innebærer derfor utgangssignal i form av en puls til en (f.eks.) relekopling<sup>1</sup> med innebygget **hold**: For starting legg inn motorrele med holdekopling, og for stopp: frigjør holdekoplingen. I dette tilfelle kan det ut fra realiseringen med en viss rett argumenteres i favør av Moores modell: En digital utgang (en bit) fra datamaskinen går direkte til motorreleet, som derfor ligger inne så lenge som utgangen genereres. Ved feil som medfører at datamaskinen stopper kan man da utforme program og utgangskretser slik at utgangssignalet går til 0, hvorved motoren stopper. Ved Mealys modell må den samme sikkerheten oppnås ved at f. eks. drivspenningen for releene faller bort sammen med at datamaskinen faller ut.

### **Et annet eksempel:**

En kontinuerlig variabel  $x$  innleses og sammenliknes med en grenseverdi  $x_0$ , lagret som parameter. Hvis grensen overskrides, skal alarm gis. I Huffman-tabellen inngår da den boolske

$$b_1 = ((x - x_0) > 0)$$

som tabellinngang. I elementet for aktuell tilstand og denne inngang angis alarmtilstanden og tilhørende aksjon.

---

1. En slik relekopling med hold kan naturligvis gjerne realiseres i programvare. Det er den *prinsipielle* virkemåten vi poengterer her.

### 1.7.2 Beslutningstabell

Beslutningstabell (eng.: "decision table") er et hjelpemiddel som gir en overskuelig fremstilling av sammenhengen mellom betingelser og aksjoner. Den egner seg derfor godt for valg mellom en rekke *kombinatoriske* funksjoner.

Beslutningstabellen er særlig fordelaktig ved komplekse valgsituasjoner og gir, som få andre metoder, programmereren en god oversikt over problemet, samtidig som den gir lett mulighet for kontroll av at alle kombinasjoner av inngangene er dekket. I så måte er den på linje med Huffman-tabellen. Beslutnings-tabellen kan også benyttes direkte som programunderlag. Benyttes den sammen med Huffman-tabell i sekvensstyringsprogram, vil disse to kunne utfylle hverandre godt.

En beslutningstabell oppstilles, på papir, gjerne i 4 deler slik:

betingelsesstamme ("condition stub")	Regler							betingelsesdel ("condition value")
	1	2	3	4	5	6	7	
Liste over betingelser								
Liste over aksjoner								
aksjonsstamme ("action stub")	Aksjoner som utføres							aksjonsdel ("action value")

Fig. 29 . Beslutningstabell, skjematisk



Metoden kan best beskrives gjennom et enkelt eksempel, og vi kan da ta for oss en heis-styring, forholdene rundt en etasje. Senere vil vi utbygge eksempelet til å omfatte hele heisstyringen, men for å ikke gjøre eksempelet for komplekst i begynnelsen, tar vi som et utdrag forholdene rundt en etasje, som vi antar er en “typisk”, dvs. en mellometasje, hverken den helt nederste eller helt øverste. Vi kan kalle dette etasje nr.  $e_1$ . På denne bakgrunn gjør vi den forenkling at vi her ikke tar hensyn til oppkall fra andre etasjer, idet dette hører med til den utvidede betraktningen.

Etasjene nummererer vi fra 1 (nederste) og oppover. Symbolsk betegner vi etasjen heisen står i som  $e_1$ , og etasje som heisen skal gå til:  $e_2$ . Dette betyr da at heisen skal gå oppover hvis  $e_2 > e_1$ , og nedover hvis  $e_2 < e_1$ .

Vi har her følgende mulige normale inngangssignaler:

Dør lukket (dvs. mekanisk kontakt koplet til dør)	
Etasjeindikator (heisen befinner seg innen etasjen)	
Knapp i kupé:	Gå til etg. $e_2$
Knapp i kupé:	Stopp
Knapp i etasjen:	Jeg vil OPP
Knapp i etasjen:	Jeg vil NED

Aksjonene er:

- Start nedover
- Start oppover
- Stopp
- Feilindikering

For enkelhets skyld vil vi ved feil kun gi Feilindikering, uten å forfølge dette videre. I et virkelig tilfelle må dette behandles komplett.

Vi kan sette opp følgende beslutningstabell:

	Regler					
	1	2	3	4	5	6
Dør lukket	1	1	-	0	0	1
Etasjeindikator	1	1	-	0	1	0
Kupéknapp $e_2 > e_1$	-	1	-	-	-	-
Kupéknapp $e_2 < e_1$	1	0	-	-	-	-
Kupéknapp $e_2 = e_1$	-	-	-	-	-	-
Kupéknapp STOPP	0	0	1	-	-	0
Etasjeknapp OPP	-	-	-	-	-	-
Etasjeknapp NED	-	-	-	-	-	-
Start nedover	X					
Start oppover		X				
Stopp			X	X		
Feilindikering				X		

Fig. 30 . Beslutningstabell for heiskupé

For inngangene betyr - at det ikke tas hensyn til dette, spørsmålet er irrelevant. Svar 1 eller 0 er da likegyldig. For aksjoner betyr X at aksjonen skal utføres, mens blankt betyr ingen aksjon. Signalet fra kupéknappene angir numerisk ønsket etasje. Dette kunne angis som en heltallsvariabel, ikke boolsk. Imidlertid kan det forekomme at noen trykker på knapper både opp og ned samtidig. Dette må tas hånd om på en fornuftig måte, og her er det valgt å la passende elektroniske sammenlikningskretser gi de tre signalene som direkte inngår som innganger i tabellen, dvs. der står det boolske resultatet av sammenlikningen med den etasje heisen befinner seg i. Vi kommer for øvrig tilbake til dette spørsmålet om samtidig trykk på flere knapper.

Med et annet system, kunne det meget vel tenkes at det bare var én mulighet, uttrykt som en numerisk verdi. Eksempel: Nivåmåling i en tank.

Målesignalet gir entydig nivået. Da ville det være naturlig å sette de boolske relasjonene inn i tabellen, slik som her, men hvor samtidig “klaff” på flere enn én samtidig ikke var mulig, fordi sammenlikningen foregår i programmet, dvs. en beregning på grunnlag av *én* foreliggende innlest verdi. Dette er en viktig detalj og viser hvordan numeriske variable benyttes til å gi innganger i beslutningstabellen, og dette er også et eksempel på prosessbetingelser, omtalt i avsnitt 1.6.2, side 26.

Tabellen må sies å gi et godt og oversiktlig bilde av forholdene, og det er forholdsvis lett å kontrollere at alle kombinasjoner er tatt med.

Kolonnene i tabellens høyre del representerer en rekke **regler**: En kolonne angir kombinasjoner av inngangene, og hvis et foreliggende tilfelle har 0 på alle de steder hvor regelen foreskriver 0, og 1 alle steder hvor det står 1 i kolonnen, har vi “klaff”, og tilhørende aksjoner nedenfor skal utføres. For de innganger hvor det står - i kolonnen, tas ikke hensyn til inngangsverdien, dvs. inngangsverdien er *irrelevant*.

Sammenlikner man de forskjellige reglene, kan det tenkes at flere gir samsvar med foreliggende situasjon. Denne *flertydighet* løses vanligvis slik at vi foreskriver undersøkelse av regel etter regel fra venstre mot høyre, inntil en regel med samsvar er funnet. Da stanser sammenlikningen. Det kan også tenkes at ingen regler passer: Beslutningstabellen ville da være *man-gelfull*. Dette løses ved at man tar med, som en spesiell regel til slutt, en *ellers*-regel, som dekker “default”-situasjon. Denne regelen settes altså opp slik at den gir klaff i alle tilfeller der de øvrige ikke passet. Ofte vil denne regelen foreskrive aksjonen “feil-reaksjon”, eller den kan bringe kontrollen i programsløyfen tilbake til utgangspunktet uten å gjøre noe. Gjennomløpet av tabellen blir altså uten virkning. Denne algoritmen om å undersøke fra venstre mot høyre innebærer at innbyrdes rekkefølge mellom reglene kan være av betydning.

Når vi skal lage et program som behandler beslutningstabellen, går vi frem slik:

1. Oppstill en boolsk BETINGELSESMATRISSE svarende til betingelser og regler, hvor alle JA angis med 1 og alle NEI eller IKKE RELEVANT med 0.

2. Oppstill en boolsk MASKEMATRISE av samme form som Betingelses-matrisen. Her innsettes 1 tilsvarende alle relevante spørsmål og 0 i alle ikke-relevante.
3. En gitt situasjon karakteriseres med en DATAVEKTOR med boolske (binære) komponenter som angir svar på betingelsene.

Vi får derved følgende algoritme:

I tur og orden undersøkes slik om datavektoren svarer til en av reglene: Dann OG-funksjonen (logisk produkt) av datavektor og vedkommende kolonne i maske-matrisen. Hvis resultatet er identisk med tilsvarende kolonne i betingelsesmatrisen, er overensstemmelse funnet. Tilsvarende aksjon skal utføres.

For dette aktuelle eksempelet blir betingelsesmatrise og maskematrise slik:

#### BETINGELSEMATRISE

1	1	0	0	0	1
1	1	0	0	1	0
0	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0
0	0	1	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

#### MASKEMATRISE

1	1	0	1	1	1
1	1	0	1	1	1
0	1	0	0	0	0
1	1	0	0	0	0
0	0	0	0	0	0
1	1	1	0	0	1
0	0	0	0	0	0
0	0	0	0	0	0

Antar vi eksempelvis at situasjonen kan beskrives slik: Vi er i 2. etasje. Heisdøren er lukket, etasjeindikatoren viser at heisen befinner seg i denne etasjen, og det trykkes på kupéknapp for å gå til 4. etg. Da er datavektor slik:

1  
1  
1  
0  
0  
0  
0  
0

Bemerk at datavektoren beskriver foreliggende tilfelle, derfor vil aldri noen komponent her være “ikke-relevant”, “ubestemt” eller liknende.

For denne datavektoren blir OG-funksjonen mellom datavektor og hver spalte i maskematrisen:

1	1	0	1	1	1
1	1	0	1	1	1
0	1	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Undersøkes overensstemmelsen med Betingelsesmatrisen, ser man at regel 2 gir overensstemmelse, de andre regler ikke. Aksjon oppført under regel 2 skal da utføres, og dette er “Start oppover”.

En nyttig regel for oppsetting av en betingelsestabell: Utnytt eksisterende innbyrdes forhold mellom betingelsene. Hvis JA for en betingelse nødvendigvis medfører NEI for en annen, bør én av disse angis med -, dvs. LIKEGYLDIG (Ikke relevant).

Eksempel: Betingelser:

Alder > 68 år	Det er åpenbart ikke mulig med JA på
" > 50 "	alle disse betingelser samtidig, dvs.
" < 18 "	de utelukker hverandre (delvis).

I tilfellet med heisen har vi noe tilsvarende med kupé-knappene: Det gir ingen mening å gå både opp og ned samtidig. I dette tilfelle er imidlertid slikt motstridende *signal* absolutt mulig, det viser at en person har trykket flere knapper samtidig, evt. det har oppstått f. eks. overledning på kablingen eller annen liknende systemfeil. Dette må det tas hensyn til, og det kan være nærliggende i første omgang å anta at det er trykket samtidig på flere knapper. Systemfeil kan vi ta hånd om ved at hvis situasjonen varer over en viss tid, f.eks. en time, kan dette utløse feil-alarm. Vi går ikke inn på muligheten for systemfeil her, men derimot må vi ta hensyn til samtidig trykking av flere knapper. Dette har vi gjort i tabellene over: Man har her bestemt at knappen for egen etasje helt neglisjeres. Da har vi tre muligheter igjen:

entydig Neste etasje er ovenfor,  
entydig Neste etasje er nedenfor,  
Oppkall om både opp og ned

I dette eksempelet viser tabellen at i tilfellet “både opp og ned” er “ned” gitt prioritet over “opp”, dvs. det skal da reageres som om bare “ned” er trykket. M.a.o. for at kommandoen skal oppfattes som “oppover”, må trykkes entydig knapp for høyere etasje.

Med Beslutningstabeller kan blant annet oppnås følgende:

1. Logikken fremtrer presist og på kompakt form.
2. Kompliserte tilfeller blir oversiktlig fremstilt.
3. Tydelig relasjon mellom variable.
4. Forenklet programmering.
5. Enkel og oversiktlig dokumentasjon.
6. Overflødighet, tvetydighet og mangelfull tabell avsløres systematisk.

Beslutningstabellen egner seg godt til evaluering av betingelsene i kombinatoriske funksjoner som bestemmende for tilstandstransisjon i sekvensstyringsprogram.

## 1.8 Eksempel på sekvensstyre-system: Heis-styring

Vi vil nå utbygge eksempelet med heisstyring og velger å benytte Huffman-tabell og beslutningstabeller i kombinasjon: Huffman-tabell for det overordnede system og behandling av tilstander, mens beslutningstabellen benyttes til å avgjøre tilstandstransisjonene. Dette bygger videre på de oppstillinger av beslutningstabell vi laget i avsnitt 1.7.2 .

### 1.8.1 Heis-systemet

Heis-systemet er av de vanlige moderne systemene som kan beskrives slik:

Heisen skal betjene et ubestemt antall etasjer. Ved hver etasje (mellom nederste og øverste) finnes to trykknapper merket **OPP** og **NED**. Ved nederste og øverste etasje er det kun én knapp, henholdsvis **OPP** og **NED**. Trykk på en etasjeknapp kan gjøres når som helst, og dette registreres av styreprogrammet som oppfører dette som "bestilling" i to køer, *oppover-kø* og *nedover-kø*.

Heis-styresystemet kan beskrives med tilstandsmodellen i Fig. 31 på side 48. Utgangsposisjonen er tilstand nr. 1: Heisen står i ro, og intet er bestilt. Den står i en tilfeldig etasje. Som man ser av figuren, skiller ikke tilstandene mellom de enkelte etasjer, men programmet holder naturligvis rede på hvilken etasje heisen står i. Hvis det nå kommer "bestilling" på å gå oppover, enten ved at en person går inn i heiskupeen og trykker for en etasje ovenfor, eller det trykkes på en av etasjeknappene ovenfor, får vi først tilstands-transisjon til tilstand 2. En intern variabel som angir retning settes derved til OPP. Så startes heisen og vi får tilstandstransisjon til tilstand 3. Idet heisen passerer en etasje, går systemet til tilstand 4. Hvis dette var den bestilte etasje, går tilstand over til nr. 2, samtidig med aksjon **stopp**. Hvis etasjen derimot ikke var bestilt, går tilstanden over til nr. 3 og det fortsettes oppover. Slik hoppes det mellom tilstand 3 og 4 for hver "uvedkommende" etasje, inntil riktig etasje nås, og tilstanden altså blir 2. Hvis det etter dette foreligger flere bestillinger oppover (oppover-køen), fortsettes fra tilstand 2 til 3, og det hele gjentas.

Modellen er symmetrisk, så beskrivelsen ovenfor gjelder tilsvarende for retning **nedover**. Mellomtilstanden 1: **Nøytral** er dels hvor systemet befinner seg når intet er bestilt, og dels overgang mellom "opp" og "ned". Som vi ser av denne modellen, vil systemet "låse seg opp" i én retning så lenge det finnes bestillinger i denne. Dette er ønskelig, og overensstemmende med vanlige regler.

La oss nå se nærmere på bestillingskøene, kort omtalt på side 45. Egentlig bør vi kalle dem *bestillingstabeller*, for å unngå misforståelsen at de opererer etter "først inn - først ut"-prinsippet, som er vanlig for *køer*. Det er altså to enkelt-tabeller, en for hver retning heisen går. Vi organiserer dette som en todimensjonal tabell (matrise eller array), hvor den ene dimensjonen svarer til retning OPP eller NED, og den andre til etasjene, i dette tilfelle 1...5. Vi illustrerer dette slik:

**Bestillingstabell:**

Etasje	OPP	NED
5		
4		
3		
2		
1		

I utgangspunktet er bestillingstabellens celler tomme. Når en bestillingsknapp trykkes, plasseres et merke, flagg, i tilhørende celle, og dette skal bewirke at heisen stopper ved vedkommende etasje når den er på vei henholdsvis oppover eller nedover. Idet heisen stopper ved en etasje, nullstilles tabellcellen.

Trykkes en etasjeknapp, skal flagg settes i OPP- eller NED-cellen for vedkommende etasje, uavhengig av heisens umiddelbare posisjon eller tilstand (bortsett fra hvis heisen allerede står i vedkommende etasje). Trykkes en kupéknapp (bestillingsknapp i heiskupéen), skal flagg settes dersom man trykker på knapp for en høyere liggende etasje når heisen er under veis oppover. Tilsvarende for nedover. Derimot skal ikke registreres bestilling til lavere etasje når heisen er under veis oppover, og tilsvarende høyere etasje mens den er på vei ned. Dette er valgt fordi slike bestillinger kan anses for feilbetjening: Det kommer inn i heiskupeen passasjerer som egentlig skal fraktes i motsatt retning av heisens rute, eller det er trykket feil. På denne



måten får vedkommende passasjer mulighet for å trykke om igjen. Evt. må han/hun bare bli med heisen i "feil retning", og kan gjenta bestillingen når heisen er kommet til høyeste, henholdsvis laveste, bestilte etasje. På denne måten oppnås også en enkel indikering på lysende knapper i kupeen av fremtidige stopp, slik at passasjerer får et klart bilde av stoppene fremover.

Forbindelsen mellom trykknappene og tabellen er forholdsvis triviell og realiseres nokså direkte med programmert innlesning på lavt nivå av signaler fra knappene. Dette blir derfor ikke berørt videre her; vi forutsetter i det følgende at slik forbindelse er realisert, slik at knapptrykking straks kommer til syne som logiske variable (SANN) i tabellens celler.

Ved hver etasje er en sensor som gir et logisk signal SANN når heisen ankommer en etasje. Signalet blir SANN like før ankomst til etasjen, slik at heismotoren vinner å stanse. Det går over til FALSK idet heisen forlater etasjen. Dette signalet endrer en tellervariabel `s_etg` som derved til enhver tid viser siste etasje heisen var ved, enten heisen da stoppet eller bare passerte. Variabelen `s_etg` tvangssettes til 1 når heisen kommer til 1. etg., for å sikre synkronisering.

En Huffman-tabell for denne heis-styringen kan nå settes opp, som i Fig. 32 på side 49. Tabellen (og modellen) tar bare for seg de "overordnede" funksjoner, tilstandsovergang i forbindelse med gang mellom etasjene, og den er noe forenklet, idet den bl.a. ser bort fra muligheten for trykk på stoppknappen mens heisen er under veis mellom etasjene. Huffmantabellen må ses i sammenheng med beslutningstabellen, liknende Fig. 30 på side 40. Vi kan sette opp en slik beslutningstabell for hver av tilstandene, og i programmet realiseres beslutningstabellen som et array av strukturer (struct i C, record i Pascal). En peker skifter mellom de enkelte beslutningstabellene, i henhold til tilstanden.

På noen følgende sider er vist beslutningstabeller for noen av tilstandene, og systematikken skulle fremgå av disse. Beskrivelsen er forhåpentlig tilstrekkelig til at man ser sammenhengen, men for fullstendighets skyld bør kanskje påpekes at her er "ellers"-regelen, som ble nevnt på side 41, at intet gjøres. Systemet forblir i sin tilstand så lenge ingen av reglene tilfredsstilles.

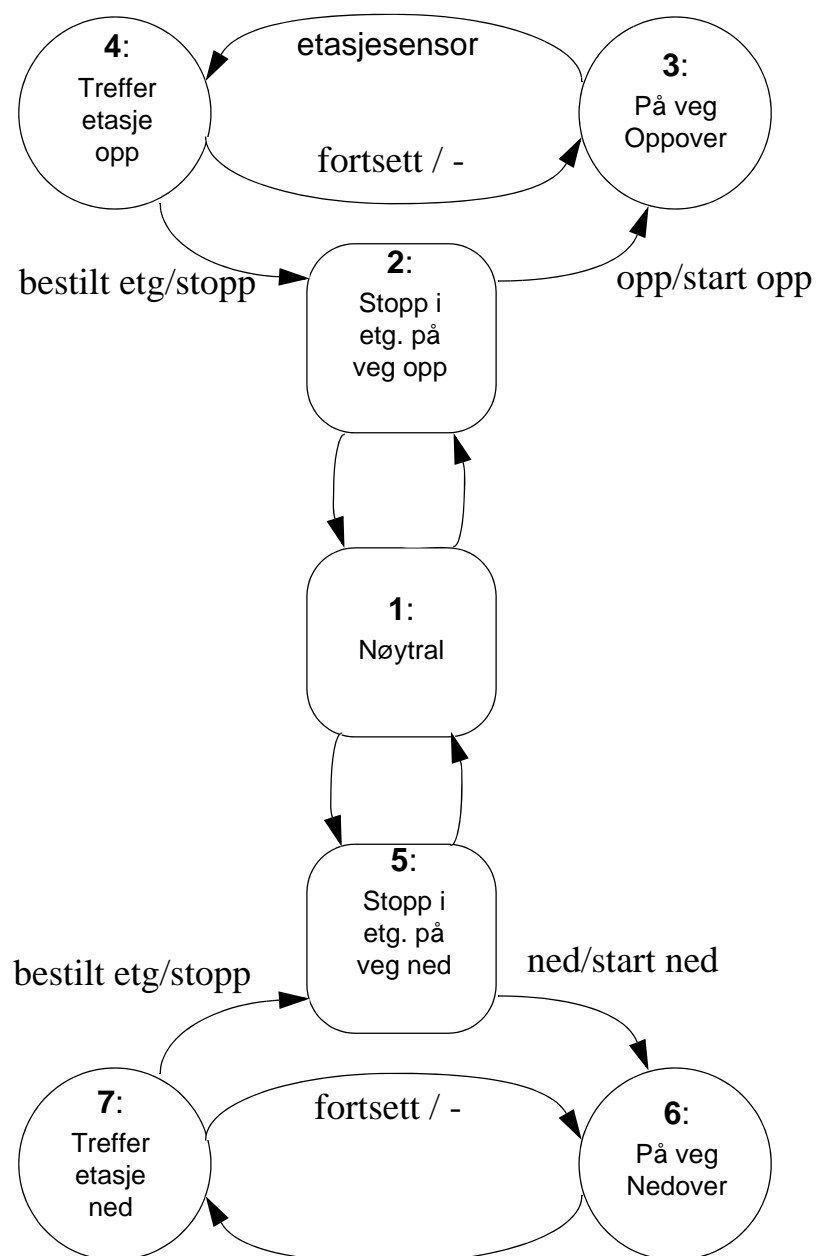


Fig. 31 . Tilstandsmodell for heis

Tilstander <b>q</b>	Innganger					
	Bestillingstabell		Etasjebryter	Programfunksjoner		
	Best.flagg finnes i celle >s_etg	Best.flagg finnes i celle <s_etg		tref-fer bestilt etasje	tref-fer "gal" etasje	tom bestil-lings-kø
1. Nøytral	2/-	5/-	-/-	-/-		
2. Stopp på veg opp	3/MO	-/-	-/-	-/-		1/-
3. På veg oppover	-/-	-/-	4/-	-/-		
4. Treffer etasje opp	-/-	-/-	-/-	2/MS	3/-	
5. Stopp på veg ned	-/-	6/MN	-/-	-/-		1/-
6. På veg nedover	-/-	-/-	7/-	-/-		
7. Treffer etasje ned	-/-	-/-	-/-	5/MS	6/-	
Tegnforklaring: MO: Motor Opp; MN: Motor Ned; MS: Motor Stopp						

Fig. 32 Huffmantabell for heisstyring

	Regler	
	1	2
Dør lukket	1	1
Etasjeindikator	-	-
Bestilling i celle >s_etg	-	1
Bestilling i celle <s_etg	1	0
Kupéknapp STOPP	0	0
Til tilstand 5 (ned)	X	
Til tilstand 2 (opp)		X

Fig. 33 . Beslutningstabell for tilstand 1

	Regler		
	1	2	3
Dør lukket	-	-	0
Etasjeindikator	1	1	0
Bestilt etasje	1	0	-
Til tilstand 2, Stopp: MS	X		
Til tilstand 3		X	
Feilindikering			X

Fig. 34 Beslutningstabell for tilstand 4

	Regler		
	1	2	3
Dør lukket	1	-	-
Etasjeindikator	1	1	0
Ventetid i etg. ferdig	1	1	1
Best.flagg finnes i celle >s_etg	1	0	-
Kupéknapp STOPP	0	-	-
Til tilstand 3, Start: MO	X		
Til tilstand 1		X	
Feilindikering			X

Fig. 35 . Beslutningstabell for tilstand 2

For beslutningstabellene er det nærliggende å spørre om regler for å kontrollere om alle kombinasjoner er med. En systematisk måte er Karnaugh-diagram, som vil bli gjennomgått i et senere kurs i logikk-design. Karnaugh-diagram er imidlertid ikke praktisk, hvis antall variable overstiger 5. En annen metode er Quine-McClusky's, men denne er ganske omstendelig. Teoretisk har man, med  $n$  uavhengig variable,  $2^n$  kombinasjoner. Dette blir imidlertid alt for tungvint. I de aller fleste tilfeller ønsker vi tilstandstransisjon i noen få tilfeller, når bestemte betingelser inntreffer. Før dette skjer, foretas det intet. Dermed blir de interessante kombinasjonene ganske fåtallige, slik at systemet blir oversiktlig. Karnaugh-diagram kan da benyttes, hvis man ikke innser kombinasjonene direkte. Karnaugh-diagrammet gir bl.a. en metode til å finne "don't cares", altså - (strek) i diagrammet. Disse kan også finnes direkte, der to regler med samme aksjon(er) har like betingelser unntatt én, hvor vi har 0 og 1.

Som det fremgår av tabellene ovenfor, har vi egentlig ikke benyttet oss av Huffmantabellen. Beslutningstabellene inneholder selv aksjoner for skifting av tilstand, og det hører én beslutningstabell til hver tilstand. Tilstanden er gjennom dette *realisert* gjennom sin beslutningstabell. Dermed er det tilstrekkelig med settet av beslutningstabeller, organisert innen styre-

programmet på en systematisk måte med peker som skifter mellom tabellene i henhold til tilstandene.

Til slutt skal vi se hvordan vi kan stille opp en beslutningstabell for setting av flagg i bestillingstabellen, basert på knappetrykk (umerkede rubrikker: likegyldig ("dont'care")).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Kupéknapp 5	1															
Kupéknapp 4		1			1											
Kupéknapp 3			1			1										
Kupéknapp 2				1			1									
Kupéknapp 1								1								
Etasjeknapp ↑ i 4. etg.									1							
Etasjeknapp ↑ i 3. etg.										1						
Etasjeknapp ↑ i 2. etg.											1					
Etasjeknapp ↑ i 1. etg.												1				
Etasjeknapp ↓ i 5. etg.													1			
Etasjeknapp ↓ i 4. etg.														1		
Etasjeknapp ↓ i 3. etg.															1	
Etasjeknapp ↓ i 2. etg.																1
Knapp > s_etg	1	1	1	1												
Knapp < s_etg					1	1	1	1								
Knapp = s_etg	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bestillingstab. OPP, 5.etg.	X															
Bestillingstab. OPP, 4.etg.		X							X							
Bestillingstab. OPP, 3.etg.			X							X						
Bestillingstab. OPP, 2.etg.				X							X					
Bestillingstab. OPP, 1.etg.												X				
Bestillingstab. NED, 5.etg.													X			
Bestillingstab. NED, 4.etg.					X									X		
Bestillingstab. NED, 3.etg.						X									X	
Bestillingstab. NED, 2.etg.							X									X
Bestillingstab. NED, 1.etg.								X								

Fig. 36 Beslutningstabell for oppsetting av Bestillingstabell

## 1.9 IEC-848: Grafisk notasjon for sekvensstyring

Mens de grunnleggende Mealy og Moore -modellene gjerne fremstilles grafisk i form av generelle rettede grafer, slik som Fig. 22 på side 28 og eksempelet i Fig. 31 på side 48, finnes det spesielle grafiske notasjoner som er mer "skreddersydd" for sekvensstyring. Disse inneholder en del spesifikke detaljer som gir mer detaljert uttrykksfullhet. En viktig blant disse er IEC-standard 848 (Preparation of function charts for control systems) som også bygger på den franske standarden Grafcet og den tyske DIN 40719. For enkelhets skyld omtaler vi denne IEC-måten derfor som Grafcet-metoden, selv om det korrekte egentlig er "Standard IEC-848".

IEC-848 benytter også rettede grafer. En slik graf kalles her *funksjonsdiagram* (Function chart). Hierarkisk oppdeling (se side 29) er meget enkelt å gjennomføre med denne metoden. Det som i Mealy og Moores modeller kalles *tilstand* heter *trinn* i IEC-848. Den svarer til Moores modell ved at aksjoner er knyttet til trinnene (tilstandene).

Den noe nyere standarden IEC1131-3, beskrevet i kapittel 1.10, har tatt opp i seg IEC-848 med noen modifikasjoner. Denne del av IEC1131-3 kalles "Sequential Function Charts" (SFC) og er én av de fem metodene som inngår i IEC1131-3.

Et funksjonsdiagram består av **trinn**, **transisjonsbetingelser** og **rettede forbindelser**, som sammenknytter trinn og transisjoner.

### 1.9.1 Trinn

Et trinn tegnes som et rektangel (helst kvadrat) med etikett (nummer og/eller tekst). Teksten kan vise hva som skjer i trinnet. Ett bestemt trinn i et sekvenssystem er *starttrinnet*. Dette representerer starten i sekvensen og tegnes som et rektangel med dobbel ramme.

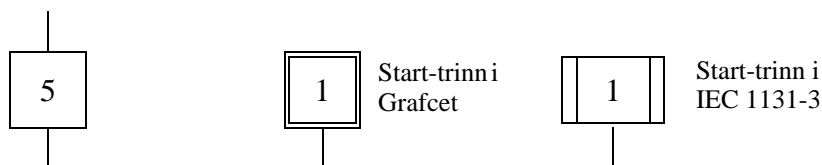


Fig. 37 . Symbol for trinn i Grafcet og 1131-3

Under beskrivelsen av et sekvensielt system kan vi ha bruk for å illustrere en øyeblikkssituasjon. Da vil systemet befinne seg i en bestemt tilstand, dvs. ett bestemt trinn i funksjonsdiagrammet er *aktivt*. Dette kan illustreres med en prikk i trinn-rektangelet:

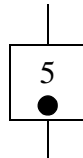


Fig. 38 .

### 1.9.2 Kommandoer og aksjoner

Til et trinn kan knyttes *kommandoer* og *aksjoner* (begge går under begrepet *aksjoner* i Moore) som utføres mens trinnet er aktivt. Det skilles ikke strengt mellom kommandoer og aksjoner, men kommandoer er noe som utføres i trinnet, mens en aksjon er noe som forårsakes i trinnet. Eksempel: "Åpne ventil 2" er en kommando, mens "Åpning av ventil 2" er en aksjon. Det er ellers ikke grunn til å utdype denne forskjellen videre, og i de fleste tilfeller vi i det følgende omtaler en *kommando*, kan man like gjerne erstatte den med en *aksjon*.

En kommando eller aksjon angis i en rektangulær boks knyttet til trinnet:

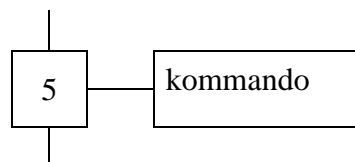


Fig. 39 .

En kommando iverksettes når det tilhørende trinnet blir aktivt. Idet trinnet blir deaktivert idet et etterfølgende trinn overtar som aktivt, vil kommandoen enten

- terminere, eller
- beholde sin tilstand.



Det er m.a.o. anledning til å skille mellom en "enkelt-handling" (transient) og en aksjon som strekker seg over tid ("steady state"). Forskjellen fremgår av følgende lille eksempel:

- a. En ventil åpnes i trinn 5, og det er underforstått at ventilen lukkes idet trinnet forlages:

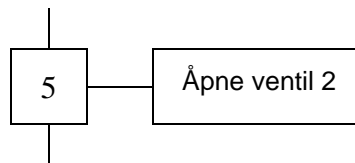


Fig. 40 .

- b. Hvis lukkingen skal knyttes til et etterfølgende trinn kan vi angi selve åpningen slik:



Fig. 41 .

Vi skal senere vise en mer systematisk måte til å angi dette på.

Flere aksjoner eller kommandoer kan knyttes til samme trinn, tegnet i hver sin boks. Innbyrdes arrangering av boksene er uten betydning for rekkefølgen av utførelse. Her vises en form, hvor b) er en forenklet form av a):

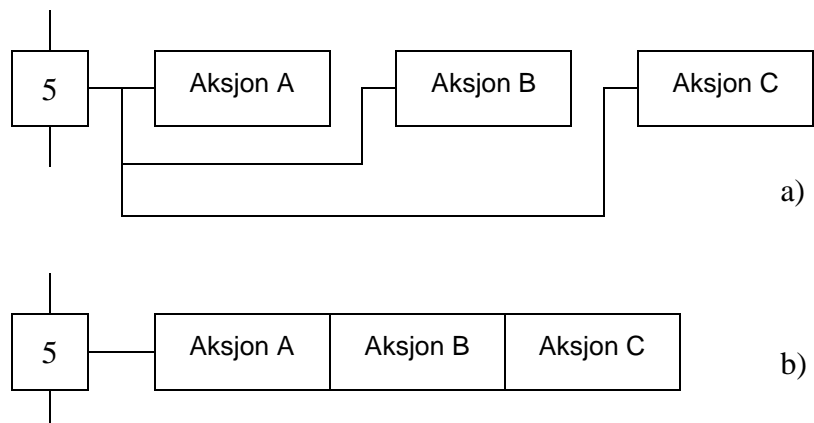


Fig. 42 .

Boksene kan gjerne plasseres under hverandre. Her er også form b) en forenklet tegnemåte for formen a):

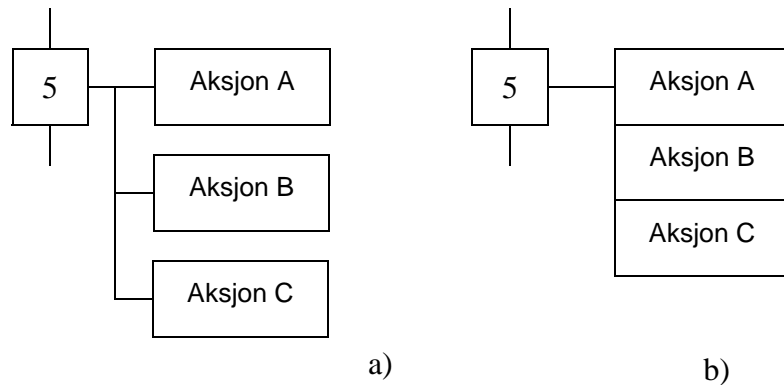


Fig. 43 .

En transisjon foregår langs forbindelseslinjen mellom to trinn og går "normalt" nedover eller mot høyre i figuren. Retningen trenger da ikke å angis med piler. Settes på pilspiss, kan transisjonen gå i hvilken som helst retning i skjemaet. Transisjonen symboliseres ved et lite tverrstrekk på forbindelseslinjen. Ved siden av tverrstreken angis *betingelsen* som fører til transisjonen. Transisjonen foregår *bare* når *både* foregående trinn er *aktivt* og betingelsen (B) er *sann*:

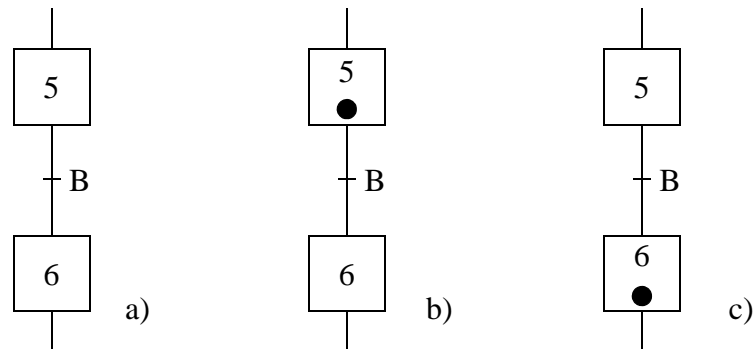
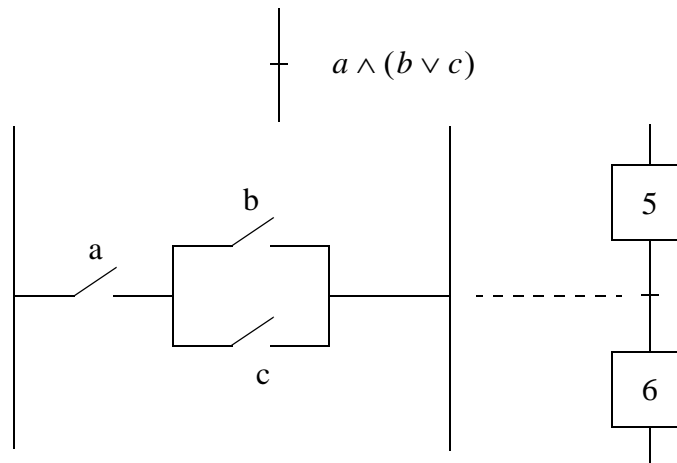


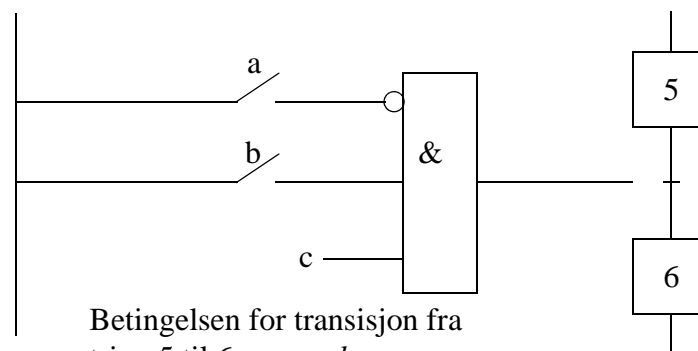
Fig. 44 .

I tilfelle a) foregår ingen transisjon, uansett om B er sann eller ikke, fordi trinn 5 ikke er aktivt. I b) er det klart for transisjon som inntreffer idet B blir sann. Tilfelle c) viser situasjonen etter en transisjon når B var sann, og vi kan betrakte c) som en situasjon som etterfølger b).

Betingelser angis matematisk, som boolske uttrykk, eller fysisk, som eksempelvis sammenkopling av relekontakter eller elektroniske (logiske) kretselementer, eller som en kombinasjon. Noen eksempler:



Betingelsen for transisjon fra  
trinn 5 til 6:  $a \wedge (b \vee c)$



Betingelsen for transisjon fra  
trinn 5 til 6:  $\neg a \wedge b \wedge c$

Fig. 45 .

Vi har naturligvis behov for å uttrykke valg mellom alternative transisjonsveier. I figurene 22 og 31 uttrykkes dette ved at flere transisjonslinjer

kan føre ut fra en tilstand, hver med forskjellig betingelse som utløsende årsak. I den formen vi beskriver her uttrykkes dette litt annerledes: Ut fra selve trinnet fører bare 1 linje, men den forgrener seg til flere, hver med sin egen og forskjellig betingelses-strek på tvers av transisjonslinjen:

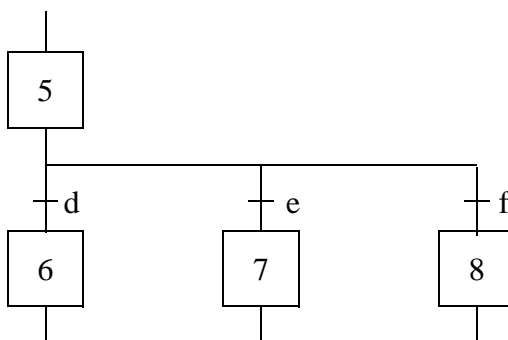


Fig. 46 .

I figuren over er d, e og f tre forskjellige betingelser, som ved oppfyllelse fører til hver sin alternative gren. Det vil være klart at før eller senere må  $d \vee e \vee f = 1$ , og de må også være *gjensidig utelukkende*.

I likhet med grafformen i figurene 22 og 31 er ett og bare ett trinn *aktivt* i de uttrykksformene som er vist hittil. I Grafcet-formen har vi imidlertid i tillegg også mulighet for å uttrykke *parallelitet*, dvs. to eller flere deler av diagrammet hvor prosessering foregår samtidig. Dette innebærer at flere enn ett trinn er aktivt samtidig. Dette uttrykkes på liknende måte som i figuren over, bare med den forskjell at den horisontale linjen som fører til alternativ forgrening i figuren over erstattes med en dobbel-linje, samt at vi trenger en betingelses-strek ovenfor, rett etter utløpet fra foregående trinn, mens det ikke skal være betingelses-strek mellom den horisontale dobbelt-linjen og de etterfølgende trinn-boksene (se nedenfor). Hvorfor?

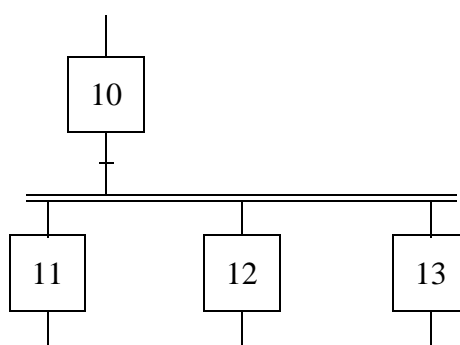


Fig. 47 .

Etter parallelle grener må grenene føres sammen igjen, der parallell operering slutter. Dette uttrykkes også med en horisontal dobbeltlinje. Samling av alternative grener uttrykkes tilsvarende, med horisontal enkeltlinje. Følgende figur er et eksempel som inneholder begge disse typene forgrening, samt samling igjen:

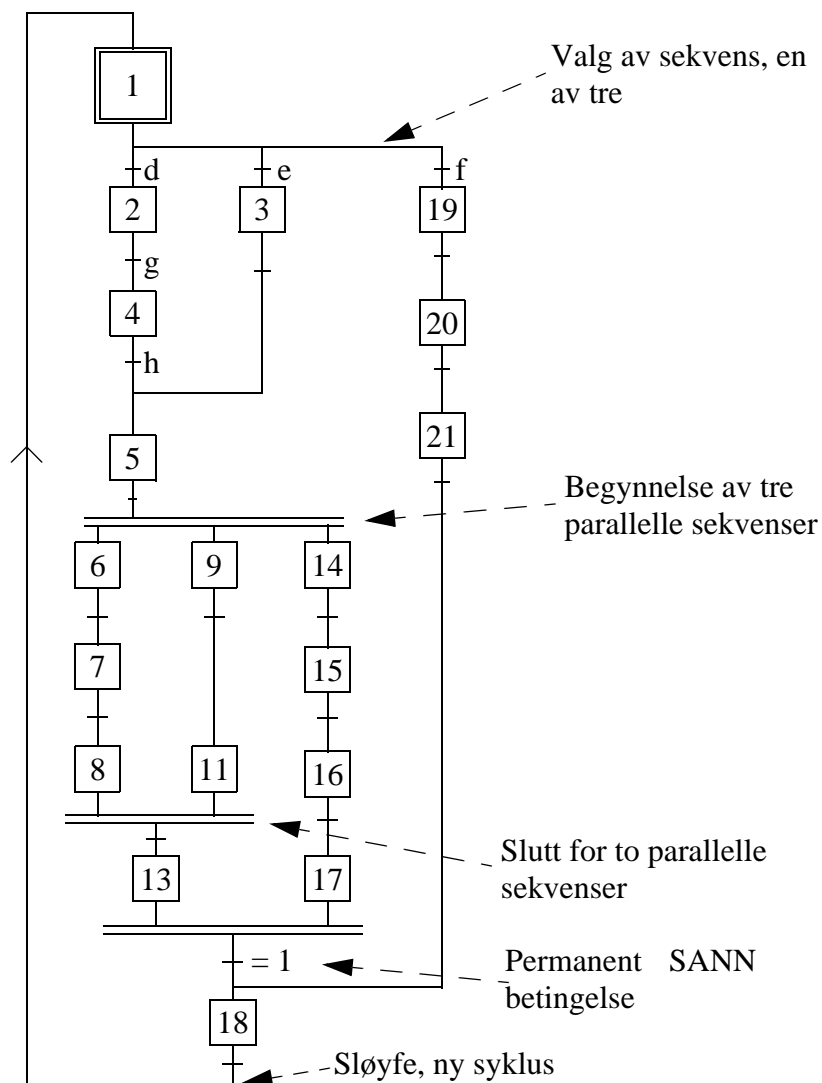


Fig. 48 Eksempel på sekvens-system

### 1.9.3 Detaljert beskrivelse av kommandoer

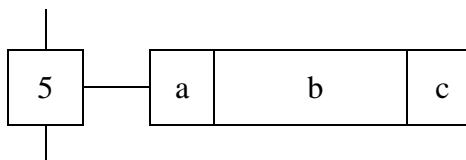


Fig. 49 Trinn med kommandoblokk

En kommando har generelt tre felt innenfor rektangelet. Feltene a og c utelates gjerne hvis de ikke brukes. Felt b skal være minst dobbelt så stort som a og b.

Felt b inneholder selve kommandoen. c er en etikett som kan brukes som referanse. Felt a kan inneholde bokstavsymboler for å vise behandlingen av binære signaler, som forklart nedenfor.

På side 55 ble vist forskjell mellom lagrede og ikke lagrede kommandoer. Varigheten av "ikke-lagret" kommando er forutsatt å være lik lengden av perioden vedkommende trinn er aktivt. I praksis vil mange ikke-lagrede kommandoer være begrenset i tid og kan også være forsinket i forhold til tidspunktet trinnet blir aktivt. Tilsvarende forhold gjelder for lagrede kommandoer.

Den korrekte sammenheng mellom varigheten av kommandoene og varigheten tilhørende trinn er aktivt uttrykkes ved å merke et detaljert kommandosymbol med et bokstavsymbol i felt a (se figur 49) med en av følgende bokstaver:

- S (Stored, lagret)
- R (Resett en lagret aksjon, *kun i IEC 1131-3*)
- D (Delayed, forsinket)
- L (time Limited, tidsbegrenset)
- C (Conditional, betinget)

Hvis L er svært kort, kan L erstattes med

- P (Pulse shaped, pulsformet)

Vi kan også bruke en kombinasjon av bokstavsymboler. Vanlig lese-rekkefølge svarer da til hvordan et binært signal fra vedkommende trinn behandles for å forme den ønskede kommando. Eksempelvis vil symbolet

DSL betyr at det binære signalet blir forsinket, lagret og så gitt en viss varighet. Virkemåten fremgår for øvrig best av en rekke eksempler vist i det følgende.

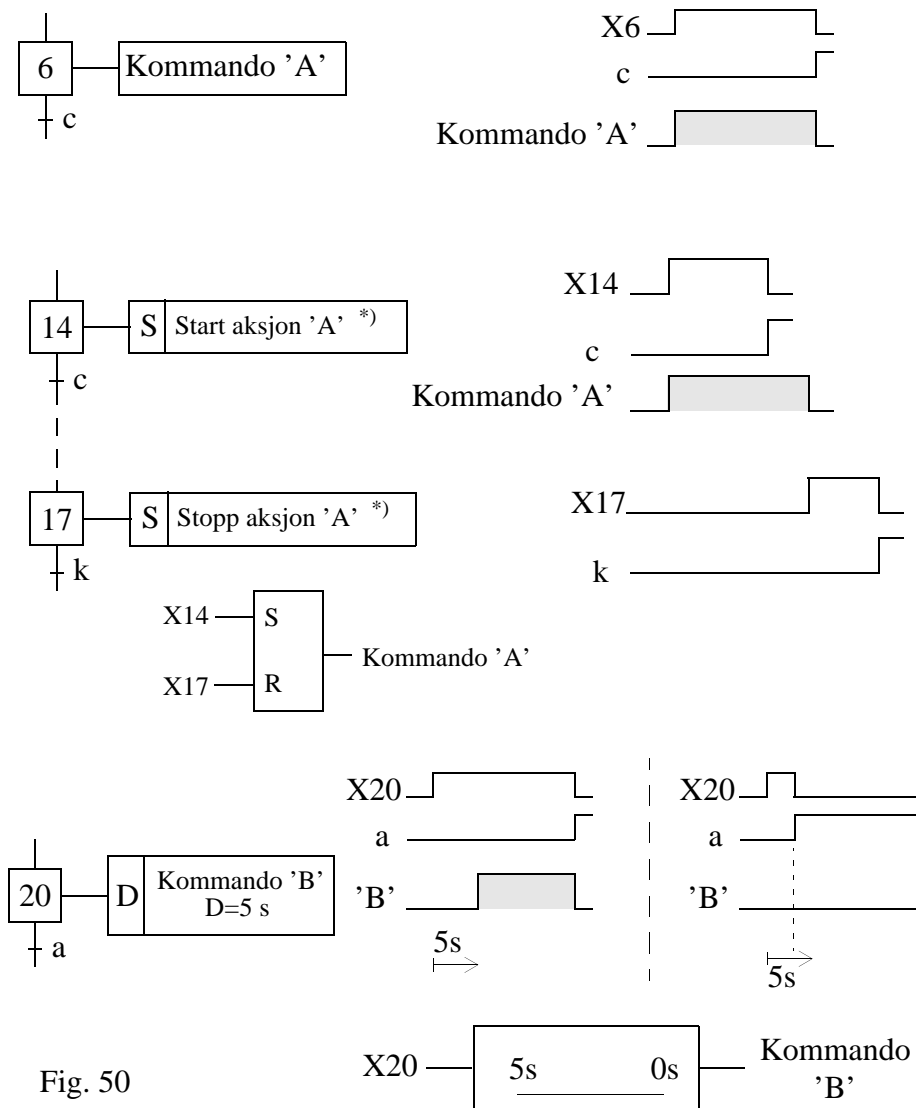


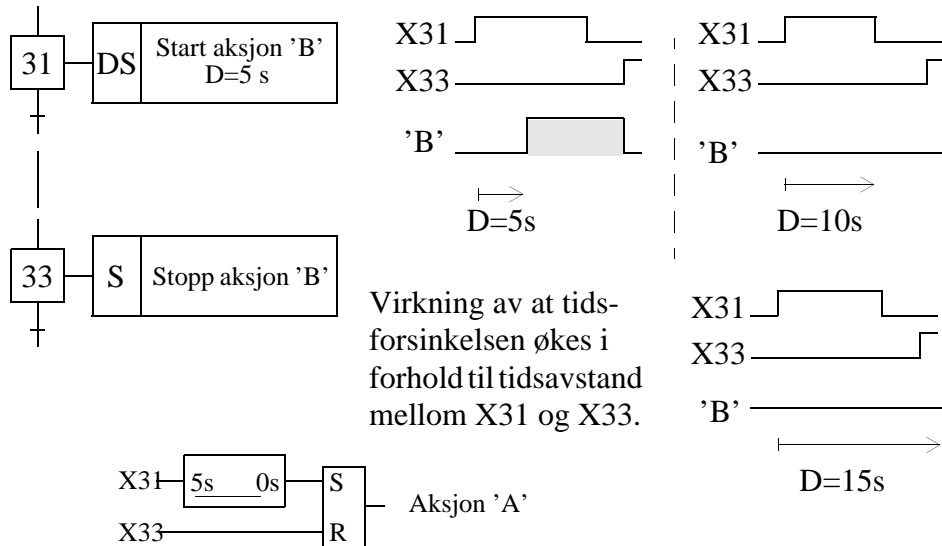
Fig. 50

\*) Med IEC1131-3 brukes for trinn 14 og 17 henholdsvis S: Aksjon 'A' og R: Aksjon 'A' for Start og Stopp.





Forsinket og lagret aksjon:



Lagret og tidsbegrenset aksjon:

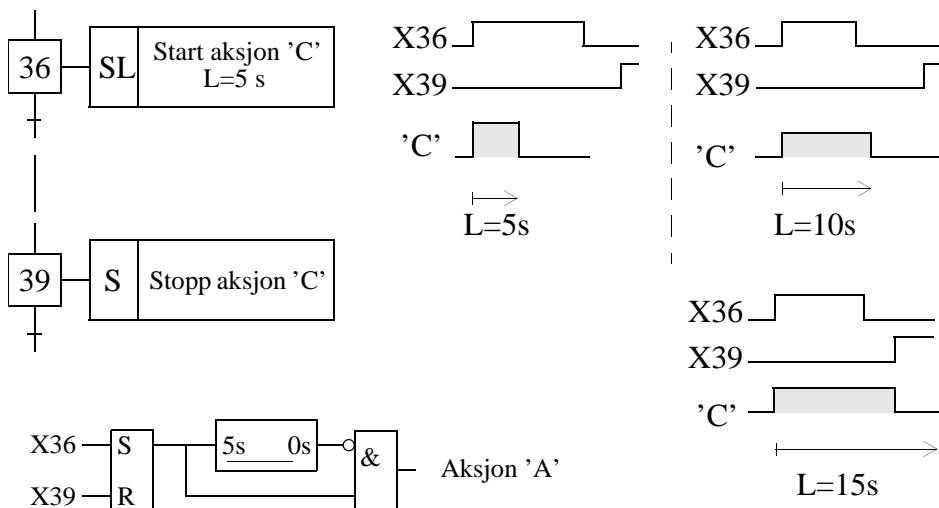


Fig. 52

En kommando kan sendes gjennom en *logisk aktiverende betingelse* før eller etter den angitte prosesseringen av flankesignalet (felt "a" i kommando-symbolet i Fig. 49 på side 60). Dette er særlig nyttig når kommandoen "lagres" (holding). Slik betingelse kan angis innenfor eller utenfor kommando-symbolet, avhengig av tilgjengelig tekstplass. Eksempler:

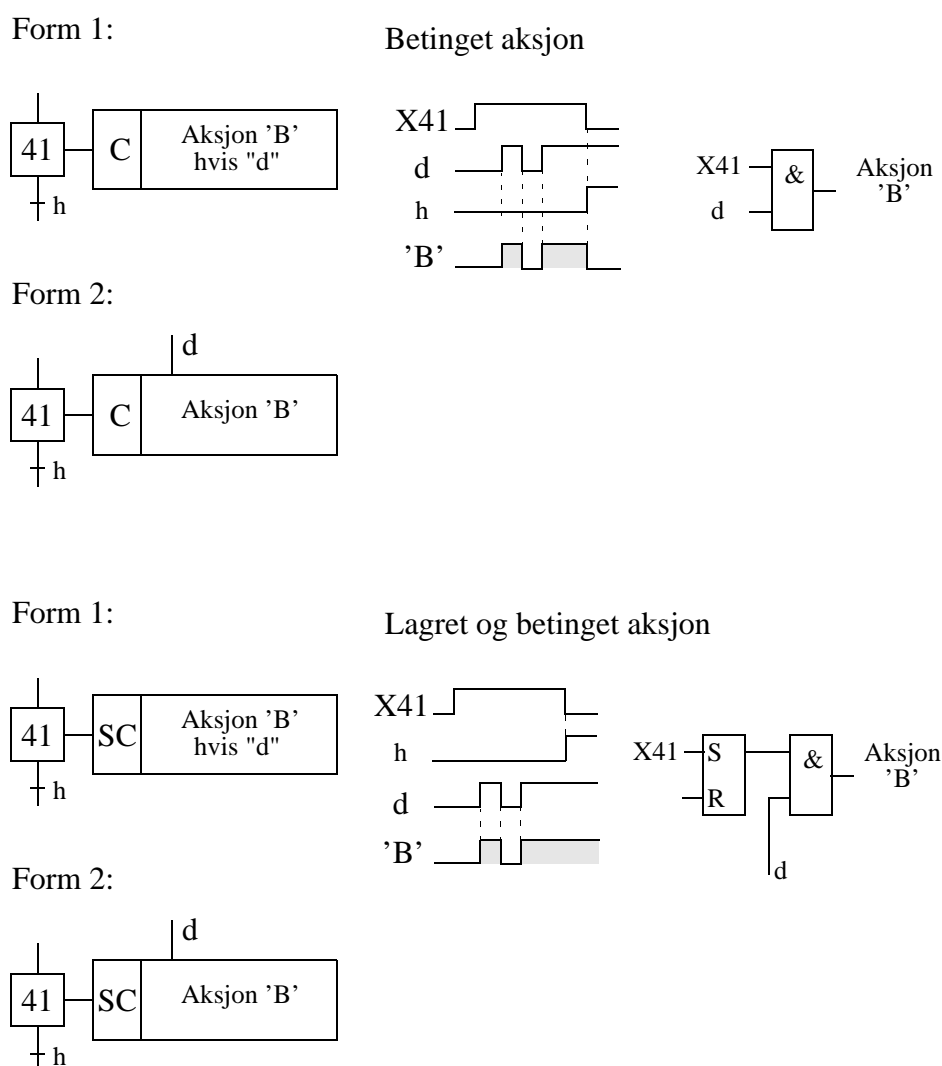


Fig. 53

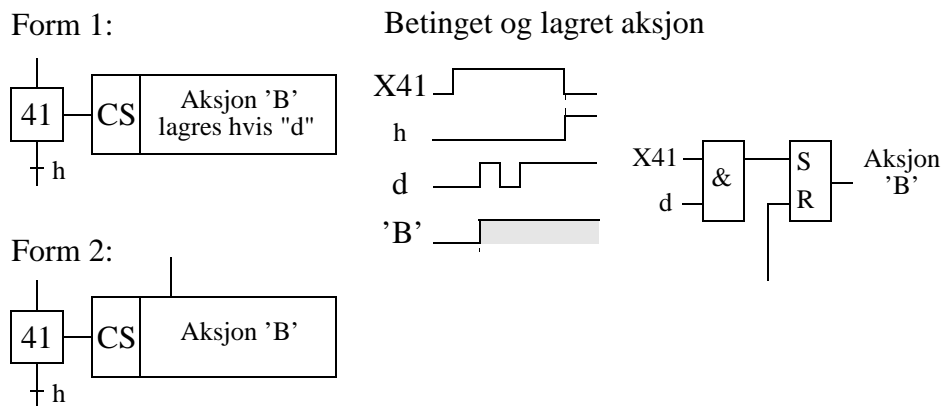


Fig. 54

### 1.9.4 Detaljert beskrivelse av transisjoner

Som omtalt innledningsvis i avsnitt 1.9.2, side 54, kan transisjoner angis bl.a. med tekst, som boolske uttrykk eller grafiske symboler. Vi skal nå utdype dette nærmere.

#### Tidsavhengighet

Vi kan ha behov for å uttrykke at en betingelse skal utløses en viss tid etter at en logisk variabel har endret seg. Mer presist: Hvis en logisk variabel **a** endrer seg fra logisk 0 til 1 og en tid etter tilbake til 0, kan dette fremstilles som en firkantpuls. *Virkningen* av denne pulsen kan benyttes som en betingelse, som eventuelt kan være forsinket i forhold til "inngangen" **a**. Forsinkelsen kan være i forkant eller bakkant av pulsen **a**, eller eventuelt ved begge pulsflanker. Dette sees lettest i en figur:

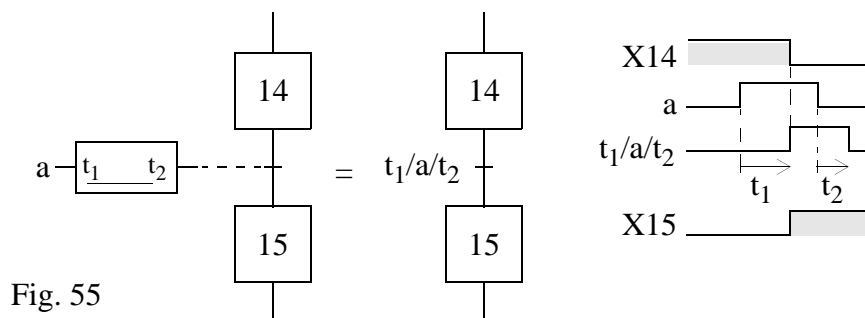


Fig. 55

I figur 55 er vist to alternative måter å angi en forsinket betingelse: enten ved hjelp av et logikksymbol for forsinkelse, eller ved den sammensatte beskrivelse  $t_1/a/t_2$ . Hvis en av tidsforsinkelsene  $t_1$  eller  $t_2$  er 0, sløyfes leddet. Dvs. hvis  $t_1=0$  men  $t_2 \neq 0$ , skriver vi helst  $a/t_2$ , og hvis bare  $t_2=0$ , uttrykkes tidsforsinkelsen som  $t_1/a$ . Hvis  $t_1$  eller  $t_2$  er konstante tall, skrives tallverdiene, altså ikke som symbolene  $t_1$  og  $t_2$ .

I figur 55 ser vi at  $t_1/a/t_2$ , dvs. betingelsen for transisjon fra trinn 14 til 15, inntreffer forsinket  $t_1$  etter at  $a$  går 0 til 1. Transisjonen innebærer at X14 opphører og X15 begynner, slik det ble forklart i avsnitt 1.9.3.

Helt i overensstemmelse med reglene vist foran kan man eksempelvis angi at varigheten av trinn 27, dvs. Aksjon 'B', skal være 4 sekunder på følgende måte:

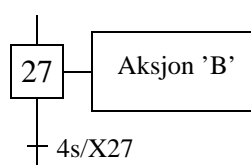


Fig. 56

Notasjon for logiske tilstander og tilstandsoverganger:

Notasjon	Sann ved	Signalform
$c$	$c=1$	
$\neg c, \bar{c}$	$c=0$	
$\uparrow c$	positiv flanke av $c$	
$\downarrow c$	negativ flanke av $c$	

### 1.9.5 Gjenbruk av sekvens

En sekvens av trinn, som opptrer flere steder, kan representeres hvert sted med ett enkelt trinn-symbol av generell type, som henviser til en detaljert fremstilling av disse trinnene. Eksempel:

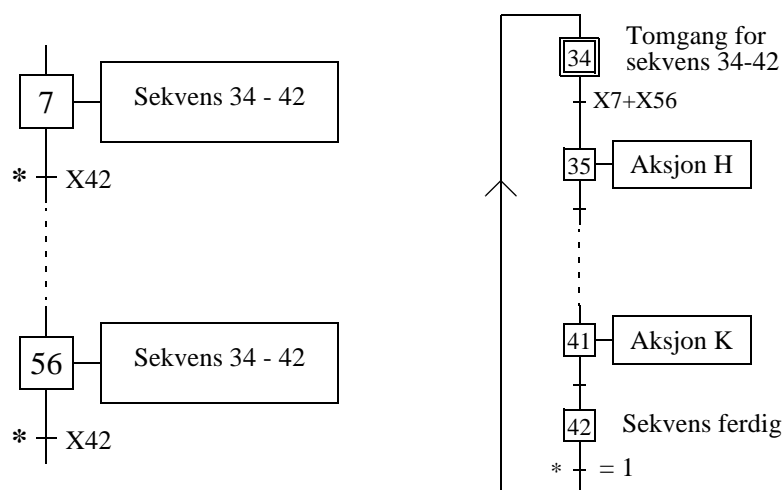


Fig. 57

Stjernesymbolet (\*) brukes til å markere transisjoner, plassert i forskjellige grafer, som skal nullsettes samtidig.

Egentlig er dette en subrutine, og fremstillingen som en egen sekvens (aktivitet eller prosess) slik høyre side av figur 57 viser er ikke helt god: Den forutsetter at det er en subrutine, og som sådan starter utføringen idet subrutinen kalles, og utføringen stopper idet det returneres til kallende sekvens. I ovennevnte figur er subrutinen fremstilt som en egen sekvens eller prosess, og hvis dette skal virke korrekt, *må* "sekvensen" stå i ro i trinn 34 når den ikke er kalt. Dessuten er betingelsen ut fra trinn 34 bundet: Her *må* stå den logiske eller-funksjonen av de mulige kallene. Det er med andre ord spesielle bindinger som ikke eksisterer i "normale" sekvenser. Denne måten tas imidlertid med her, fordi den er med i standarden, og den virker helt fint, hvis reglene følges fullt ut. Ikke sett en vilkårlig betingelse etter trinn 34, imidlertid!

### 1.9.6 Flere nivåer

Et skjema kan gjerne utformes i flere nivåer med forskjellig detaljeringsgrad. Eksempel:

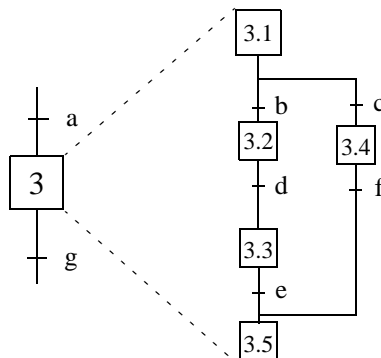


Fig. 58

I figur 58 er sekvensen til høyre en detaljering av trinn 3. Derfor er det også naturlig å la nummeret til hovedtrinnet komme frem i trinn-numrene, altså som under-trinn til trinn 3. For øvrig er denne fremstillingsformen bedre enn den som ble brukt i figur 57: detaljskjemaet kan godt oppfattes som en subrutine og er ikke beheftet med de problemene som det ble advart mot ved figur 57.

### 1.9.7 Eksempler

#### Start/stopp av stor maskin

Noen maskiner må startes gjennom en sekvens av enkelt-aksjoner. Dette kan også gjelde stopp. Mange maskintyper kommer under denne kategorien. Noen få typiske eksempler: Store dieselmotorer, elektriske sleperingsmotorer, vannturbiner med elektriske generatorer som må fases inn mot nettet. I de trinnvise oppstart- og stopp-prosedyrene inngår gjerne **forrigling**, som betyr at vi går videre til neste trinn først etter at visse kriterier er "bevist" oppfylt. Kriteriene er ofte bestemt av sikkerhetskrav.

En slik overordnet sekvens kan fremstilles slik:

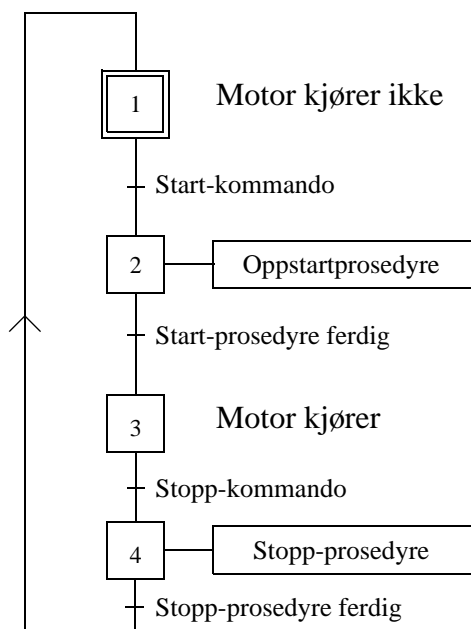


Fig. 59

Hver av aksjonene "Oppstartprosedyre" og "Stopp-prosedyre" kan så videreutvikles mer detaljert, etter samme prinsipp som i figur 58. Standarden IEC-848 viser et eksempel på dette, men vi tar det ikke med her.

### Automatisk veiing og blanding

Systemet er vist skjematisk i Fig. 60 på side 70 og består av to pulverbeholdere A og B plassert over en vekt C. Videre et transportbelte hvor det ankommer "briketter", samt en roterende blander. Beholderne A og B er utstyrt med hver sin mateinnretning MA og MB på undersiden, slik at pulver kan slippes ned på vekten i passende hastighet. Brikettene ankommer på transportbåndet en etter en og "tippes" over kanten og ned i blanderen, hvor de blir knust og blandet med pulveret. figur 60 viser disse enhetene.

En operasjonsyklus foregår slik:

Etter trykk på en startknapp starter oppveiing av først pulver A opp til en mengde  $p_A$ , deretter oppveies pulver B inntil vekten viser  $p_B$ . Etter dette

tømmes vekt-innholdet ned i blanderen ved hjelp av materen MC. Samtidig med at oppveiingen begynner, starter transportbåndet og en spesiell mater som slipper briketter ned på båndet. Hastighetsforskjellen mellom transportbåndet og brikettmateren er slik at brikettene blir liggende adskilt, en og en.

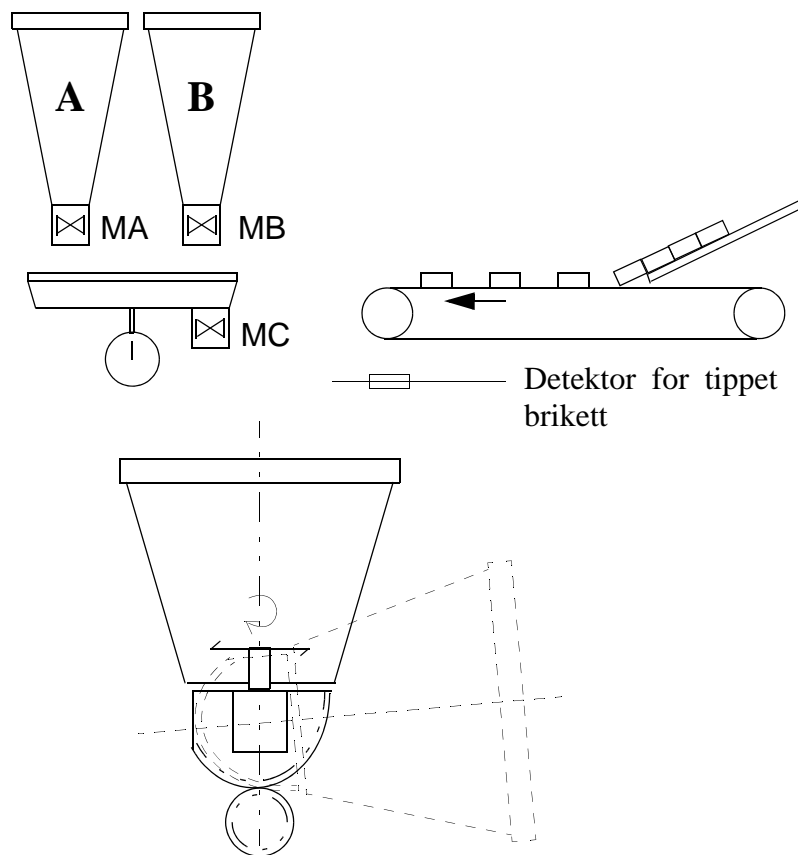


Fig. 60

Idet en brikett tipper ned i blanderen registreres dette av en detektor. Etter at foreskrevet antall briketter og oppmålt pulver er kommet ned i blanderen starter denne. Etter en viss blandetid tippes blanderen som vist stiplet, slik at den tømmes, og etter en viss tid i tippet stilling anses blanderen tom. Da stoppes motoren som roterer blanderen, og blanderen reises til opprettet



stilling. Når blanderen er kommet opp, stoppes syklusen.

Dette kan fremstilles som i figur 61:

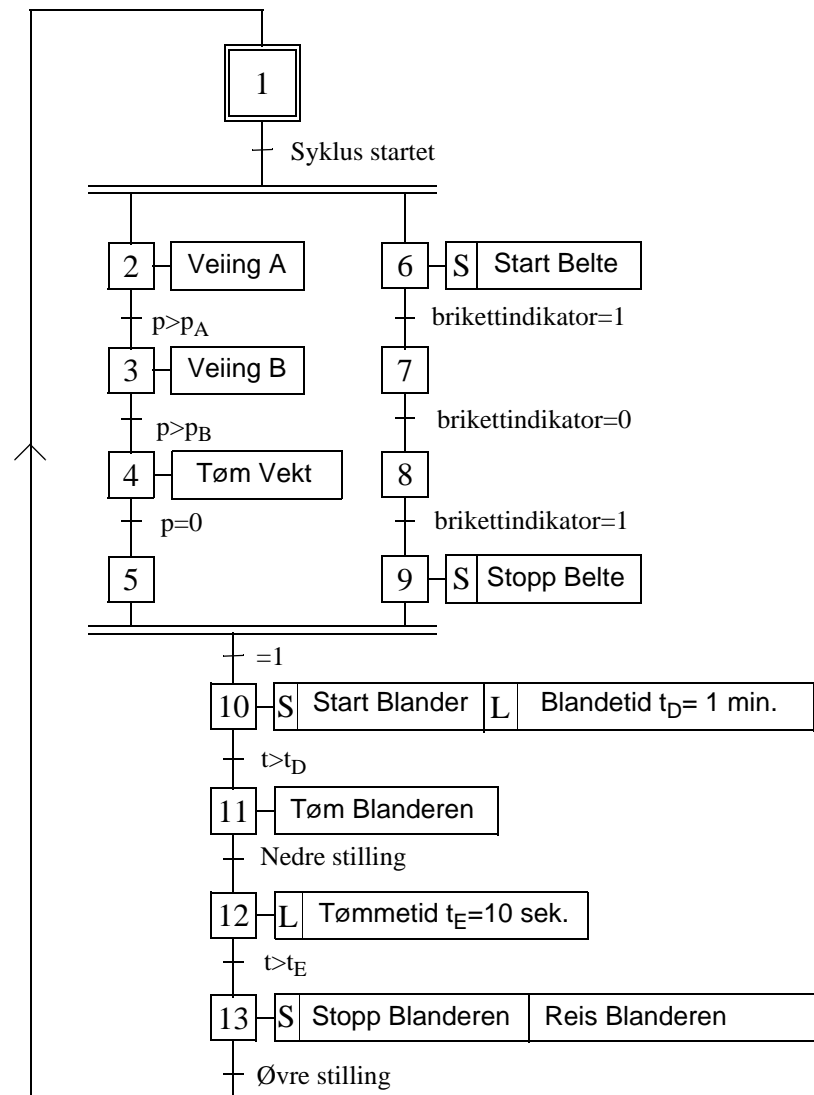


Fig. 61 Veie- og blandeprosess

## 1.10 Metoder og utstyr for realisering av logikkstyring

### 1.10.1 Oversikt

Logikkstyring kan realiseres med én av to forskjellige prinsipper:

- Trådbundet, og
- Programmert eller programmerbart

I “gamle dager” var alle styresystemer “trådbundet”. Det vil si at logikksystemet var oppbygget basert på et antall logiske elementer som var fast forbundet, på en slik måte at selve forbindelsene var bestemmende for den logiske oppførsel av systemet. Et enkelt eksempel på dette er sammenknytning mellom noen enkle logiske funksjonsblokker:

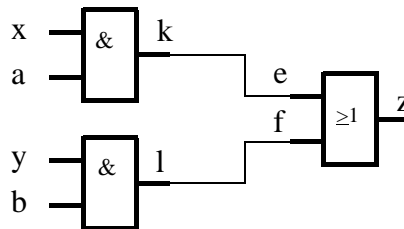


Fig. 62 . Trådbundet styring med 3 funksjonsblokker

Her har vi tre funksjonsblokker: To “OG”-funksjoner, og én “ELLER”-funksjon. Hvis de sammenkoples som vist, ved at inngangene **e** og **f** til ELLER-blokken koples til utgangene **k** og **l** fra OG-blokkene, oppnås at “utgangsvariabelen”  $z = (x \& a) \mid (y \& b)$ .

Samlet er dette en *multiplekser*: Hvis **a** og **b** er to gjensidig utelukkende styresignaler, vil disse velge ut én av inngangssignalene **x** eller **y** og presentere den valgte ut på utgangen **z**. Sammen med de elementære OG- og ELLER-blokkene er ledningene fra **k** til **e** og fra **l** til **f** bestemmende for den samlede oppførselen av dette “systemet”: Multiplekserfunksjonen **z** er fast og “trådbundet”, den er knyttet til de fast oppkoblede ledningstrådene. Dette prinsippet kan utbygges til store systemer, og dette var tidligere eneste måten man hadde mulighet for.

Det store problemet med slike løsninger var at endringer var svært komplisert da det medførte forandring av kabelopplegg, og i den sammenheng vanligvis også ombygging av elementære funksjonsblokker.

Denne store begrensningen ble man ikke kvitt før man kunne bruke *programmerbare* innretninger, og denne mulighet fikk man først da man tok i bruk programmerbare prosessorer, dvs. datamaskin-teknologi, og dette begynte å bli vanlig fra midten av 1960-årene av. Fremdeles, etter overgangen til nytt århundre, benyttes trådbundet styring, men nå bare i helt spesielle tilfeller, til realisering av små, begrensede, faste og veldefinerte styringsbehov. Ett av de siste områdene som tok i bruk programmerbar styring var til biler, men her benyttes fremdeles trådbundet styring i stor utstrekning.

En type systemer som fremdeles er basert på trådbundet styring, og som vel alltid vil være det, er datamaskiner internt: En datamaskin er realisert med kretskort, som hvert består av et antall integrerte kretser og andre elektronikkomponenter, fast sammenkoplet gjennom kretskortmønsteret. Kretskortet kan bare i begrenset grad endres til ny oppførsel. Det benyttes altså fremdeles trådbundet teknologi, men nå helst begrenset til lavere abstraksjonsnivå, til å realisere komplekse byggelementer som er laget programmerbare slik at de tilsammen utgjør et programmerbart styresystem.

### Logikkfunksjons-elementer

I figur 62 foran ble brukt to typer byggelementer uten nærmere presentasjon: OG og ELLER. Det er lett og naturlig å tenke seg disse som elementære integrertkrets-brikker, eller deler av slike. Før vi fikk integrerte kretser, eller i alle fall før halvlederteknologien kom i vanlig bruk, var releer den eneste komponenttype man hadde som byggelementer til logiske styringer. Releer blir ikke nå lengre mye brukt til realisering av logikkfunksjoner. Vi skal likevel ta med en oversikt over denne komponenttype, av to grunner:

- Releer blir fremdeles brukt til svært enkle formål, der digitale styringsenheter ville være "å skyte spurver med kanoner", samt til å styre store strømstyrker, som eksempelvis å slå på lyskasterne på en bil, styrt av en liten bryter ved førersetet over tynne kabler, eller motorer. I sistnevnte tilfelle kalles releet gjerne en *kontaktor*.
- Releer danner konseptuell basis for en måte å programmere PLS<sup>2</sup> på, s.k. stigediagrammer. Se Fig. 66 på side 77 og omtale side 81.

---

2. PLS = Programmerbar Logisk Styring: se forklaring side 76.

### 1.10.2 Elektromekaniske releer

Et elektromekanisk rele består av en elektromagnet, et jernanker som kan tiltrekkes av elektromagneten, og av ett eller flere sett kontaktfjærer som legges om når ankeret opereres av elektromagneten. Prinsipp og skjematisk:

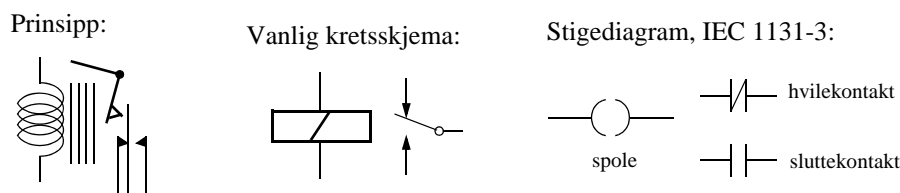


Fig. 63 Elektromekanisk rele

Et lite utvalg releer:

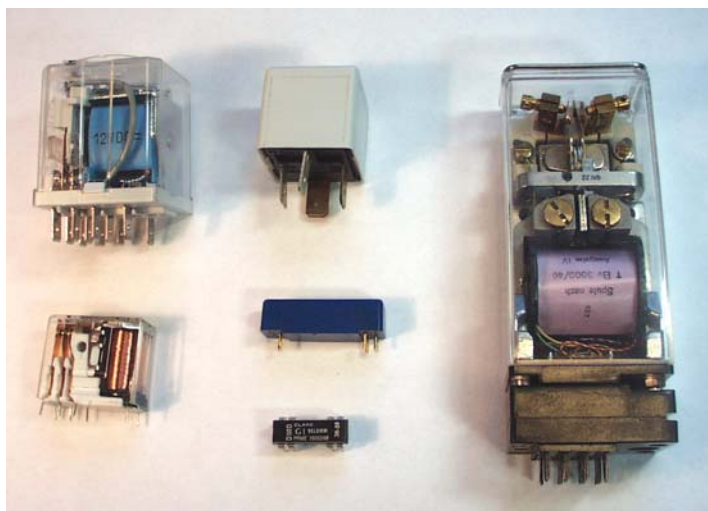


Fig. 64 Noen eksempler på releer

Venstre kolonne, ovenfra: “General purpose”, Tre venderfjærsett med kontakter som tåler 10A ved 250 V AC. Spole: 12 V DC.

Miniatyr, for montering på kretskort: To venderfjærsett 5 A 250 V AC, Spole 24 V DC.

Midtre kolonne: Enkelt rele med ett slutte-fjærsett. Spole: 24 V DC.  
 Reedrele for kretskort (fra ca. 1980). Enkel sluttekontakt.  
 Reedrele for kretskort (fra slutten av 1990-tallet). Enkel sluttekontakt.  
 Helt til høyre: Polarisert rele med stor følsomhet. Enkel venderfjærsett.  
 (Siemens, konstruert på 1940-tallet).

figur 63 viser to symbolstandarder for relekontakter og -spoler.  
 “Hvilekontakt” er en kontakt som danner forbindelse i strømløs tilstand og som bryter forbindelsen når relespolen energiseres. Motsatt for sluttekontakt, hvor forbindelse dannes når releet slår til, energiseres. En venderkontakt er en kombinasjon av en hvilekontakt og en sluttekontakt, og kontaktfjæren som enten danner forbindelse mot hvilefjæren eller slutfjæren kalles venderfjær. Venderkontakten finnes i to utgaver: break-before-make og make-before-break, som vist i figur 65. Fjæren som beveges av ankeret kalles arbeidsfjær, og for både hvilekontakt, sluttekontakt og break-before-make -typen av venderkontakt er det venderfjæren som er arbeidsfjær, mens det er slutfjæren som er arbeidsfjær ved make-before-break.

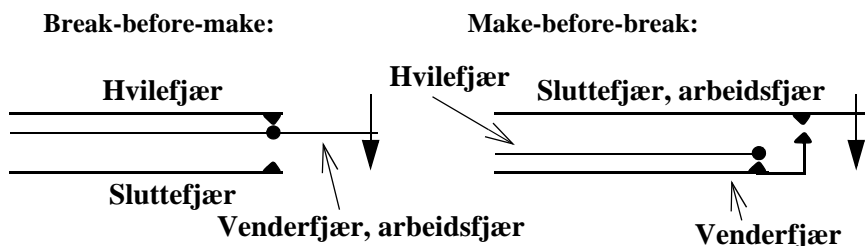


Fig. 65 To venderfjær-prinsipper

Reledigrammer viser **alltid** relekontaktene slik de er når spolen er strømløs.

Man realiserte ganske store systemer basert på releer. Typiske eksempler på slike store systemer som fikk stor utbredelse var de automatiske telefonsentralene. Systemene ble beskrevet og dokumentert med kretsskjemaer, og det var vanskelig å praktisere hierarkisk systembeskrivelse, slik at skjemaene ble store og lite oversiktlige. Likevel var systemkunnskapen stor hos de som arbeidet med slike systemer, inkludert montører og driftspersonell. Disse var vel vant med å lese kretsskjemaer, forstå dem til bunns og i

stand til å gjøre endringer, som da medførte endring av kabling.

I midten av 1960-årene begynte man å ta i bruk datamaskiner til styring av både kontinuerlige og logiske prosesser. Siden dette var helt nytt og man ikke hadde hverken hensiktsmessige programmeringsverktøy eller designverktøy, ble dette rene forskningsprosjekter til å begynne med. Etter at et slikt styresystem var ferdig utviklet og satt i drift, var det fremdeles meget tungvint å endre på, for man måtte da bruke programmerere, og helst de samme som hadde utviklet systemet. Man innså fort at det var behov for metodikk og utstyr som kombinerte datateknikkens potensielle fleksibilitet med mulighet til at montører og prosessingeniører selv raskt kunne gjøre alle nødvendige endringer, slik som de hadde kunnet gjøre med relebaserte systemer. Dette ga på slutten av 1960-tallet opphav til PLC, Programmable Logical Controller, eller på norsk PLS, Programmerbar Logisk Styring.

Siden det var sterkt ønskelig at montører og driftspersonell selv skulle kunne gjøre endringer i et system, ble det utviklet programmeringsverktøy hvor logikken ble uttrykt ved hjelp av rele-kretsskjemaer, selv om realiseringen var med elektroniske blokker og programmerbare prosessorer. Man *simulerer* altså relekoplingen. Et slikt skjema består av mange relekontakter i sammenkopling slik at de tilstrebeta logiske operasjoner oppnås. For eksempel vil to seriekoblede sluttekontakter kunne gi en sluttet strømkrets hvis begge kontaktene er operert, dvs. de danner en OG-funksjon. En parallellkopling av to kontakter skaper tilsvarende en ELLER-funksjon. Hvis en slik sammenkopling innkopler strøm til en relespole, vil releet operere og bevirke videre koplinger. Den ene enden av en slik seriekopling må da tilknyttes en spenningskilde, og den andre til kraftforsynings 0-punkt. Når vi så har en rekke slike kretser, koplet mellom den samme spenningskildens poler, blir dette av utseende som en "stige", som illustrert i figur 66. Se også Fig. 67 på side 78, som viser et komplett styresystem, av moderat størrelse, og hvor logikkfunksjonene er realisert med en blanding av transistorer og releer.

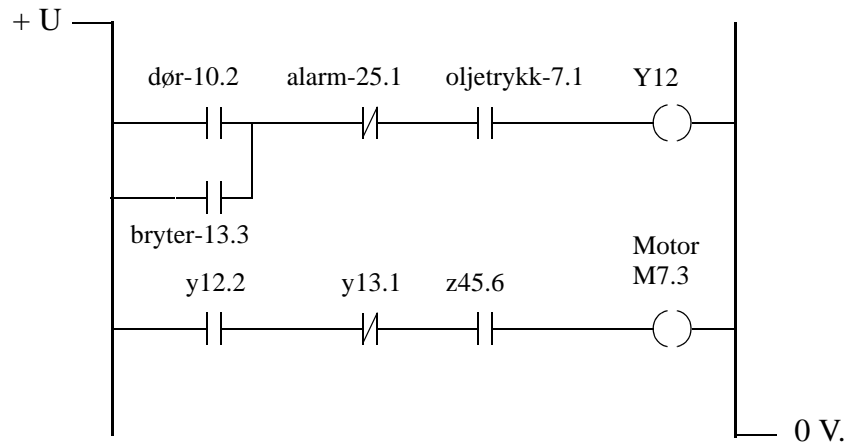
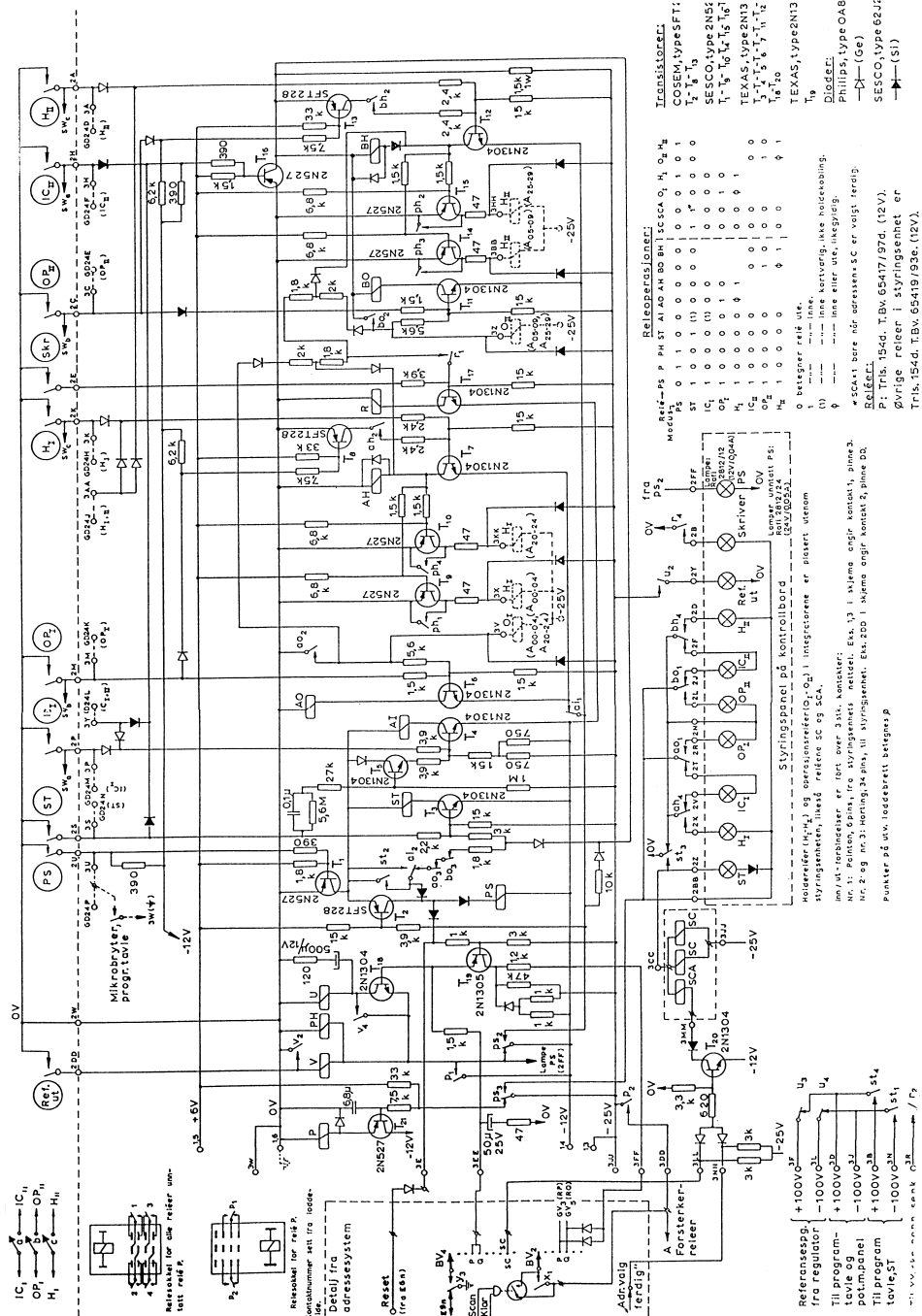


Fig. 66 Eksempel på stigediagram

En grunnleggende vanskelighet, som oftest blir oversett, er forskjellen mellom parallelle, logisk kombinatoriske problemer og metoder på den ene side, og sekvensielle, tilstands-betingede problemer og metoder på den andre. Fordi denne forskjell sjelden blir berørt, vil vi behandle dette spesielt i kapittelet Parallele og sekvensielle funksjoner, side 79.

## 1.10 Metoder og utstyr for realisering av logikkstyring





### 1.10.3 Parallelle og sekvensielle funksjoner

Releer og logikkmoduler er naturlig parallelle, distribuerte. Releer og deres kontakter ett sted i et system kan gjerne operere samtidig med releer andre steder i systemet. En prosessor, derimot, gjør jobben ved å utføre en rekke enkeltoperasjoner, én etter en. Dette har som konsekvens at det er enklest hvis metoden direkte tilsvarer problemets art:

- Releer og logikkmoduler samsvarer med parallelle og kombinatoriske problemer, og realiseringen av disse problemer blir ganske direkte. Det gir derimot problemer å realisere sekvenser og tilstandsbetingede funksjoner. Disse må realiseres med “simulering” av sekvenser, enten ved tilbakekoplinger eller ved at kun en liten del av koplingen er “aktiv” til enhver tid. Den “aktive” delen forflyttes gjennom systemet, mens resten av systemet er i en “ventetilstand”. Et enkelt eksempel på dette blir vist senere.
- Prosessorer, datamaskiner, utfører jobben sekvensielt og bearbeider problemet som en algoritme som består av en rekke enkelt-trinn. Etter sin natur er det derfor i prinsippet enkelt å realisere sekvens-problemer. Parallelle systemer, derimot, må simuleres i datasystemet. Et typisk eksempel er sanntids operativsystem, som utnytter prosessorens høye hastighet til å betjene en rekke “prosesser” som sett utenfra arbeider i parallell.

Når vi i det etterfølgende skal ta for oss forskjellige programmeringsmetoder for PLS'er, vil jeg henvise til disse grunnleggende forskjeller og påpeke vansker som følger av dem.

### 1.10.4 Standarden IEC 1131-3<sup>3</sup>

Standarden IEC 1131, Programmable Controllers, fra IEC (International Electrotechnical Commission) kom i begynnelsen av 1990-årene og er den mest autoritative samlingen av regler og metoder for programmering og

---

3. God oversikt, mange eksempler og sunn vurdering finnes i flg. bok, som er brukt ved utarbeidelsen av dette kapittelet:

R.W. Lewis: Programming industrial control systems using IEC 1131-3.  
IEC Control Engineering series, vol. 50, 1995. ISBN 0-85296-827-2

bruk av PLS. Selv om ikke alle PLS-fabrikat følger standarden helt, så er IEC 1131 den standarden som alle PLS bør vurderes mot. Før IEC 1131 kom, hadde alle som laget PLS sin egen måte å programmere og å bruke dem på, og det var spesielt problematisk å knytte et bestemt fabrikat sammen med utstyr levert av andre, enten det nå var andre PLS, eller sensorer, pådragsorganer etc. IEC 1131 tar sikte mot et felles rammeverk som skal lette slike problemer.

IEC 1131 er inndelt i fem deler:

- Part 1: General information
- Part 2: Equipment requirements and tests
- Part 3: Programming languages
- Part 4: User guidelines
- Part 5: Messaging service specification

Her vil vi bare ta for oss del 3, som altså omhandler programmerings-"språk", dvs. metoder for hvordan en PLS programmeres. Første revisjon av IEC 1131-3 ble publisert i 1993 og er gjeldende versjon.

IEC 1131-3 omfatter fem forskjellige programmerings-"språk", hvorav to er tekstbasert og tre er grafiske:

- Strukturert tekst (Structured text)
- Funksjonsblokkdiagram (Function Block Diagram)
- Stigediagram (Ladder diagram)
- Sekvensielle funksjonsdiagram (Sequential Function Chart)
- Instruksjonslister (Instruction List)

## Strukturert tekst (ST)

Dette er et høynivåspråk med syntaks som likner på det ordinære programmeringsspråket PASCAL men er også påvirket av ADA. Et eksempel:

```
IF SPEED1 > 100.0 THEN
    Flow_Rate := 50.0 + Offset_A1;
ELSE
    Flow_Rate := 100.0;
    Steam := ON;
END_IF;
```

## Funksjonsblokk-diagram (FBD)

Et grafisk språk for avbildning av signal- og dataflyt gjennom funksjonsblokker, som skal oppfattes som gjenbrukbare programelement:

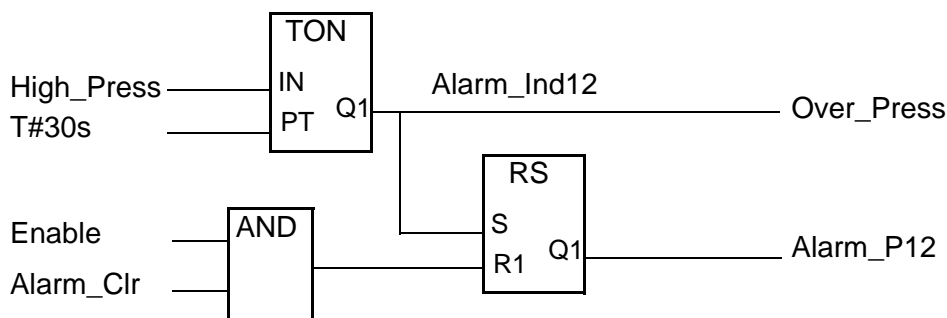


Fig. 68 Eksempel på FBD

## Stigediagram (LD)

Dette er et grafisk språk basert på stigediagram for releer, som beskrevet foran, blant annet rundt Fig. 66 på side 77. Releer kan fritt suppleres med logikk-funksjonselementer:

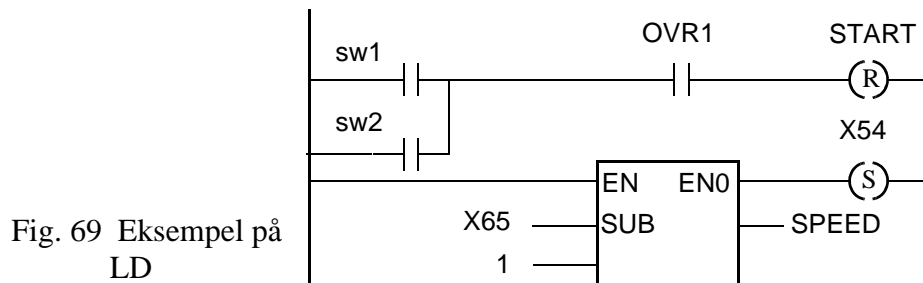


Fig. 69 Eksempel på LD

### Instruksjonsliste (IL)

Et lav-nivå språk som likner på assembly, og som er typisk for mange liknende språk som finnes i eksisterende PLS'er. Eksempel:

```
LD    R1
JMPC  RESET
LD    PRESS_1
ST    MAX_PRESS
RESET: LD    0
      ST    A_X43
```

### Sekvensielle funksjonsdiagram (SFC)

Dette grafiske språket er omtalt flere ganger allerede; det kan ses som en videreutvikling fra IEC 848 eller Grafcet. Det er velegnet for definering av tids- og hendelsesdrevne styresekvenser:

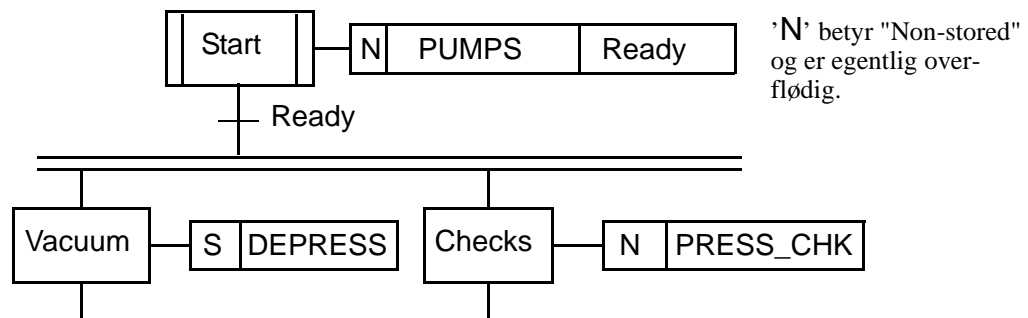


Fig. 70 Eksempel på SFC

Se *Lewis* (fotnote side 79) for mer detaljert beskrivelse.

### Vurdering av de fem metodene

Ved vurdering av hvilken metode og språk som egner seg for et gitt tilfelle, bør man ha klart for seg forskjellen mellom parallelle og sekvensielle funksjoner, som behandlet i et eget avsnitt foran, på side 79. Der nest er abstraksjonsnivået viktig. Hvis man velger en uhensiktsmessig metode eller språk, blir det en dårlig løsning.

To av språkene er tekstbasert: Strukturert tekst (ST) og Instruksjon-

sliste (IL). Begge er sekvensielle, så de egner seg bare for sekvensielle funksjoner, dvs. systemer som må beskrives av tilstander. Programmeringen blir som konvensjonell sekvensiell datamaskinprogrammering, og man har intet uttrykksmiddel for parallell operasjon. IL kan best sammenliknes med assemblykode og er på lavt abstraksjonsnivå. Begge språkene kan lett realisere sub-funksjoner (subrutiner).

De to språkene Funksjonsblokk-diagram (FBD) og Stigediagram (SD) er begge grafiske og parallelle. Begge egner seg best til beskrivelse av systemer som skal realiseres direkte i hardware, dvs. FBD som elektronikk på kretskort og SD med fysiske releer. Brukt for PLS, kan de også være hensiktsmessige til detaljer i et sekvensielt funksjonsdiagram (SFC), til beskrivelse av dannelsen av *betingelser* i et SFC. Siden de kan blandes, kan disse to språkene like greit behandles under ett (se bemerkning under avsnitt Stigediagram (LD), side 81).

Som hovedspråk til programmering av PLS er disse to språkene lite egnet. Her er noen avveininger:

SD har egentlig bare én fordel:

- Det er kjent for montører og har intuitiv virkemåte for små systemer.

Listen over ulemper er lang:

- Det er vanskelig å lage velstrukturerte program p.g.a. begrenset mulighet til subrutiner, programblokker og objekter:
  - Vanskelig å oppbygge program hierarkisk
  - Vanskelig gjenbruk
  - Begrensede muligheter til overføring av parametre mellom programblokker
- En del i stigediagram kan lett lese og sette kontakter og utganger hvor som helst i diagrammet. Dette vanskeliggjør innkapsling av data, som er viktig for god programkvalitet og vedlikeholdbar programvare.
- Problemer ved programutvikling med flere programmerere.
- Programmeringsenheter viser grafisk bare lite utsnitt av stigediagrammet. Det blir som å programmere gjennom et kikk-hull.
- Eksisterende PLS bruker lite standardiserte symboler og virkemidler.
- Dårlige muligheter til adressering og manipulering av datastrukturer.

- Dårlig datastruktur, tungvinte aritmetiske operasjoner.
- Tungvint realisering av sekvenser. Dermed er det begrensede muligheter til å bygge komplekse sekvenser.
- Liten styring med program-eksekvering.

Mange av problemene over kan omgås med diverse knep og design av PLS, men de kan aldri elimineres.

Her er en figur som viser hvordan sekvensforløp kan realiseres med releer. Som vi ser, er ikke sekvensforløpet særlig iøynefallende:

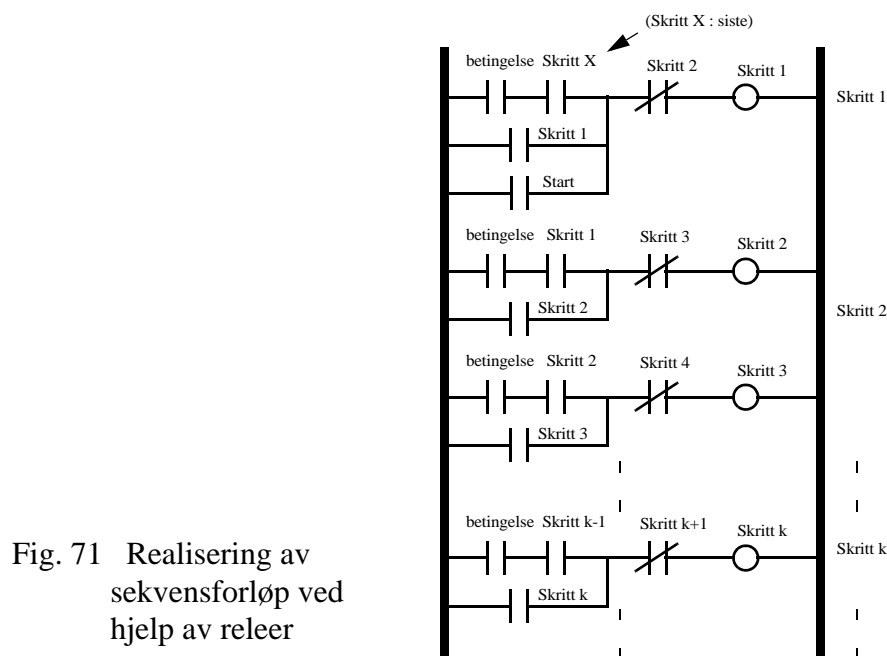


Fig. 71 Realisering av sekvensforløp ved hjelp av releer

Med alle begrensninger og innvendinger mot de fire nevnte språkene, står vi igjen med ett: **Sekvensielle funksjonsdiagram (SFC)**. Dette er egentlig det eneste som i dag kan anbefales som **generelt programmeringsspråk for PLS**. Det er grafisk og kan brukes hierarkisk slik at det er tjenlig både til oversikt i et stort system, og til detaljnivå. Det er både sekvensielt og parallelt, idet man kan legge inn parallelle deler etter behov. Funksjonsblokker og stigediagram kan gjerne benyttes til *detaljer* innen et SFC, spesielt for å uttrykke *betingelser*.

---

## KAPITTEL 2

# Pro세서orarkitektur

## 2.1 Innledning

I kap. 1 ble det antydnet hvordan logikkstyringer kan realiseres med logiske porter og vipper. I moderne systemer realiseres i praksis nesten all logikkstyring ved hjelp av datamaskiner.

I kjernen av enhver moderne datamaskin ligger det en *pro세서or*. Dette kapitlet gir en overordnet innføring i pro세서orarkitektur, og skal gi en konseptuell forståelse for hva som skjer i en slik pro세서or under utførelse av et program. Målet med dette er å avmystifisere datamaskinen, enten den er i form av en PC, en programmerbar logisk styring (PLS) eller et mikrokontrollerkort, og forberede leseren på å forholde seg til datamaskiner på “lavt nivå”. Dette er blant annet viktig ved programmering av mikrokontrollere, der programmeringen ofte omfatter operasjoner på bit- eller registernivå.

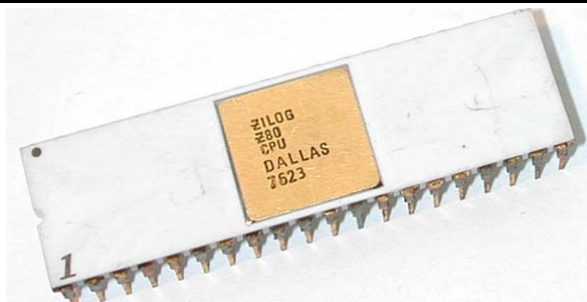
Dette kapitlet gir som sagt kun en meget overordnet behandling av temaet pro세서orarkitektur, og er ikke utfyllende på noen måte, og fokus er på enkle pro세서orer. For en grundigere behandling henvises leseren til annen faglitteratur, f.eks. er noe av stoffet i dette kapitlet hentet fra den klassiske boka John P. Hayes, *Computer Architecture and Organization*, 2nd Edition, McGraw-Hill (1988), som også finnes i nyere utgaver.

### 2.1.1 Historisk perspektiv

Datamaskinen er et relativt moderne fenomen, men den har gjennomgått en fantastisk teknologisk utvikling fra den første kjente mekaniske regnemaskinen så dagens lys i 1623 til vår tids maskiner basert på høyintegriert halvlederteknologi. Mange tror at vi står ved terskelen til en ny æra der kvantedatamaskiner (eng.: *Quantum Computers*) på nytt vil revolusjonere fagfeltet både når det gjelder maskinvarearkitektur, programmering og ikke minst regnekraft. Her skal vi imidlertid konsentrere oss om typiske trekk ved transistorbaserte pro세서orer, som i praksis er enerådende i dagens marked.

Datamaskinkonstruktører ble tidlig klar over nødvendigheten av et *modulært design*, dvs. maskiner satt sammen av distinkte deler som har forskjellige oppgaver. De mest åpenbare modulene i en datamaskin er illustrert nederst i Figur 2.2. Dette kapitlet fokuserer på *pro세서oren* (eng.: Central Processing Unit, CPU). Det er denne modulen som styrer aktiviteten i datamaskinen, inkludert programstyrt utføring av logiske operasjo-

ner og beregninger. De tidlige elektroniske prosessorene var satt sammen av enkeltkomponenter; først radiorør, senere transistorer og enda senere integrerte kretser (IC'er) på registernivå (se avsnitt 2.1.2). Etter hvert kom komplette prosessorløsninger integrert i én enkelt IC, et eksempel er vist i Figur 2.1. Disse ble referert til som *mikro-*



**Figur 2.1** Tidlig utgave Zilog Z80. Foto: Gennadiy Shvets/CC-BY-2.5.

*prosessorer*. I våre dager er alle prosessorer i praksis mikroprosessorer, så vi dropper gjerne forstavelsen *mikro* og refererer til dem bare som *prosessorer*.

### 2.1.2 Digitalteknikkens tre kompleksitetsnivåer

Ved design av eller kommunikasjon om digitale systemer, er det vanlig å skille mellom tre ulike kompleksitetsnivåer. Dette har å gjøre med det velkjente prinsippet *Divide-and-conquer*, som er mange ingeniørdisipliners viktigste grunnmetodikk: ved løsning av komplekse problemer deler en problemet opp i mindre deler, som deretter kan løses hver for seg og gir en enklere designprosess. Denne metodikken passer for menneskehjernen, som har en begrenset evne til oversikt og problemløsning.

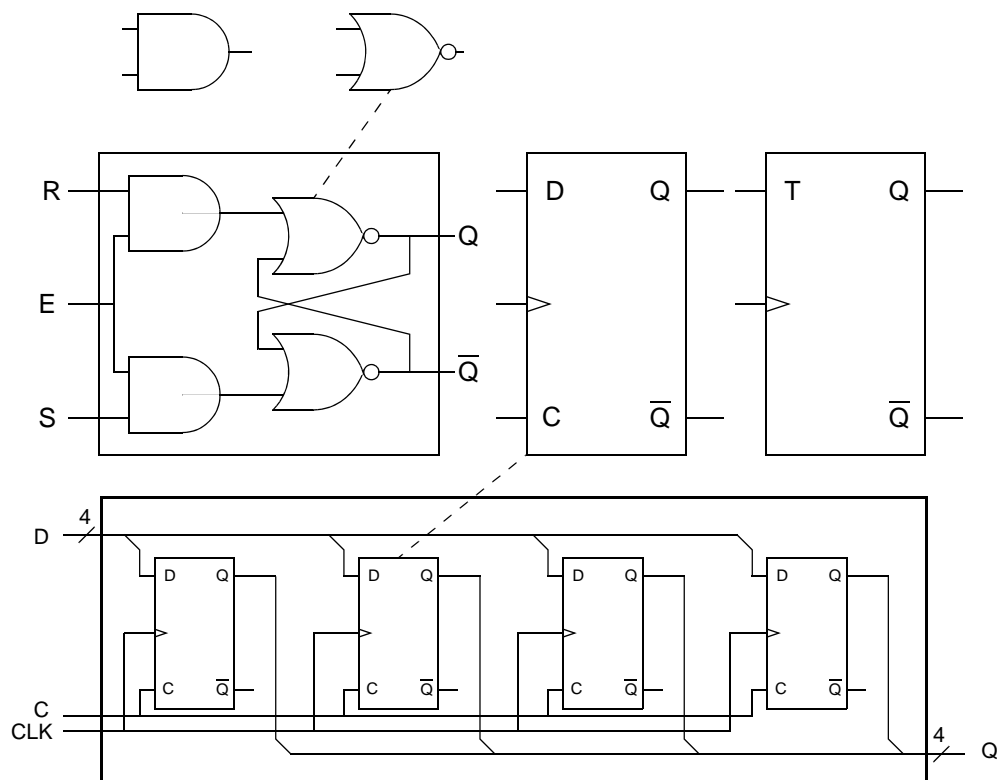
De tre nivåene i digitalteknikken kalles hhv. portnivå, registernivå og prosessornivå, og er illustrert i Figur 2.2. I dette kapitlet skal vi studere prosessorens sammensetning og virkemåte på registernivå. Som antydnet i Figur 2.2 er et klassisk register egentlig en samling vipper som kan inneholde et antall bits, men også andre kretser med tilsvarende kompleksitet regnes som registernivåkomponenter. Dette gjelder bl.a. kombinatoriske kretser som aritmetikkretser (f.eks. adderere), multipleksere og dekodere, og sekvensielle kretser som tellere og liknende.



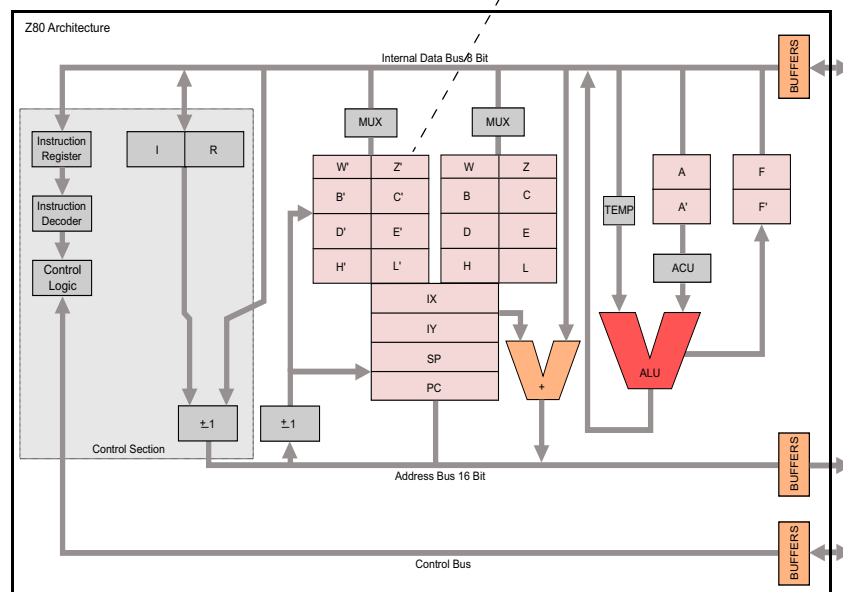
## Logiske porter

Vipper beskrives på *portnivå*. (En vippe kan anses som et ett-bits register.)

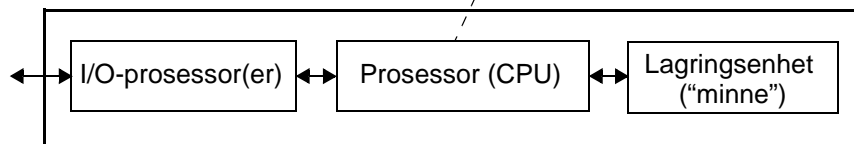
Eksempel på et 4-bits register. Også rent kombinatoriske kretser som multipleksere og adderere regnes som registernivåkomponenter.



Prosessorer beskrives på *registernivå*,...



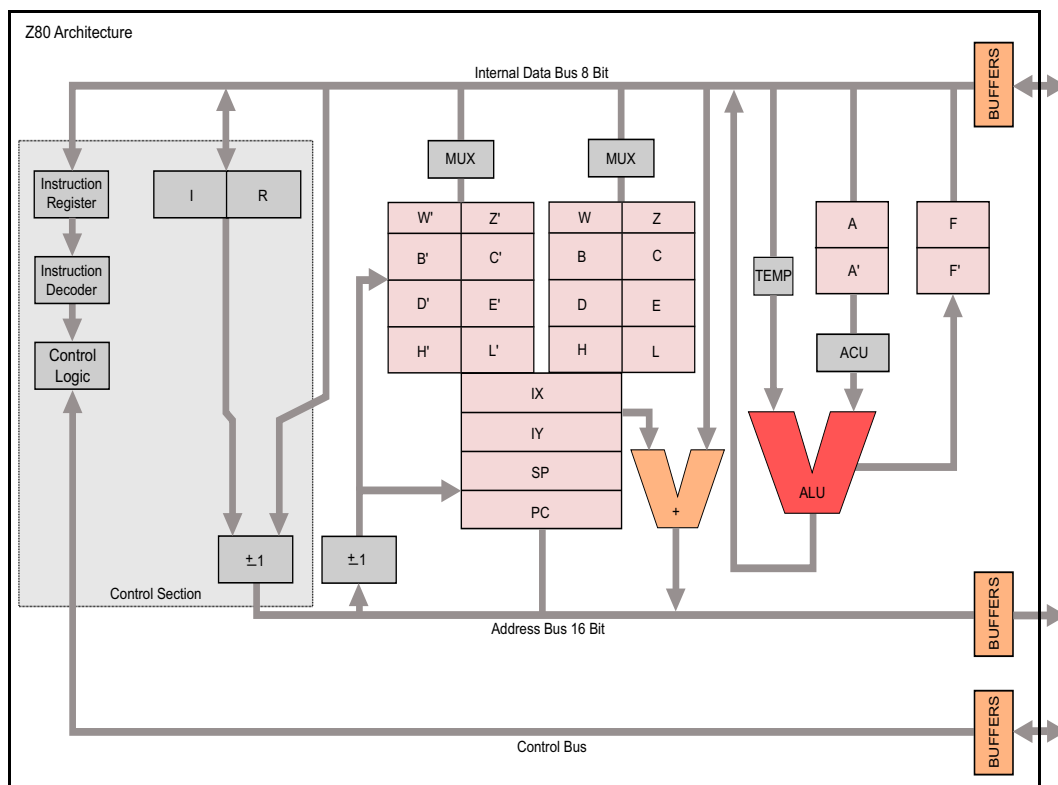
...mens hele datamaskiner beskrives på *prosessornivå*.



**Figur 2.2** Tre kompleksitetsnivåer: port-, register- og prosessornivå.  
Se Figur 2.3 for rettighetsdetaljer.

## 2.2 Z80: En klassisk prosessorarkitektur

Arkitekturen i prosessoren Z80 fra produsenten Zilog Corp. (Figur 2.1) har røtter tilbake til den eldre konkurrenten Intel 8080, og var den mest brukte mikroprosessen i 1970- og 80-årene. Disse prosessorene er også forløpere til den viktige x86-familien, som blant annet finnes (riktignok i utvidede og *mye* kraftigere utgaver) i alle “Intel-kompatible” PC’er og moderne Apple-maskiner. Figur 2.3 viser Z80’ens arkitektur på registernivå, og vi skal nå gjennomgå denne arkitekturen “register for register” og beskrive samspillet mellom de forskjellige delene.



**Figur 2.3** Z80-arkitekturen på registernivå.

Figuren er hentet fra [http://commons.wikimedia.org/wiki/File:Z80\\_arch.svg](http://commons.wikimedia.org/wiki/File:Z80_arch.svg) og gjengis under lisensen CC-BY-SA-3.0 (<http://creativecommons.org/licenses/by-sa/3.0/deed.en>).

### 2.2.1 Overordnet arkitekturbeskrivelse

Registernivåkomponentene eller modulene i Figur 2.3 er i hovedsak representert med rektangler. De to unntakene er de V-formede blokkene merket hhv. + og *ALU*, som er prosessorens “aller helligste”; det er her logiske og matematiske operasjoner utføres (mer om disse senere).

De rosa modulene merket med én eller to bokstaver er registre med ulike egenskaper, og de grå modulene merket *MUX* er multiplexsere som styrer hvilke av de to alternative registerblokkene som er aktiv (se avsnitt 2.2.2).

Blokken *Control Section* inneholder sekvensielle og kombinatoriske kretser for tolking av programinstruksjoner, noe som bl.a. omfatter styring av nær sagt alle de øvrige modulene i prosessoren (samt eksterne moduler via linjene merket *Control Bus*). Linjene for styring av de interne modulene (MUX, ALU osv.) er forøvrig ikke tegnet inn, men er implisitte, i Figur 2.3.

Til slutt har vi bussene, som i praksis består av parallelle ledere som overfører ett bit hver. Øverst har vi databussen, merket *Internal Data Bus 8 Bit* i figuren, som følgelig består av 8 digitale linjer. Alle linjene er koplet til alle registernivåkomponentene som er inntegnet med en forbindelse til bussen. Databussen er “transportmekanismen” for data mellom systemets deler. Lenger ned i figuren finner vi adressebussen (*Address Bus 16 Bit*), som tilsvarende består av 16 digitale overføringslinjer. Adressebussen overfører adresser (til instruksjoner eller data) mellom modulene. Endelig har vi styringsbussen (*Control Bus*), som består av et antall signallinjer for overføring av logiske styringssignaler. Alle bussene har et eksternt grensesnitt (t.h. i figuren) slik at prosessoren kan kommunisere med eksternt program- og dataminne osv.

## 2.2.2 Registrene

### *8-bits registre*

Z80 regnes for å være en 8-bits prosessor fordi de fleste operasjoner bare gjøres på 8 bits om gangen. I utgangspunktet er derfor prosessorens registre basert på 8-bits blokker. I Figur 2.3 er disse angitt med en enkelt bokstav, f.eks. *A*- og *F*-registrene øverst t.h.. Hvert av disse kan altså inneholde (“huske”) 8 bits, dvs. en *byte*.

### *16-bits registre*

De fleste 8-bitsregistrene er tegnet opp parvis, som f.eks. *W* og *Z*, *B* og *C*, osv.. Dette indikerer at de to registrene kan kombineres til et 16-bits register, og vi refererer da til disse som hhv. *WZ*- og *BC*-registeret.

De fire registrene *IX*, *IY*, *SP* og *PC* er rene 16-bits registre. Disse brukes primært til lagring av adresserelaterte data som må være lett tilgjengelig under programutførelsen. Av spesiell interesse her er *PC*-registeret (*Program Counter*, No.: *instruksjonspekeren*), som til enhver tid inneholder adressen til den neste programinstruksjonen som skal utføres. Den kalles teller (counter) nettopp fordi den “teller opp” instruksjonsadressen ettersom programutførelsen skrider frem.

### *Alternative registreblokker for hurtig oppgavesvitsjing*

De fleste av 8-bitsregistrene finnes i to alternative utgaver, der den ene er merket kun med en bokstav mens den andre i tillegg er merket med en tøddele (’), som *D*’ og *D*, *E*’ og *E* osv.. Dette representerer en ren duplisering, og de alternative registrene har identiske egenskaper.

Motivasjonen for denne dupliseringen er som følger. Registrene er prosessorens raskeste “arbeidsminne”, på den måten at logiske og matematiske operasjoner bare kan utføres på data som ligger i prosessorens registre. I situasjoner der prosessoren må skifte mellom to programmer, f.eks. hvis den får et *avbrudd*<sup>1</sup> (eng.: *interrupt*) som

---

1. Begrepet avbrudd behandles ikke videre her, men dekkes grundig i andre fag.

representerer en ytre hendelse og må reagere raskt på denne, må i utgangspunktet alle registres innhold lagres unna i et langsommere minne, som i Z80-tilfellet må ligge utenfor selve prosessoren, før det avbrytende programmets data kan lastes inn i registrene og utføringen starte. Når prosessoren senere skal fortsette til sin opprinnelige programutføring, må det avbrytende programmets registerdata skrives til det eksterne minnet og det opprinnelige programmets data leses inn igjen før utføringen starter. I noen sanntidssystemer kan slike avbrudd inntreffe flere tusen ganger hvert sekund, og det er derfor nødvendig at svitsjingen mellom de ulike oppgavene går så fort som mulig. Her kommer de dupliserte registrene inn: ved å ha to utgaver av hvert register, der den ene utgaven er dedikert til én oppgave og den andre utgaven til en annen oppgave, kan prosessoren skifte mellom de to oppgavene uten å måtte “mellomlagre” registerinnholdet. Alt den trenger å gjøre er å forholde seg til hhv. den ene eller den andre alternative registerblokken, noe som enkelt kan gjøres av programvaren ved å endre innholdet i en av prosessorens vipper som styrer *MUX*-ene øverst i figuren.

### 2.2.3 Aritmetisk-logisk enhet (ALU)

ALU'en er, som navnet tilsier, en digital krets som utfører operasjoner av aritmetisk (f.eks. addisjon og subtraksjon) eller logisk (AND, OR, NOT, XOR osv.) art. I enkle prosessorer er ALU'en en rent kombinatorisk krets, mens det i prinsippet ikke finnes noen øvre grense for hvor kompleks den kan være (f.eks. vil en flyttallsprosessor ha langt mer kompliserte ALU-ressurser enn en enkel 8-bits heltalsprosessor som Z80).

I Figur 2.3 er ALU'en tegnet som en stor rød, V-formet blokk. Den har to innganger á 8 bit, representert ved de to øvre endene av V'en. De to databytene som presenteres på disse inngangene blir gjenstand for ALU'ens aritmetiske eller logiske operasjon, og resultatet av operasjonen presenteres på utgangen (nederst på V'en). Hvilken operasjon som gjøres, styres via kontrollinjer (ikke inntegnet) fra blokken *Control Section*. Som vi ser av figuren, vil ALU'ens resultat bli sendt ut på databussen. Hvor resultatet lagres bestemmes igjen fra *Control Section* ved at denne enabler registratorer som skal lagre dataene og disables de øvrige gjennom dertil egnede kontrollinjer.

Den litt mindre V-formede blokken til venstre for ALU i figuren, er en “mini-ALU” som gjør kun én ting: den adderer dataene som ligger på databussens linjer til innholdet i et av 16-bits registerne IX eller IY, og presenterer resultatet på adressebussen (*Address Bus 16 Bit*). Denne operasjonen trengs i forbindelse med enkelte adresseringsmodi, noe som ligger utenfor dette kapitlets tema.

### 2.2.4 Styringsenheten (Control Section)

Nå som vi har en viss oversikt over de øvrige ressursene i prosessoren, tar vi en nærmere kikk på blokken *Control Section*. Som navnet antyder, og som beskrevet ovenfor, er dette en svært viktig del av prosessoren; mens registrene lagrer data, ALU'en utfører operasjoner på dem, data-, adresse- og kontrollbussene knytter de ulike modulene sammen og MUX'ene ruter data til den relevante registerblokken, er det styringsenheten som styrer og koordinerer alt dette. Og styringen følger selvsagt direkte av programmet som utføres.

### ***Instruksjonsregisteret (Instruction Register)***

Et ferdig kompilert, kjørbart program består av *instruksjoner*. Hver instruksjon består av et antall bits som forteller nøyaktig hva prosessoren skal gjøre under utføringen av den. Instruksjonsregisteret er et ordinært register, men det har en meget spesiell rolle: det inneholder til enhver tid den instruksjonen som skal utføres eller er under utføring.

### ***Instruksjonsdekoderen (Instruction Decoder)***

Det er alltid et mål å gjøre programmer så korte som mulig for å redusere behovet for lagrings- og overføringskapasitet, derfor tilstreber en bl.a. ofte å redusere antall bits i hver instruksjon. Vi kan tenke på et ferdig kompilert, kjørbart program som en slags komprimert oppgaveformulering, der alt som prosessoren skal gjøre er spesifisert i instruksjonene, men ikke alltid på en helt eksplisitt måte. Instruksjonsdekoderens oppgave er nettopp å tolke bitmønsteret i instruksjonsregisteret og omsette det i mer “anvendbare” styringssignaler.

Et konkret eksempel: hvis en prosessor inneholder 8 registre, og en instruksjon skal spesifisere at innholdet i to registre skal adderes og resultatet legges i et tredje register, er det tilstrekkelig å bruke 3 bit ( $2^3=8$ ) i instruksjonen for å angi ett register; følgelig må denne instruksjonen inneholde  $3 \times 3 = 9$  bits for å kunne spesifiserer de to operandene og resultatregisteret. Under utføring av instruksjonen må derimot hver av disse 3-bits-kodene dekodes til åtte separate kontrollinjer, én for hvert register, som kan enable det angitte registeret og disable de øvrige sju. Slik dekoding er blant instruksjonsdekoderens oppgave.

### ***Styringslogikk (Control Logic)***

Denne blokken kan være en omfattende sekvensiell krets, som sammen med instruksjonsdekoderen produserer de riktige logiske signalene, til de riktige tidene, for at prosessoren skal utføre den aktuelle instruksjonen.

## **2.2.5 Eksempel på utføring av en instruksjon**

Programmer utføres én instruksjon om gangen, og hver instruksjon utføres grovt sett som følger. Det understrekes at denne fremstillingen er sterkt forenklet for å få frem det overordnede prinsippet.

Vi antar at instruksjonen som står for tur spesifiserer at tallet 3 skal legges til et tall som tidligere er blitt lagret i register A, og at resultatet skal legges tilbake i A-registeret.

#### **1. Henting av instruksjon:**

- innholdet i PC-registeret (instruksjonspekeren) settes ut på adressebussen, og et styringssignal sendes til det eksterne programminnet (gjennom en av linjene i *Control Bus*) om at tilhørende data ønskes. Programinstruksjonen leses deretter fra databussen inn i instruksjonsregisteret.

#### **2. Dekoding og utføring: Styringsenheten**

- ser (av innholdet i instruksjonsregisteret) at en konstant skal inngå i operasjonen,

- setter ut adressen der denne konstanten (her: tallet 3) er lagret, på adressebussen (dette kan typisk være adressen etter den som inneholdt selve instruksjonen, og det er nettopp denne adressen som nå ligger i instruksjonspekeren *PC*),
- setter ut et styringssignal til det eksterne programminnet om at tilhørende data ønskes,
- mellomlagrer deretter konstanten fra databussen i registeret merket *TEMP*.
- mellomlagrer innholdet i A-registeret i registeret merket *ACU*.
- identifiserer (ut fra innholdet i instruksjonsregisteret) instruksjonen som operasjonen *Addér*, og aktiverer styringslinjene til ALU'en slik at den er konfigurert for addisjon. Resultatet av addisjonen legges direkte på databussen.
- laster resultatet fra databussen inn i A-registeret.

Instruksjonen er dermed utført, og prosessoren henter neste instruksjon.

Noen av trinnene i instruksjonsutførelsen kan foregå i parallell, mens andre (f.eks. alle de ulike operasjonene som involverer data- eller adressebussen) åpenbart må skje sekvensielt. Det er nettopp denne koordineringen som foregår i styringslogikken.

## 2.3 AVR: Et moderne eksempel

Nå som vi har gjort oss kjent med en relativt enkel prosessor og de typiske elementene som inngår der, er det naturlig å ta et blick på et mer moderne eksempel. Vi velger Atmel Corp. sin 8-bits AVR-arkitektur, som er utviklet i Trondheim og er blitt en av tidenes største salgssuksesser i sitt markedssegment. Vi bruker også denne prosessoren utstrakt i undervisningen ved NTNU, der den blant annet inngår i en av øvingene i faget som dette kompendiet er skrevet for.

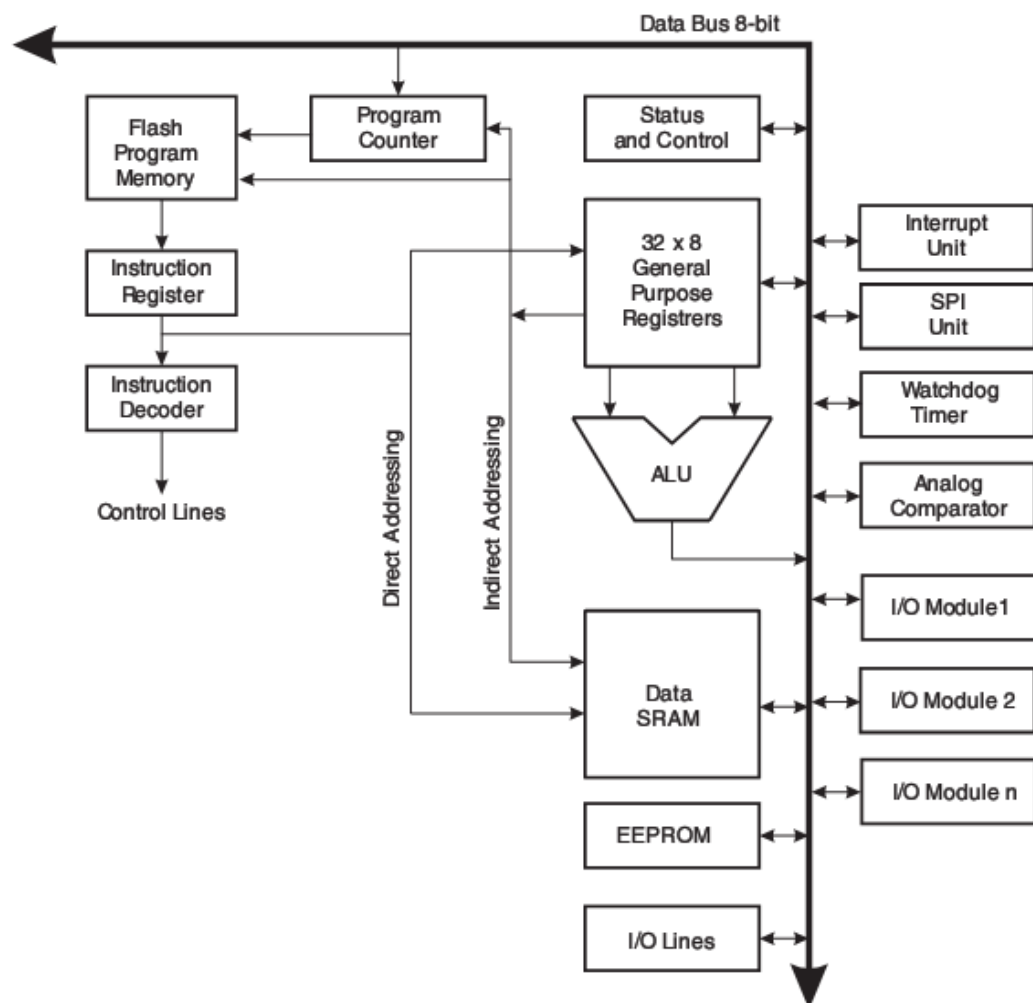
AVR'en finnes i en mengde utgaver med ulik innpakning, klokkefrekvens og ulike maskinvareressurser, men Figur 2.4 viser en typisk arkitektur. Fra Z80-arkitekturen kjenner vi umiddelbart igjen den aritmetisk-logiske enheten (*ALU*), instruksjonspekeren (*Program Counter*), instruksjonsregisteret og instruksjondekoderen.

Det er imidlertid også store forskjeller mellom de to arkitekturene, og disse beskrives kort nedenfor.

### 2.3.1 Mikroprosessor vs. mikrokontroller

Mens Z80 definitivt må kunne kalles en klassisk *mikroprosessor*, er AVR en familie av *mikrokontrollere* (eng.: *microcontroller*, “mikro-styringssystem”), komplette mikro-datamaskiner som i tillegg til selve prosessoren har et fullt sett med lagringsenheter og kommunikasjonsgrensesnitt mot omgivelsene. Dette gjør at en i enkelte anvendelser kan klare seg med *én eneste integrert krets*, eller generelt et meget lite antall komponenter, for å realisere et komplett elektronisk styresystem (logikkstyring og/eller reguleringssystem).

Vi skal se litt nærmere på noe av det som skiller AVR-kontrollere fra Z80-prosessorer.



**Figur 2.4** Eksempel på moderne mikrokontrollerarkitektur: Atmel AVR.  
 Figuren er hentet fra <http://www.avrportal.com/>.

### **Registerstruktur og instruksjonssett**

Z80 kan sies å ha 2x10 8-bits- og fire 16-bits “arbeidsregistre”, der mange av disse er svært spesialiserte og ikke kan brukes fritt under programutførelsen. Til sammenlikning har AVR 32 generelle 8-bits registre som kan brukes nesten helt fritt. Dette gir selvsagt større fleksibilitet og mer effektive programmer.

Videre har AVR et relativt lite omfangsrikt instruksjonssett, men tilgjengjeld utføres nær sagt alle instruksjonene på én klokkesykel, i motsetning til Z80 som bruker opp til seks sykler på én instruksjon. Instruksjonssettet til AVR-kontrollerne er videre optimalisert for å gi ekstremt kompakt kode ved kompilering av høynivåspråk som C.

### **Lagringsmuligheter**

Figur 2.4 viser hele tre typer lagringsmedium: *Flash Program Memory*, *EEPROM* og *Data SRAM*. Disse er basert på ulik teknologi, og er optimalisert for ulike bruksmønstre.

AVR-en har en såkalt Harvard-arkitektur, noen som betyr at program og data lagres separat. Programminnet er typisk basert på flash-teknologi, som også benyttes i moderne “minnepinner”. Dette er et minne som beholder sitt innhold selv om spen-

ningforsyningen forsvinner. Innholdet kan også endres (overskrives) utallige ganger, slik at kontrolleren kan programmeres om ved behov (f.eks. under utvikling og debug-ging av programvaren).

Flash-minne må skrives i relativt store “bolker” om gangen. I et tilpasset datasystem er det ofte nødvendig å lagre unna mindre mengder data, så som enkelte måleverdier, parametre, justerte alarmgrenser osv. Slike data kan med fordel lagres i AVR’ens *EEPROM* (*Electrically Erasable Programmable Read-Only Memory*), som har liknende egenskaper som flash-minnet men kan skrives byte for byte.

Blokken *Data SRAM* representerer kontrollereens arbeidsminne, der variabler, dynamiske datastrukturer etc. lagres under programkjøring. Betegnelsen *SRAM* står for *Static Random-Access Memory*, og denne minneblokken er prinsipielt realisert med samme teknologi som de generelle prosessorregistrene. Innholdet i *Data SRAM* kan imidlertid ikke inngå i ALU-operasjoner uten først å ha blitt flyttet til et register.

### ***I/O-ressurser***

I Figur 2.4 er I/O-ressursene (I/O = *Input/Output*, altså ressurser for inn- og utlesing av data og signaler) tegnet inn helt til høyre og nederst på figuren. Vi skal ikke gå detaljert inn på disse, men konstaterer at AVR-kontrolleren har et meget fleksibelt batteri av I/O-muligheter som kan omfatte alt fra avanserte digitale kommunikasjonsgrensesnitt til generelle digitale og analoge I/O-linjer.

Mange av de fysiske linjene kan konfigureres ved å sette eller resette bits i spesielle konfigureringsregistre (“kontrollregistre”), slik at systemutvikleren eksempelvis kan velge hvor mange av de tilgjengelige linjene som skal være digitale innganger, digitale utganger, analoge innganger osv.. Dermed kan kontrolleren programmeres til å passe til en mengde ulike anvendelser med ulikt antall analoge og binære måle- og pådragssignaler.



---

## KAPITTEL 3

# *Transmisjonslinjer og bølgefenomener*

### 3.1 Innledning

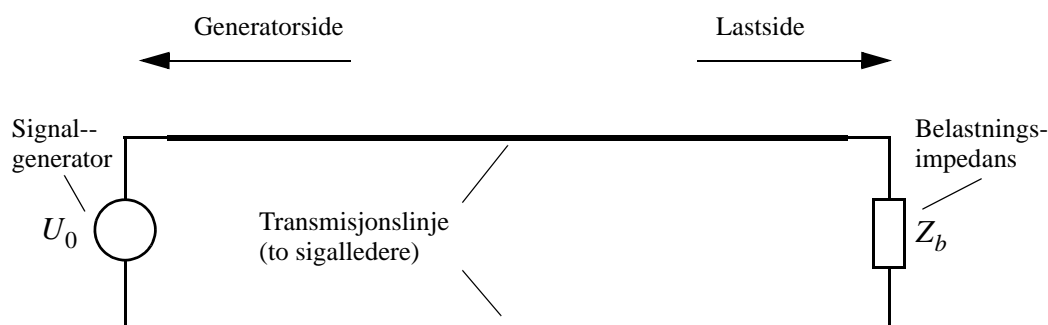
Overføring av informasjon mellom enheter, transmisjon, er nødvendig i de fleste systemer. Når man overfører et signal gjennom en linje vil det kunne oppstå en del uønskede fenomener, som forsinkelser, demping av signalet, avrunding av pulser og forstyrrelser som skyldes refleksjoner i linjeforgreninger og -avslutninger. Disse fenomenene vil bidra til å forvrengte det overførte signalet og begrense oppnåelig overføringskapasitet. I dette kapitlet skal vi se på de viktigste av disse fenomenene og vise hvordan vi unngår eller minimaliserer problemene de skaper.

Når vi bruker begrepet transmisjonslinjer tenker vi gjerne på lange kabler i f.eks. prosessanlegg, men også på korte kabler internt i instrumenteringssystemer og ledningsforbindelser i datamaskiner kan transmisjonslinjefenomener observeres. Temaene som dekkes av dette kapitlet er med andre ord svært viktige for alle moderne digital systemer.

#### 3.1.1 Terminologi og begrepsavklaring

Vi skal begrense analysen til spenningssignaler, men siden strøm og spenning alltid er forbundet med hverandre iht. Ohms lov, er alle resultatene like relevante i systemer der signaler overføres i form av strøm ("strømsløyfer").

Figur 3.1 viser en generell transmisjonslinje med signalgenerator  $U_0$  og belastnings-



**Figur 3.1** Transmisjonslinje med signalgenerator  $U_0$  og belastningsimpedans  $Z_b$ .

impedans  $Z_b$ . Signalgeneratoren representerer her den komponenten eller kretsen som *driver linja*, dvs. den kretsen som setter opp en spenning mellom signallederne. I digitale systemer er signalgeneratoren f.eks. utgangen fra en logisk port eller en driverkrets for en databuss, men i andre systemer kan den være utgangen av en analog signalforsterker eller en hvilken som helst annen krets som genererer et spenningssignal. Hvis linja er en høyspentlinje, kan signalgeneratoren rett og slett være generatoren (og den tilhørende transformatoren) i et tilknyttet kraftverk.

Impedansen  $Z_b$  representerer de elektriske egenskapene (sett fra transmisjonslinja) til komponenten eller kretsen som “tar imot” signalet fra signalgeneratoren. Siden det ofte er en spenningsforskjell (dvs. selve signalet) mellom signallederne, vil denne impedansen medføre at det trekkes en strøm gjennom kretsen; vi sier at  $Z_b$  *belaster* linja, derav betegnelsen *belastningsimpedans* eller bare *lastimpedans*. I et digitalt system kan  $Z_b$  f.eks. representere inngangsimpedansen til en logisk komponent, og i dette tilfellet kan den være meget stor (i størrelsesorden  $M\Omega$  eller  $G\Omega$ ). Som vi skal se er det ofte behov for en belastningsimpedans med en helt konkret verdi, og dette kan gjøres ved å kople inn diskrete komponenter, f.eks. en motstand, mellom signallederne. Denne kalles da en *linjetermineringsmotstand*, *termineringsmotstand* eller bare *linjeterminering*. Impedansen  $Z_b$  i modellen vår representerer resultatanten av tilkoplede utstyr og eventuelle diskretet linjetermineringer.

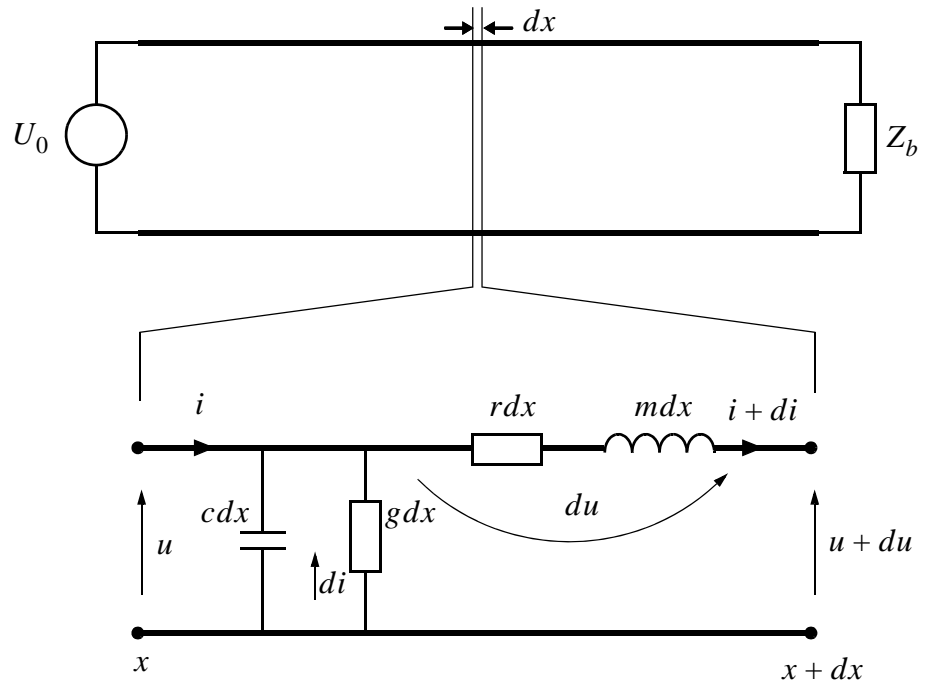
## 3.2 Telegraflikningene

I dette avsnittet skal vi ta utgangspunkt i kjente og grunnleggende fysiske egenskaper ved transmisjonslinjen. Vi skal deretter etablere en matematisk modell som viser oss signalers utbredelse lang linjen og hvordan signaler dempes med tid og avstand, og som beskriver fenomener som bølgeforplantning og signalrefleksjon.

Den matematiske behandlingen i dette avsnittet er på ingen måte komplett, og den er primært tatt med for å illustrere hvordan vi med utgangspunkt i en kretsteknisk modell kan komme fram til differensiallikninger som i sin tur gir opphav til bølgelikninger. De viktigste kunnskapselementene i dette avsnittet er den differensielle kretsekvivalenten i figur 3.2, det faktum at transmisjonslinja kan beskrives med et sett bølgelikninger. Implikasjonene av dette er beskrevet i resten av kapitlet.

### 3.2.1 Likningene

Vi bruker størrelser med enheter fra SI-systemet; spesifikk motstand med enhet Ohm [ $\Omega$ ], induktans med enhet Henry [H], kapasitans med enhet Farad [F], konduktans med enhet Siemens [S] og lengde med enhet meter [m]. I figur 3.2 vises transmisjonslinja



**Figur 3.2** Transmisjonslinje med differensiell ekvivalentkrets.

vi tok for oss i figur 3.1. Mellom signalgenerator og belastningsimpedans antar vi at vi har en uendelig lang transmisjonslinje. Betrakt nå en liten bit av linja med lengde  $dx$  [m]; denne kan representeres med en ekvivalent krets som vist nederst i figur 3.2. Langs linja vil vi i praksis ha en motstand  $r$  [ $\Omega/\text{m}$ ] som representerer den ohmske motstanden i selve linja, og selvinduktansen representeres ved induktansen  $m$  [H/m]. I parallell har vi kapasitansen mellom lederne,  $c$  [F/m]. Vi tar også med en parallellmotstand som representerer strøml lekkasjen mellom lederne – her representert ved den inverse størrelsen, konduktansen  $g$  [S/m] (for å gi enklere uttrykk i den følgende utledningen). Verdien av disse komponentene er angitt pr. lengdeenhet, slik at de virkelige verdiene finnes ved å multiplisere med lengden  $dx$  [m]. Seriermotstanden i linjestykket  $dx$  er dermed gitt ved  $R = rdx$  [ $\Omega$ ], induktansen ved  $L = mdx$  [H], parallellkapasitansen ved  $C = cdx$  [F] og lekkasjekonduktansen ved  $G = gdx$  [S]. Hele transmisjonslinjen kan tenkes å bestå av en seriekopling av et stort antall slike ledd.

Siden impedansene i modellen ovenfor er *jevnt fordelt langs hele transmisjonslinja* (i motsetning til diskrete komponenter som vi kjenner fra elementær kretsanalyse), vil strømmer og spenninger i kretsen variere både på langs av linja og med tiden;  $u = u(t, x)$  og  $i = i(t, x)$ . Følgelig kan disse verdiene deriveres både med hensyn på  $x$  og  $t$ . Vi innfører nå skrivemåten  $\partial u / \partial x$  for den *partiellderiverte*<sup>1</sup> av  $u$  med hensyn på  $x$ , dvs:  $\partial u / \partial x$  er den deriverte av  $u$  med hensyn på  $x$  ved en gitt (konstant) tid  $t$ .

Tilsvarende er  $\partial u / \partial t$  den deriverte av  $u$  med hensyn på  $t$  ved en gitt (konstant) posisjon  $x$  langs linja. Samme skrivemåte brukes selvsagt for de partiellderivate av strømmen  $i$ .

Betrakt nå transmisjonslinjestykket  $dx$  i figur 3.2. Spenningen i posisjon  $x$  (t.v. i figuren) er  $u$ , mens spenningen i posisjon  $x + dx$  (t.h. i figuren) er  $u + du$ . Spenningsforskjellen er følgelig gitt av uttrykket  $u - (u + du)$ . Dette må tilsvare spenningsfallet over serieresistansen og -induktansen i modellen, som er gitt av hhv.  $iR$  og  $L\partial i / \partial t$ . Vi får dermed likningen

$$u - (u + du) = iR + L \frac{\partial i}{\partial t}. \quad (10)$$

Ved å utnytte identiteten  $du = (\partial u / \partial x)dx$  samt definisjonene av størrelsene  $R$  og  $L$  ovenfor, kan (10) skrives som

$$u - \left(u + \frac{\partial u}{\partial x}dx\right) = i(rdx) + \frac{\partial i}{\partial t}(mdx) \quad (11)$$

Med tilsvarende resonnement får vi for strømmen:

$$i - \left(i + \frac{\partial i}{\partial x}dx\right) = u(gdx) + \frac{\partial u}{\partial t}(cdx)$$

En forenkling av disse uttrykkene gir:

$$\frac{\partial u}{\partial x} = -ir - m \frac{\partial i}{\partial t} \quad (12)$$

$$\frac{\partial i}{\partial x} = -ug - c \frac{\partial u}{\partial t} \quad (13)$$

---

1. Partiellderivasjon, partielle differensiallikninger, Fouriertransformasjon og andre matematiske begreper vi viser til i dette kapitlet tilhører pensum i viderekomne matematikkemner. Vi innfører begrepene her for å vise at det er en streng matematisk sammenheng mellom modellen i figur 3.2, den resulterende bølgelikningen og de angitte implikasjonene. Den interesserte student finner lett stoff om disse temaene på nettet, evt. i pensumlitteratur for matematikkfag i senere årskurs.

Dersom vi nå deriverer den første likningen mhp.  $x$  og den andre mhp.  $t$ , kan vi ved innsetting av både de opprinnelige og de deriverte uttrykkene eliminere én variabel fra hver av likningene og ende opp med følgende:

$$\frac{\partial^2 u}{\partial x^2} = mc \frac{\partial^2 u}{\partial t^2} + (rc + mg) \frac{\partial u}{\partial t} + rgu \quad (14)$$

$$\frac{\partial^2 i}{\partial x^2} = mc \frac{\partial^2 i}{\partial t^2} + (rc + mg) \frac{\partial i}{\partial t} + rgi \quad (15)$$

Dette er et sett med *bølgelikninger*, og blir gjerne kalt *telegraflikningene*. Løsningen på disse likningene forteller oss hvordan strøm- og spenningssignaler oppfører seg på transmisjonslinja.

### 3.2.2 Løsningen

Vi studerer her forløpet av vilkårlige signaler langs transmisjonslinja. Fra matematikken knyttet til den såkalte Fouriertransformasjonen vet vi at ethvert signal kan skrives som en sum av (potensielt uendelig mange) sinussignaler med ulike frekvenser. Vi kan derfor velge å se på kun én vilkårlig frekvens om gangen. Det som skjer med hele signalet vil da kunne uttrykkes som summen av bidragene fra hver frekvenskomponent.

Det kan vises at løsninger på bølgelikningen for spenningssignalet, (14), kan skrives på formen

$$u(x, t) = u_1(x, t) + u_2(-x, t) \quad (16)$$

der  $u_2$  beskriver en bølge som beveger seg fra generatorsiden mot lastsiden mens  $u_1$  er en bølge som beveger seg fra lastsiden mot generatorsiden. Matematikken åpner mao. for at signaler kan bevege seg begge veier på transmisjonslinja, noe som stemmer med vår intuisjon (en vanlig ledning fungerer selvsagt like godt begge veier). Helt konkret kan  $u_1$  og  $u_2$  skrives som følger:

$$u(x, t) = \underbrace{|a_1| e^{\alpha x} \cdot \cos(\omega t + \beta x + \phi_1)}_{u_1} + \underbrace{|a_2| e^{-\alpha x} \cdot \cos(\omega t - \beta x + \phi_2)}_{u_2}, \quad (17)$$

der  $\omega$  er frekvensen på det aktuelle sinussignalet. Konstanten  $\alpha$  angir hvor raskt amplituden på en bølge avtar (dempes) med posisjonen  $x$  langs linja, mens *bølgetallet*  $\beta$  angir antall bølgelengder per lengdeenhet. Konstantene  $\phi_1$  og  $\phi_2$  angir bare signalets fasevinkel ved  $x = 0$  og  $t = 0$ , og er uten prinsipiell betydning i denne sammenheng.

For oversiktes skyld gjentas at likning (17) viser angir oppførselen til et sinusformet signal med den vilkårlige frekvensen  $\omega$ . Ethvert ikke-sinusformet signal kan uttrykkes som en sum av slike sinusoider, og dermed uttrykker likning (17) indirekte oppførselen til *ethvert* signal på linja.

### 3.2.3 Karakteristisk impedans

La  $U_2(x) = |a_2|e^{-\alpha x}$  være den rent posisjonsavhengige faktoren<sup>1</sup> i uttrykket  $u_2$ , dvs. det leddet som beskriver *amplituden* til signalet i posisjonen  $x$ . Løsningen på strømlikningen (15) vil være på nøyaktig samme form som (17), og bestå av en bølge  $i_2$  med utbredelsesretning fra generatorsiden mot lastsiden, og en bølge  $i_1$  i motsatt retning. Den rent posisjonsavhengige faktoren i uttrykket for  $i_2$  (amplituden til strømdelen av signalet) er derfor på formen  $I_2(x) = |b_2|e^{-\beta x}$ .

I avsnitt 3.2.1 modellerte vi transmisjonslinja som en seriekopling av uendelig mange differensielle seksjoner med lengde  $dx$ . Når vi nå har uttrykk både for strøm og spenning, kan vi benytte Ohms lov til å finne et uttrykk for den ekvivalente impedansen. Vi innfører begrepet *linjens karakteristiske impedans*  $Z_0$ , definert som<sup>2</sup>

$$Z_0 = \frac{U_2(x)}{I_2(x)} \quad (18)$$

Det kan vises at  $Z_0$  kan skrives

$$Z_0 = \sqrt{\frac{r + j\omega m}{g + j\omega c}}. \quad (19)$$

Linjens karakteristiske impedans skal vise seg å ha stor praktisk betydning. Foreløpig skal vi bare oppsummere noen av dens egenskaper som følger av de foregående likningene:

- $Z_0$  er uavhengig av linjens lengde, og avhenger kun av linjens elektriske parametre  $r$ ,  $l$ ,  $c$  og  $g$
- $Z_0$  er en kompleks størrelse i det generelle tilfellet
- $Z_0$  er reell (ohmsk) hvis  $r = g = 0$ , dvs. hvis linja er uten noen form for ohmsk tap. Vi sier da at vi har en *ideell transmisjonslinje*. Det er selvsagt umulig å oppnå dette i praksis, men for moderne signalledninger med høy kvalitet er det en god approksimasjon.

1. Merk at også cosinusleddet i likningen er posisjonsavhengig, men dette tar vi ikke hensyn til her.

2. Vi tar utgangspunktet i uttrykkene for strøm og spenning for et signal som beveger seg fra signalgeneratoren mot lastsiden. Hvis vi hadde brukt uttrykkene for den motsatt rettede bølgen, ville vi fått motsatt fortegn (altså en negativ  $Z_0$ ) men samme tallverdi.

### 3.2.4 Implikasjonene

#### *Fasehastighet*

*Fasehastighet* er hastigheten til et punkt på en bølgeform. I likning (17) har vi uttrykt funksjonen  $u(x, t)$  som to cosinusfunksjoner,  $u_1(x, t)$  og  $u_2(x, t)$ . For å finne fasehastigheten til  $u_2(x, t)$  betrakter vi bevegelsen til et punkt  $(x, t)$ , hvor verdien for  $u_2(x, t)$  er konstant. Vi kan se for oss at vi sitter på en bølgetopp på bølgeformen beskrevet av  $u_2(x, t)$ , og rir på denne. Dette betyr at argumentet til cosinusfunksjonen i  $u_2(x, t)$  har konstant verdi:

$$\omega t - \beta x + \varphi_2 = \text{konstant} \Rightarrow x = \frac{1}{\beta}(\omega t + \varphi_2 - \text{konstant})$$

Fasehastigheten  $v_2$  for bølgen  $u_2$  da være:

$$v_2 = \frac{dx}{dt} = \frac{\omega}{\beta}$$

For  $u_1(x, t)$  kan vi se at vi får et lignende uttrykk, men fortegnet er motsatt fordi bølgen går motsatt vei:

$$v_1 = \frac{dx}{dt} = -\frac{\omega}{\beta},$$

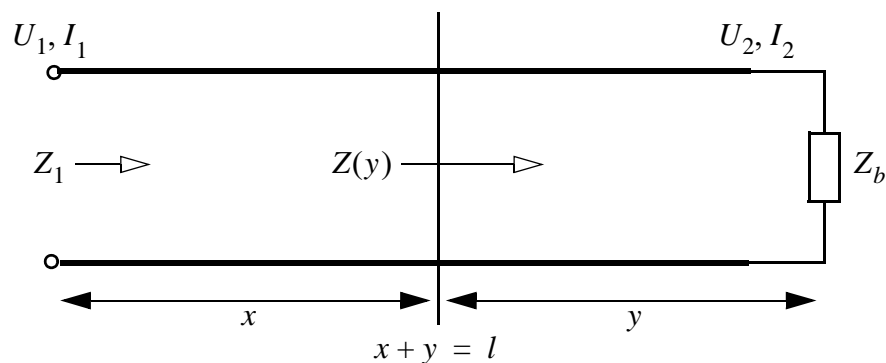
Likning (17) kan nå skrives om slik at vi tar eksplisitt hensyn til fasehastigheten. Vi setter  $v = v_2 = -v_1$ , og vi får

$$\begin{aligned} u_2(x, t) &= |a_2| e^{-\alpha x} \cdot \cos\left(\omega\left(t - \frac{x}{v}\right) + \varphi_2\right) \\ u_1(x, t) &= |a_1| e^{\alpha x} \cdot \cos\left(\omega\left(t + \frac{x}{v}\right) + \varphi_1\right) \end{aligned} \quad (20)$$

#### *Linjen sett fra et vilkårlig punkt*

Hvis vi går inn på linjen og betrakte den sett fra et vilkårlig punkt i linjen mot belastningsimpedansen, bør vi kunne forvente at bealstningsimpedansen påvirker linjens effektive impedans med mindre avstanden til  $Z_b$  er uendelig lang. Hvis vi studerer linjen fra et punkt i avstand  $y$  fra bealstningsimpedansen, der  $l = x + y$  er linjens totale lengde (se figur 3.3), kan det vises at vi vil oppleve impedansen

$$Z(y) = \frac{U(y)}{I(y)} = \frac{Z_b \cosh \gamma y + Z_0 \sinh \gamma y}{\cosh \gamma y + \frac{Z_b}{Z_0} \sinh \gamma y} \Rightarrow Z(y) = Z_0 \frac{\frac{Z_b}{Z_0} + \tanh \gamma y}{1 + \frac{Z_b}{Z_0} \tanh \gamma y}, \quad (21)$$



**Figur 3.3** Transmisjonslinjen sett fra et vilkårlig punkt på linjen mot belastningsimpedansen.

der  $\gamma$  er en konstant.

Som vi kan se inneholder uttrykket for impedansen langs linjen funksjonen tangens hyperbolicus, som varierer fra  $+\infty$  til  $-\infty$ . Dette har to viktige implikasjoner:

- Avhengig av hvor vi kopler oss til linjen, vil vi altså kunne oppleve alle tenkelige impedanser, inkludert en effektiv kortslutning ( $Z(Y) = 0\Omega$ ).
- Dersom vi setter  $Z_b = Z_0$  så ser vi imidlertid at hele uttrykket forenkles til  $Z(y) = Z_0$ , og impedansen blir dermed uavhengig av  $y$ .

### Refleksjonskoeffisienten

Vi skal nå studere hva som skjer akkurat i enden av linja, dvs. ved  $x = l$ . Det kan vises at spenningen i dette punktet kan uttrykkes som:

$$\begin{aligned}
 U(l) &= \overset{\text{reflektert}}{\downarrow} \Gamma \left( I_2 \frac{Z_b + Z_0}{2} \right) + \overset{\text{innfallende}}{\downarrow} \left( I_2 \frac{Z_b + Z_0}{2} \right) \\
 &= I_2 \frac{Z_b + Z_0}{2} [1 + \Gamma]
 \end{aligned} \tag{22}$$

der størrelsen

$$\Gamma = \frac{Z_b - Z_0}{Z_b + Z_0} \tag{23}$$

kalles for *refleksjonskoeffisienten*.

Legg her merke til at spenningen  $U(l)$  uttrykkes ved den innfallende bølgen (bølgen som kommer fra signalgeneratoren mot belastningsimpedansen), med en korreksjonsfaktor  $(1 + \Gamma)$  fordi den innfallende bølgen blir *overlagret en reflektert utgave av seg selv*. Refleksjonskoeffisienten viser hvor kraftig den reflekterte bølgen er i forhold til den innfallende. Refleksjonskoeffisienten vil gjøre seg gjeldende ved hver ende av linjen. Sender vi et signal fra generatorsiden mot belastningen så vil denne



refleksjonsfaktoren gjøre seg gjeldende ved belastningen. Sender vi en bølge fra belastningsiden mot generatorpunktet, så vil vi også få en refleksjon på generatorsiden, men her med en annen refleksjonskoeffisient som avhenger av generatorens utgangsimpedans.

Vi skal legge merke til følgende:

- Refleksjonskoeffisienten har negativ verdi dersom belastningsimpedansen er mindre enn  $Z_0$  og positiv verdi dersom belastningsimpedansen er større enn  $Z_0$ .
- Dersom belastningsimpedansen har nøyaktig verdien  $Z_0$ , får vi  $\Gamma = 0$  og *ingen refleksjon*. Det er dette vi i praksis ønsker å oppnå i de aller fleste tilfeller.

### 3.3 Oppsummering og konklusjoner

Basert på det foregående kan vi trekke følgende konklusjoner.

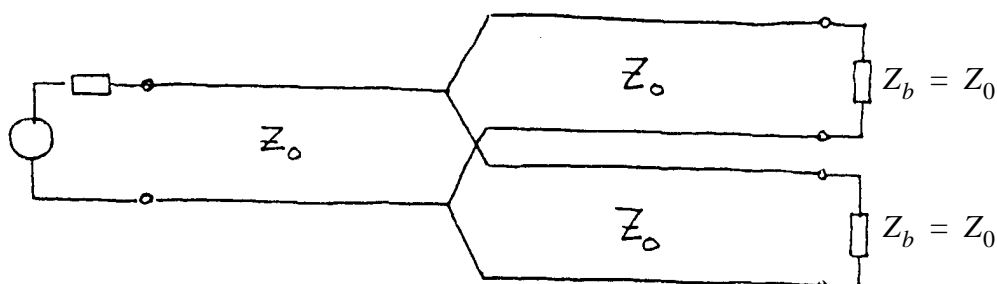
- Transmisjon skjer som overføring av effekt i form av en spenningsbølge og en strømbølge. Relasjonen mellom disse bølgene er gitt gjennom en impedans.
- Spennings- og strømprofilen langs transmisjonslinja består til enhver tid av en foroverrettet/innfallende bølge og en bakoverrettet/reflektert bølge av vilkårlig form, overlappet på hverandre.
- Linjer avsluttet med karakteristisk impedans gir konstant verdi på linjens ekvivalente impedans uavhengig av posisjon langs linjen og uavhengig av linjens lengde.
- Dersom  $r = g = 0$  har vi per definisjon en *ideell transmisjonslinje*.
- Ideell linje gir *ingen demping* i signalet langs linjen. Fysisk sett betyr dette at signalet beholder sin amplitude uendret, uavhengig av hvor lenge eller hvor langt det forplanter seg langs linjen.
- Med  $r = g = 0$  får vi  $Z_0 = \sqrt{m/c}$ , som er et *reelt* tall. Den karakteristiske impedansen for en ideell transmisjonslinje er med andre ord *ohmsk*.
- Fasehastigheten  $v$  i en ideell transmisjonslinje viser seg å være uavhengig av frekvens, og kan skrives som  $v = 1/\sqrt{m \cdot c}$ . Dette betyr at alle frekvenskomponenter har samme hastighet langs linjen. Vi får derfor *ingen dispersjon*, dvs. signalets bølgeform endres ikke mens bølgen forplanter seg langs linjen.
- Linjer avsluttet med sin karakteristiske impedans gir *ingen refleksjon*.

### 3.4 Eksempler

I dette avsnittet skal vi illustrere kapitlets hovedkonklusjoner med en rekke konkrete eksempler.

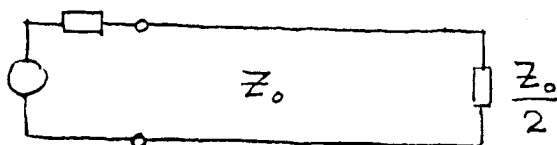
#### Eksempel 3.1 Linjeforgreining

Figur 3.4 viser en transmisjonslinje, som et stykke ute på linja blir splittet i to nye transmisjonslinjer som går til hver sin avslutningsimpedans lik den karakteristiske impedansen.

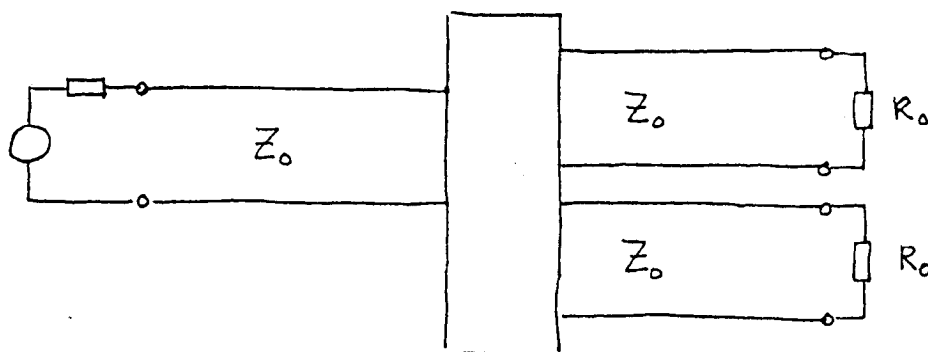


**Figur 3.4** Linjeforgreining.

Figur 3.5 viser at den splittede linjen gir en ekvivalent impedans lik  $Z_0/2$  sett fra et punkt t.v. for forgreiningen (resultatet av parallellkopling av to impedanser lik  $Z_0$ ). Denne kretsen vil få refleksjoner, og en linjeforgreining gjort på denne måten er derfor uhensiktsmessig. I et forgreningspunkt må vi ha en spesiell impedansomformer, som vanligvis er en transformator, dersom vi ønsker å ha ideelle forhold for hele linjen, slik som vist i figur 3.6.



**Figur 3.5** Linjeforgreining, ekvivalent.



**Figur 3.6** Linjeforgreining med impedansomformer.

### Eksempel 3.2 Åpen linje

Hva skjer på lastsiden av linja hvis vi ikke kopler den til noe som helst og lar signallederne ligge åpne? I dette tilfellet har vi en uendelig stor belastningsimpedans, dvs.  $Z_b = \infty \gg Z_0$ . Likeledes har vi at  $I_2(l) = 0$  (det går ingen strøm gjennom en uendelig stor impedans), men samtidig må vi være klar over at vi har en endelig sammenheng mellom disse verdiene gjennom Ohms lov:

$$I_2(l) = U_2(l)/Z_b. \quad (24)$$

Vi setter inn (24) i likning (22) og lar  $Z_b \rightarrow \infty$ , og får følgende sammenheng:

$$U(l) = \overset{\text{reflektert}}{\downarrow} \frac{U_2}{2} + \overset{\text{infallende}}{\downarrow} \frac{U_2}{2}$$

Tilsvarende beregning av strømmen gir

$$I(l) = -\frac{I_2}{2} + \frac{I_2}{2}.$$

Av disse ligningene ser vi at vi i dette tilfellet får en totalrefleksjon. Vi kan betrakte dette på følgende måte: Belastningspunktet mottar en innfallende strømbølge. Denne har ingen annen vei å gå enn tilbake. Den reflekterte strømbølgen må nødvendigvis ha motsatt fortegn dersom totalverdien skal være null i den åpne enden av linjen. På grunn av sammenhengen  $U = Z \cdot I$ , vil vi få en tilhørende spenningsbølge. Denne reflekteres i samme fase som den innfallende.

### Eksempel 3.3 Kortsluttet linje

I dette eksemplet har vi en belastningsimpedans  $Z_b = 0$ . Vi kan sette i våre ligninger:  $Z_b \ll Z_0$ . Vi har videre nødvendigvis at  $U_2(l) = 0$  på grunn av kortslutningen, men vi har fortsatt at sammenhengen mellom  $U_2$  og  $I_2$  er gitt gjennom likning (24).

Av likning (22) kan vi finne følgende sammenheng:

$$U(l) = -\frac{I_2 \cdot Z_0}{2} + \frac{I_2 \cdot Z_0}{2}.$$

Tilsvarende vil vi for strømmen få uttrykket

$$I(l) = \frac{I_2}{2} + \frac{I_2}{2}.$$

I disse likningene er det første ledd den reflekterte bølge og andre ledd den innfallende bølge. Vi ser at vi også her har totalrefleksjon. Denne gangen kan vi betrakte det som en innfallende spenningsbølge som i kortslutningspunktet må kompenseres med en reflektert bølge med motsatt fortegn fordi summen skal være lik null i kortslutningspunktet. Strømbølgen, derimot, passerer gjennom kortslutningen og reflekteres tilbake mot generatorsiden med samme fortegn.

### 3.5 Relasjonen linjelengde og signalfrekvens

Dersom vi tenker oss at vi beveger oss langs linjen i  $x$ -retning tilsvarende en bølgelengde,  $\lambda$ , kan vi for denne strekningen sette opp følgende ligning:

$$\beta\lambda = 2\pi$$

Når bølgen har en fasehastighet  $v$  vil den i løpet av tiden  $T$  nå en strekning  $\lambda$  hvor  $v = \omega/\beta$ , hvilket gir oss:

$$\lambda = vT$$

Tidligere har vi f.eks. for den innfallende bølgen skrevet ligningen på følgende måte:

$$u_2(x, t) = |a_2|e^{-\alpha x} \cdot \cos\left[\omega\left(t - \frac{x}{v}\right) + \varphi_2\right].$$

Denne ligningen kan vi omskrive:

$$u_2(x, t) = |a_2|e^{-\alpha x} \cdot \cos\left[\omega t - \frac{2\pi x}{\lambda} + \varphi_2\right].$$

Vi ser i uttrykket i cosinus-funksjonen at det består av tre bidrag:  $\omega t$ ,  $2\pi x/\lambda$  og  $\varphi_2$ . Dersom linjelengden, den maksimale verdi av  $x$ , er vesentlig mindre enn  $\lambda$ , så vil bidraget til  $u_2$  fra dette leddet være neglisjerbart. Dette betyr følgende:

- Teorien i dette kapitlet er bare relevant for linjer som er lengre enn bølgelengden til signalene som transmitteres i linjen.
- For linjer som er vesentlig kortere enn signalenes bølgelengde, kan vi se fullstendig bort fra bølge- og refleksjonsfenomenene. Vi kan i disse tilfellene betrakte linjen som den ikke har noen lengde og ekvivalere den med et rent RC-ledd (lavpass-filter).

- Anser vi linjen som lang i forhold til bølgelengden, er fasehastigheten gitt av uttrykket  $v = 1/\sqrt{m \cdot c}$ . Virkelige verdier for  $v$  vil være ca. 50-100% av lyshastigheten  $c$  som er lik  $3 \cdot 10^8$  m/sek. Ved hjelp av formelen for bølgelengden kan vi sette opp følgende tabell for sammenhengen mellom frekvens transmittert i linjen og den tilhørende bølgelengde:

Tabell 3.1

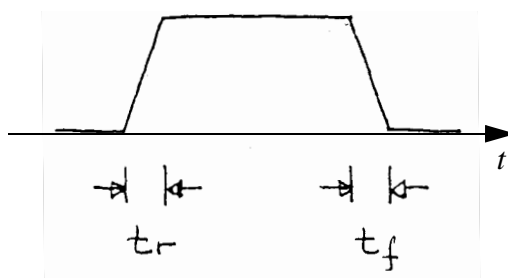
Signalfrekvens [MHz]	Bølgelengde [m]
1	150 - 300 m
10	15 - 30 m
100	1,5 - 3,0 m
1000	0,15 - 0,30 m

Vi ser her at allerede ved 10 MHz er bølgelengden 15-30 m, dvs. innenfor den størrelsen vi vil ha i små prosessanlegg og i typiske kontornettverk. Ved 100 MHz har vi bølgelengder ned i størrelsesorden 1,5-3,0 m, og dette er lengder som man oppnår ved kabling internt i en datamaskin. Dette betyr at *for frekvenser over 100 MHz må vi betrakte enhver linjestubb som en transmisjonslinje* og derved ta hensyn til dette i selve konstruksjonen.

### 3.6 Pulser

Figur 3.7 viser en enkelt puls med en stigetid og en falltid. Vanligvis er disse av samme størrelsesorden, men ikke nødvendigvis like store. Vi vet fra matematikken at vi via Fourieranalyse kan dekomponere pulser i sinus- og cosinusledd. Fra denne teorien kan man utlede at frekvenskomponenter opp til en frekvens  $f_n = 1/(2 \cdot t_r)$  er nødvendige å ta med for å få en god beskrivelse av pulsen, der  $t_r$  er stige-/falltiden (se figur 3.7). Moderne logikkfamilier har  $t_r$  i området 1-5 nanosekunder. Dette medfører at

$$100 \text{ MHz} < f_n < 500 \text{ MHz}$$



Figur 3.7 Puls med stigetid  $t_r$  og falltid  $t_f$ .

Dette tilsvarer en bølgelengde i området 30 cm - 1,5 m. Dette sier oss altså at *selv linjelengder ned til noen centimeter må betraktes som transmisjonslinjer når vi har med hurtige logiske kretser å gjøre.*

### Eksempel 3.4 Pulstransport på linje

Vi tenker oss en linje med en signalkilde med en sprangfunksjon på 1 V, som i figur 3.8. Kilden har en indre impedans som er lik  $Z_0/2$ . Linjen har en karakteristisk impedans  $Z_0$  og belastningsimpedansen er  $2Z_0$ . Vi betrakter linjen som en ideell linje. Den endelige verdien som spenningen i  $u_2$  vil få, kan vi finne ved å se bort fra linjens dynamiske egenskaper og sette  $u_2 = u_1$ , og vi får da etter ohms lov at:

$$u_2 = 1\text{ V} \cdot \frac{2 \cdot Z_0}{\frac{Z_0}{2} + 2 \cdot Z_0} = \frac{4}{5}\text{ V} \quad (t = \infty)$$

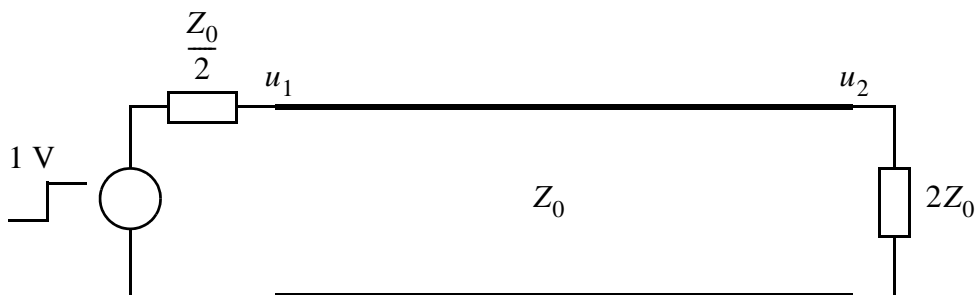
Refleksjonskoeffisientene finner vi som følger:

$$\Gamma_2 = \frac{2Z_0 - Z_0}{2Z_0 + Z_0} = \frac{1}{3}$$

$$\Gamma_1 = \frac{\frac{1}{2}Z_0 - Z_0}{\frac{1}{2}Z_0 + Z_0} = -\frac{1}{3}$$

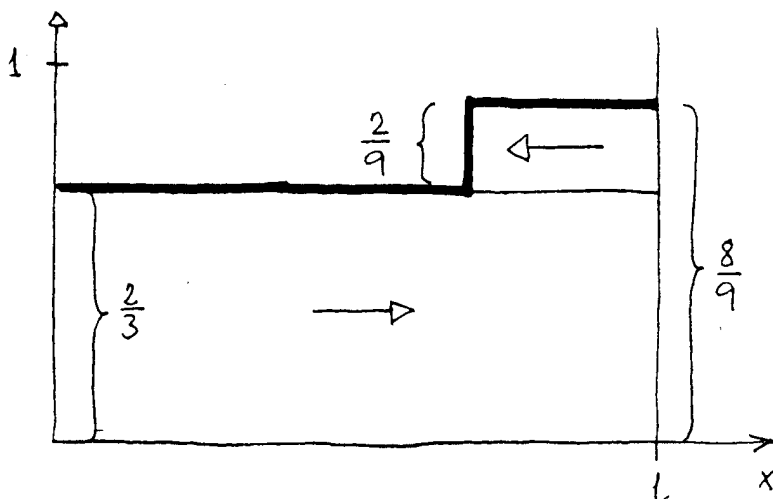
Ved tidspunkt  $t = 0$  vil  $u_1$  være den påtrykte spenningen, men delt mellom signalkildens utgangsimpedans og linjens karakteristiske impedans. Dette gir:

$$u_1(t = 0) = 1\text{ V} \cdot \frac{Z_0}{\frac{Z_0}{2} + Z_0} = \frac{2}{3}\text{ V}$$

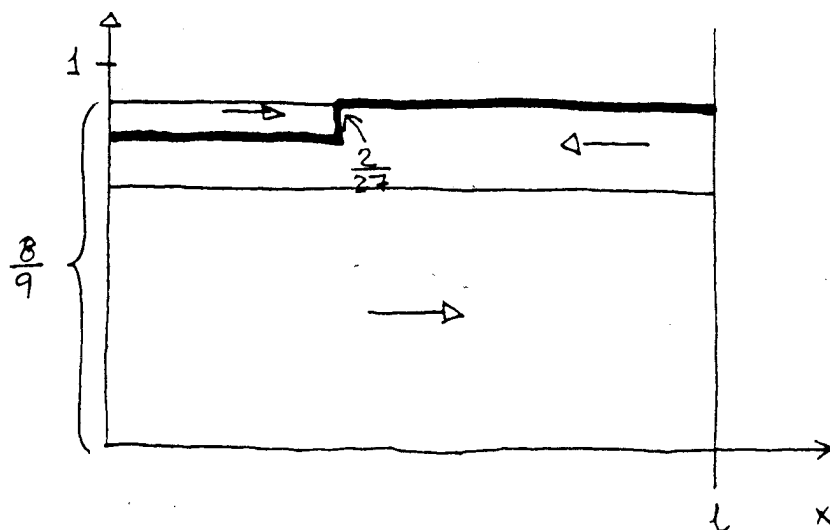


**Figur 3.8** Linje med signalkilde.

Denne sprangfunksjonen på  $2/3$  V vil nå bevege seg langs linjen bort til lastsiden, der den blir reflektert med en refleksjonskoeffisient lik  $1/3$ . Den reflekterte bølgen vil derfor være lik  $1/3$  av den innfallende bølge som var lik  $2/3$  V, dvs. at den reflekterte bølgen har en amplitude på  $2/9$  V. Amplituden i  $u_2$  vil være summen av den innfallende og den reflekterte bølgen, dvs.  $(2/3 + 2/9)$  V =  $8/9$  V. Dette er vist i figur 3.9. Den reflekterte bølge vil nå bevege seg mot punktet  $u_1$  der refleksjonskoeffisienten er lik  $-1/3$ , dvs. at  $1/3$  av den innfallende bølgen vil bli reflektert. Det gir en bølge imot  $u_2$  som har amplitude  $(-1/3)(2/9) = -2/27$ . Dette er vist i figur 3.10. Denne vil der igjen bli reflektert, og slik vil det fortsette til amplituden blir neglisjerbar på den bølgen som forplanter seg frem og tilbake.

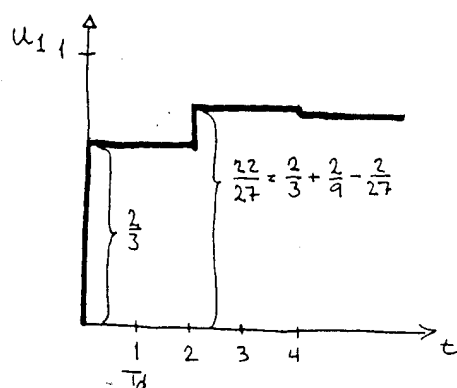


**Figur 3.9** Spenning langs linjen.



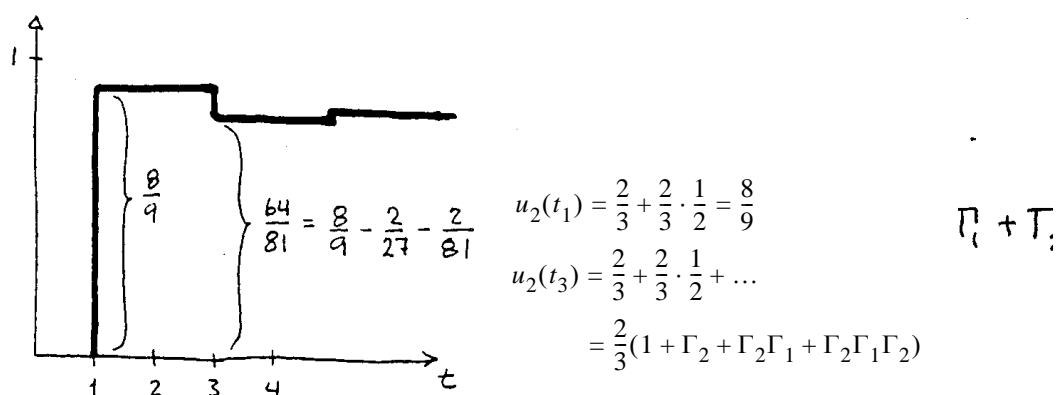
**Figur 3.10** Spenning langs linjen.

Dersom vi nå betrakter punktene  $u_1$  og  $u_2$  som funksjoner av tiden så vil vi observere for  $u_1$ -s vedkommende det som er vist i figur 3.11. Ved tidspunkt  $t = 0$  vil amplituden være  $2/3$  V. Bølgen forplanter seg med en tidsforsinkelse  $T_d$  fra punkt 1 til punkt 2, og tilsvarende like lang tid tilbake igjen. Etter en tid  $2T_d$  observerer vi den reflekterte bølge fra  $u_2$  som vil føre til at amplituden stiger opp til  $22/27$  V.



**Figur 3.11** Spenning i  $u_1$  som funksjon av tiden.

Ser vi derimot på  $u_2$ , vist i figur 3.12, vil den ha verdien null inntil tidspunktet  $T_d$ . Da vil den plutselig stige opp til verdien  $8/9$  V. Denne verdien vil vi beholde inntil tidspunkt 3 hvor vi får tilbake den reflekterte bølge fra  $u_1$ . Fordi denne ble reflektert med negativ refleksjonskoeffisient vil amplituden synke noe. Etter en tid vil refleksjonene være minimale.



**Figur 3.12** Spenning i  $u_2$  som funksjon av tiden.

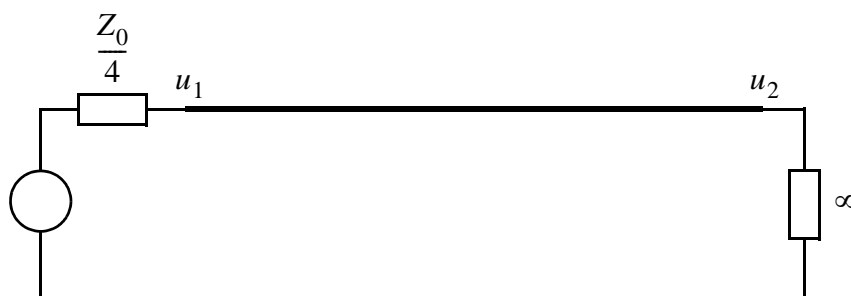
Vi legger merke til at spenningen på linja konvergerer mot sin endelige verdi  $4/5$  V, slik vi beregnet innledningsvis.



### Eksempel 3.5 Pulstransport på åpen linje

I dette eksemplet vil vi prøve å illustrere en situasjon som veldig ofte forekommer i praksis. Vi har en linje hvor kildeimpedansen er veldig lav i forhold til den karakteristiske impedansen, samtidig som mottaker har en veldig høy inngangsimpedans. Vi tenker oss at signalkilden ved tidspunkt  $t = 0$  sender ut en puls med 1 V amplitude og varighet  $8T_d$ . Basert på størrelsene i Figur 3.13 gjør vi følgende betraktninger:

$$u_1(t=0) = \frac{Z_0}{\frac{Z_0}{4} + Z_0} \text{ V} = \frac{4}{5} \text{ V} \quad \Gamma_1 = -\frac{3}{5} \quad \Gamma_2 = \frac{\infty - Z_0}{\infty + Z_0} = 1$$

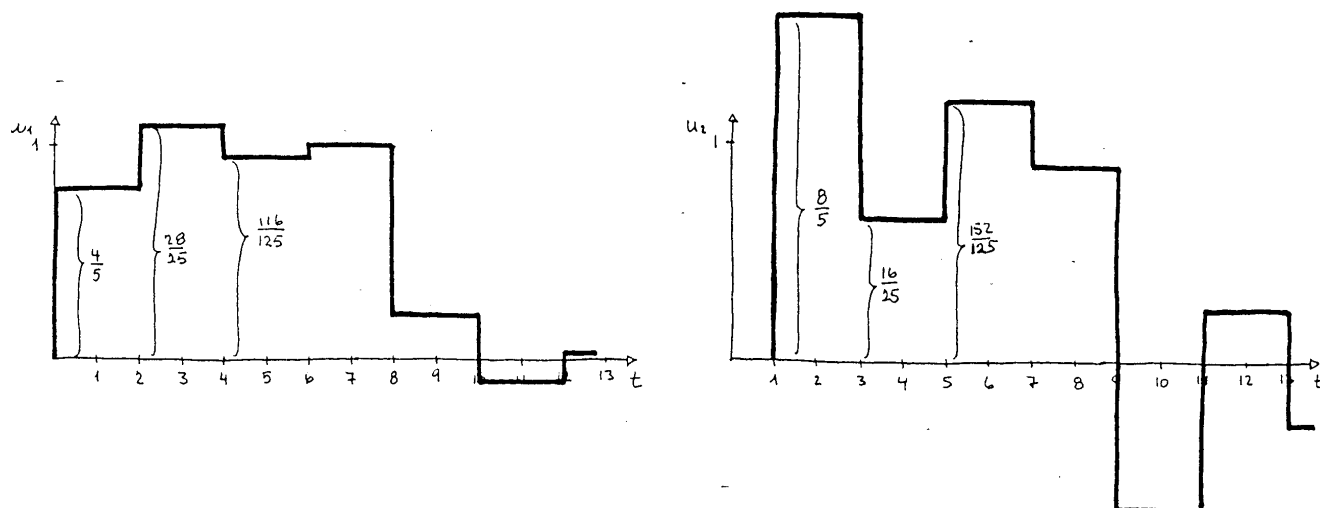


**Figur 3.13** Åpen linje.

Signalforløpet er skissert i figur 3.14. Legg merke til at kildeimpedansen og linjeimpedansen ( $Z_0$ ) ved  $t = 0$  i praksis utgjør en spenningsdeling, slik at  $u_1$  får det initielle spenningsnivået  $4/5$  V. Denne pulsen blir transmittert over linjen og blir i den åpne enden totalreflektert med samme fase ( $\Gamma_2 = 1$ ). I  $u_2$  får vi derfor en spenning lik den innkommende bølgen på  $4/5$  V, overlappet den reflekterte bølgen, som også er på  $4/5$  V, altså totalt  $8/5$  V. Den reflekterte bølgen på  $4/5$  V vil forplante seg mot signalkilden der den igjen blir reflektert, men her med negativ refleksjonskoeffisient ( $\Gamma_1 = -3/5$ ). Vi ser i figur 3.14 at ved tidspunkt 2 vil den reflekterte bølgen addere seg til den utsendte, og vi får en amplitude som stiger over den verdien som signalkilden har. Den reflekterte negative bølgen vil nå fram til den åpne enden av linjen etter 3 tidsenheter og vil da redusere amplituden i  $u_2$  noe. Denne bølgen vil igjen bli returnert mot signalkilden etter en ny totalrefleksjon. Vi ser av figuren for  $u_1$  og  $u_2$  et forløp som vil nærme seg 1 V på begge sider. Når så pulsen slås av vil vi få det samme forløpet, men med motsatt fortegn. Vi får da et fenomen som gjør at spenningen f.eks. i  $u_2$  vil bli negativ ( $-3/5$  V) i en kort periode. Dette vil gjennom refleksjoner føre til at vi også får en negativ spenning i punktet  $u_1$  et kort øyeblikk.

I praktiske logiske kretser som brukes som sendere og mottakere vil man i noen grad beskytte kretsene mot slike overspenninger ved hjelp av dioder. Disse diodene vil klippe av over- og underspenningene.

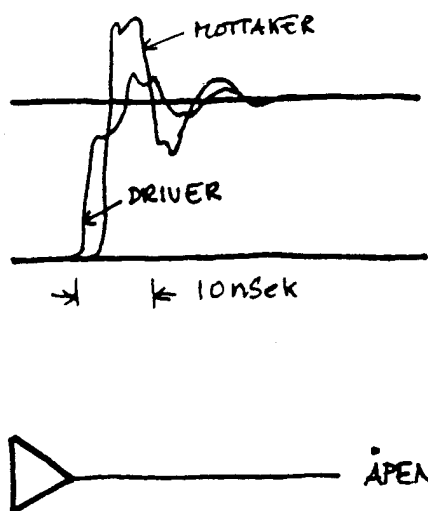
Karakteristisk impedans for vanlige linjetyper ligger i området fra noen titalls  $\Omega$  opp til ca.  $200 \Omega$ , med en typisk tidsforsinkelse i linjene ca. 5 nanosekunder pr. meter.



**Figur 3.14** Spenning i  $u_1$  og  $u_2$  som funksjon av tiden.

### 3.7 Terminering av linjer

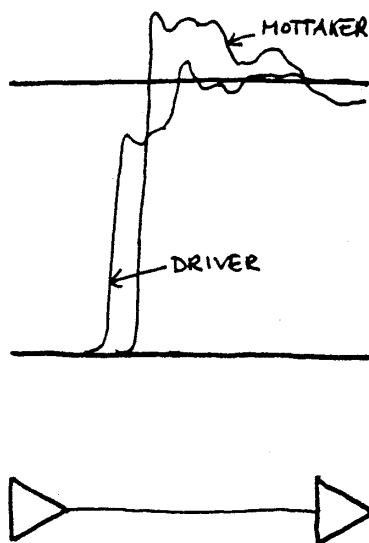
Terminering av korte linjer, spesielt i forbindelse med sammenkobling i datamaskiner, har vært og er fortsatt et meget vanskelig problem å takle. Vi skal i dette kapitlet vise en del eksempler på ulike måter og vise resultatet fra praktiske målinger. I figur 3.15 ser vi resultatet fra en 1 m lang linje som er åpen i mottakerenden. Vi får det typiske oversvinget som vi har antydnet i tidligere utledninger, og får på sendersiden et mer trappetrinnlignende utseende.



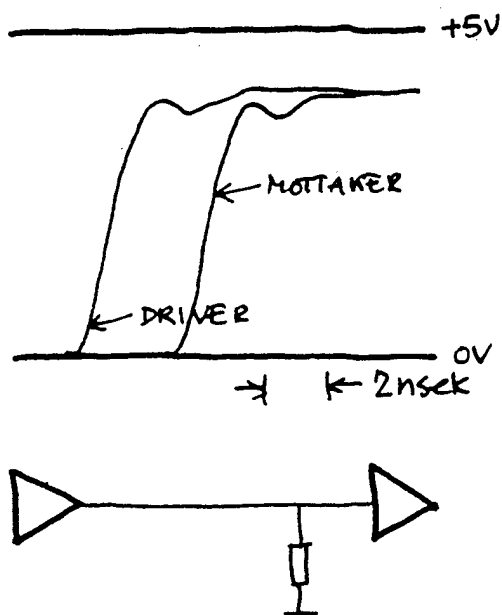
**Figur 3.15** Linje med åpen ende

I figur 3.16 har vi en mottaker med høy impedans og vi ser at fortsatt stiger spenningen til godt over  $V_{cc}$ , men vi får mindre refleksjoner enn om vi har en fullstendig åpen linje. I figur 3.17 har vi avsluttet linjen med en impedans som er ca.  $100 \Omega$ , og i nærheten av karakteristisk impedans for linjen. Vi ser da at vi får reproduisert en puls på mottaksstedet som er meget lik den vi sender ut. Oversvinget er redusert til ubetydelig.

Ulempen med en slik kobling er at vi får et effekttap i motstanden som vil være i størrelsesorden  $1/4 - 1/2$  Watt, og dette er et stort effektforbruk i forhold til kretsens forbruk.

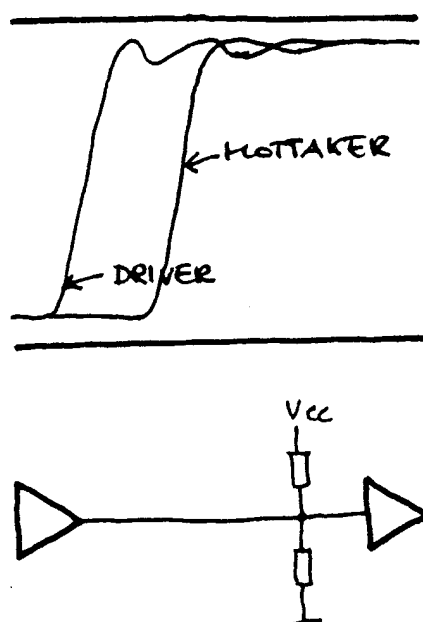


**Figur 3.16** Linje med høyohmig terminering.



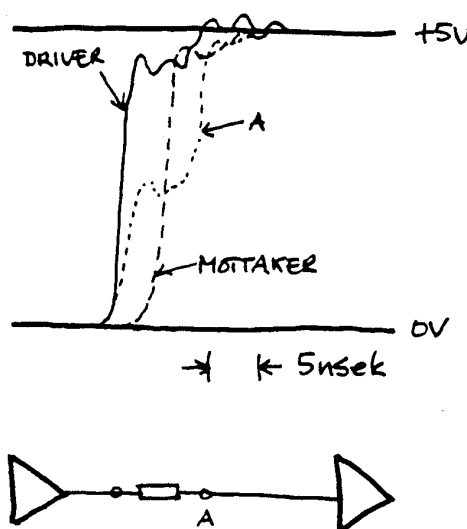
**Figur 3.17** Linje med termineringsmotstand.

Figur 3.18 viser en terminering med en motstand mot  $V_{cc}$  og en annen motstand mot jord. Dette kalles for en Thevenin-ekvivalent og passer best for TTL kretser. Mottatt signal er meget overensstemmende til det signalet som vi driver ut på linjen. Amplituden på det mottatte signalet blir noe høyere enn når vi har en enkel terminering mot jord.



**Figur 3.18** Linje terminert med Thevenin-ekvivalent

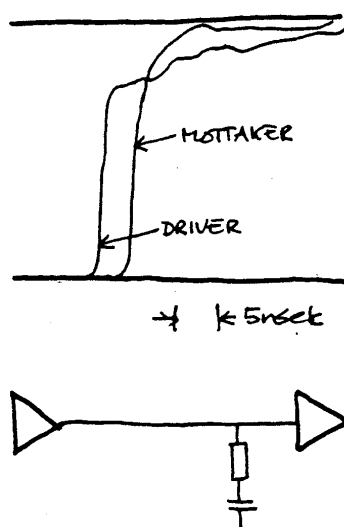
Nok en variant er beskrevet i figur 3.19. Her har vi en seriemotstand som er lik den karakteristiske impedansen for linja. Vi ser av diagrammet at dette gir relativt god signaloverføring, men vi ser også at på linjesiden av motstanden  $R$  vil vi få en spenning som har et kne. Det er ganske uheldig, spesielt ved CMOS kretser, hvor terskelspenningen for omslag ligger ved ca.  $V_{cc}/2$ .



**Figur 3.19** Linje med seriemotstand.

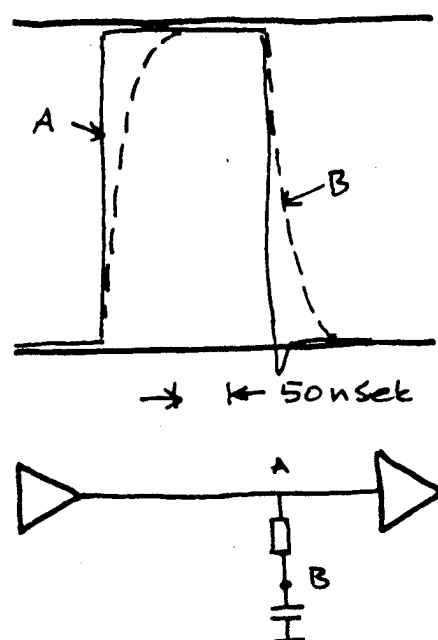
Dersom vi da har en forgrening i nærheten av dette punktet vil de mottakerne som ligger i nærheten være uvisse på hvilken tilstand de skal velge, og det vil skape støy i systemet. Det kan derfor være nødvendig å justere  $R$  slik at kneet blir liggende over eller under  $V_{cc}/2$ .

Nok en variant er vist i figur 3.20. Her har vi et RC-ledd som terminering av linjen. Hensikten med dette er at effektforbruket skal reduseres etter at den første innsvingningen av signalet har skjedd. Det mottatte signal har et tilfredsstillende utseende.



**Figur 3.20** Linjeterminering med RC-ledd.

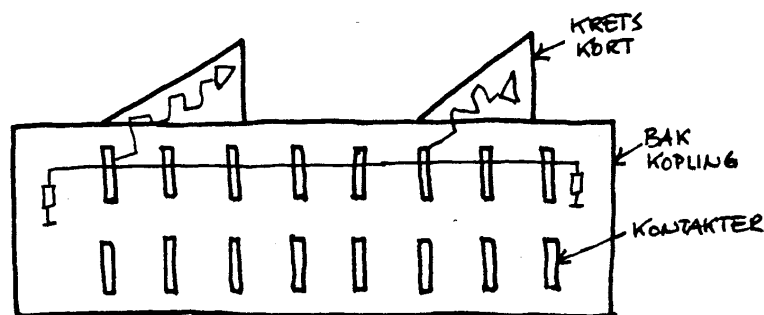
Figur 3.21 viser hvordan spenningen i punkt A og B på RC koblingen utvikler seg over tid. Etter relativt kort tid vil differansen være nær null og effekttapet i kretsen vil være tilsvarende.



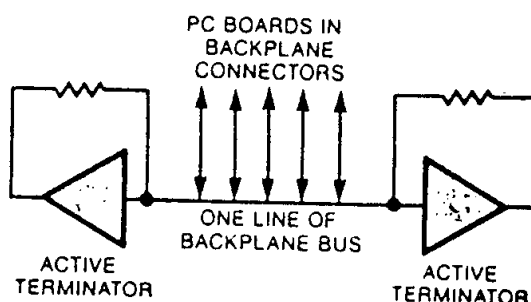
**Figur 3.21** Linjeterminering med RC-ledd. Opp- og utladning av kapasitansen.

I en datamaskin vil signalene bli generert på kort som igjen er stukket inn i en hylle hvor vi har en kobling mellom kontaktpunktene i en såkalt "back-wiring". Veien for et signal fra en sender til en mottaker kan derfor være nokså kronglete, og som antydnet i figur 3.22 har vi vanligvis terminering bare i hver ende av "back-wiringen". Vi får derfor et impedansforhold i denne linjen mellom sender og mottaker som er nokså udefinert.

For å bøte på dette har en prøvd med en aktiv terminering av "back-wiringen" i en slik hylle. Dette er vist i figur 3.23, hvor den aktive termineringen består i en operasjonsforsterker som er tilbakekoblet, og hvor man langs linjen får koblet signaler inn og ut.



**Figur 3.22** “Back-wiring” (bakplan).



**Figur 3.23** Aktiv linjeterminering.

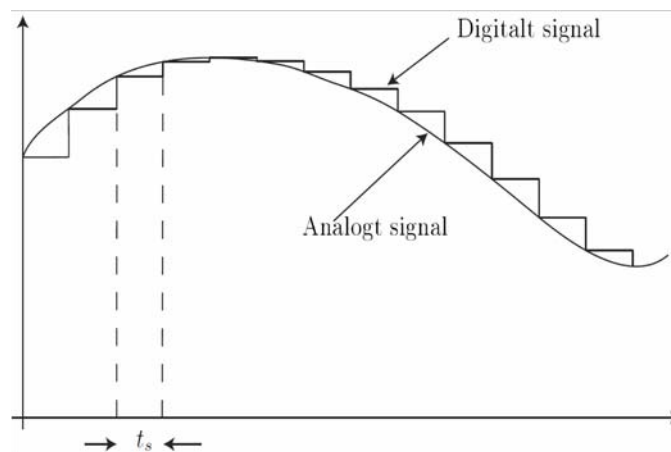
Dersom vi betrakter operasjonsforsterkeren alene hvor inngangskretsen på operasjonsforsterkeren nå vil være transmisjonslinjen, så vet vi at inngangsimpedansen på en operasjonsforsterker er lik den impedansen vi har i seriegrenen inn mot inngangen. I dette tilfelle har vi altså en transmisjonslinje med impedans  $Z(y)$ , med en verdi som vil variere ganske mye. Men operasjonsforsterkeren vil til enhver tid sette opp en impedans som er like stor. Det betyr at vi alltid vil ha perfekt impedanstilpasning så lenge vi ligger innenfor båndbredden for operasjonsforsterkeren.

## KAPITTEL 4

# Signaler og signalomsetning

### 4.1 Signalers rolle i reguleringsystemer

Signaler er informasjonsbærerne i styrings- og reguleringsystemer, og riktig behandling av signalene er derfor helt avgjørende for at systemet skal virke slik vi ønsker. Vi skal her skille mellom to typer signaler: *analoge* og *digitale*. En hovedforskjell på disse er at analoge signaler (for eksempel spennings- eller strømsignaler) er *kontinuerlige* i tid, mens digitale signaler er det vi kaller *diskrete* i tid. Et diskret signal består av et endelig antall øyeblikksverdier som representerer signalets verdi på bestemte tidspunkter. De to typene signaler er illustrert i figur 4.1.

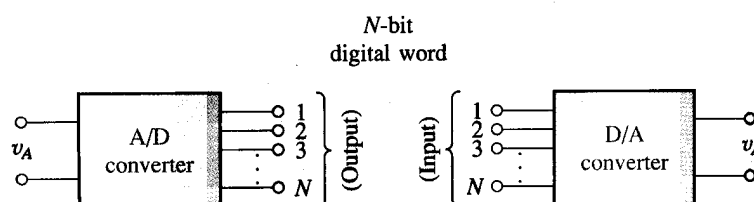


**Figur 4.1** Analogt og digitalt signal.

De fleste fysiske størrelser vi finner i en reguleringsprosess er analoge. Dette kan være nivået i en tank, hastigheten til en bil, posisjonen til en robot eller spenningen over en kondensator. Noe av signalbehandlingen som må gjøres på disse signalene er det mest praktisk (og noen ganger helt nødvendig) å utføre med analoge elektroniske kretser. På den andre siden foregår regulering og logikkstyring nesten alltid i en datamaskin som kun opererer med digitale signaler. Rent binære (“logiske”) signaler, som vi blant annet finner i alle logikkstyringssystemer, er det relativt lett å lese inn og ut av en datamaskin, og disse behandles derfor ikke videre her. Signalovertøring fra analoge måleinstrumenter til en datamaskin krever derimot at analoge signaler gjøres om til

digitale. Dette er kjent som analog-til-digital-omsetning eller bare *A/D-omsetning* (eng.: Analog-to-Digital Conversion, ADC. Forkortelsen brukes også for selve kretsen som forestår konverteringen, og bokstaven “C” står da for *Converter*). Andre begreper som brukes om dette fenomenet er *digitalisering* (eng.: *digitizing*) og *tasting* (eng.: *sampling*). Overføring fra en datamaskin til analoge pådragsorganer krever tilsvarende at digitale signaler gjøres analoge. Dette er kjent som digital-til-analog-omsetning eller *D/A-omsetning* (eng.: Digital-to-Analog Conversion, DAC. Denne forkortelsen brukes også for selve konverteringskretsen, da i betydningen Digital-to-Analog *Converter*).

Figur 4.2 viser A/D- og D/A-omsetteren representert som funksjonelle blokker på registernivå (jfr. avsnitt 2.1.2). Som antydnet tar A/D-omsetteren inn en analog signalverdi  $v_A$  og produserer et N-bits digitalt ord. Tilsvarende tar D/A-omsetteren inn et n-bits digitalt ord og produserer en analog signalverdi.



**Figur 4.2** A/D- og D/A -omsettere som funksjonelle kretsblokker.

I resten av dette kapitlet skal vi først se på noen prinsipielle sider ved innsamling av målesignaler og omsetning mellom digital og analog representasjon, og signaltilpassning som er nødvendig for å utnytte omsetterne optimalt. Til slutt skal vi ta for oss konkrete kretsløsninger for A/D- og D/A-omsettere.

## 4.2 Nettverkstyper for datainnsamling

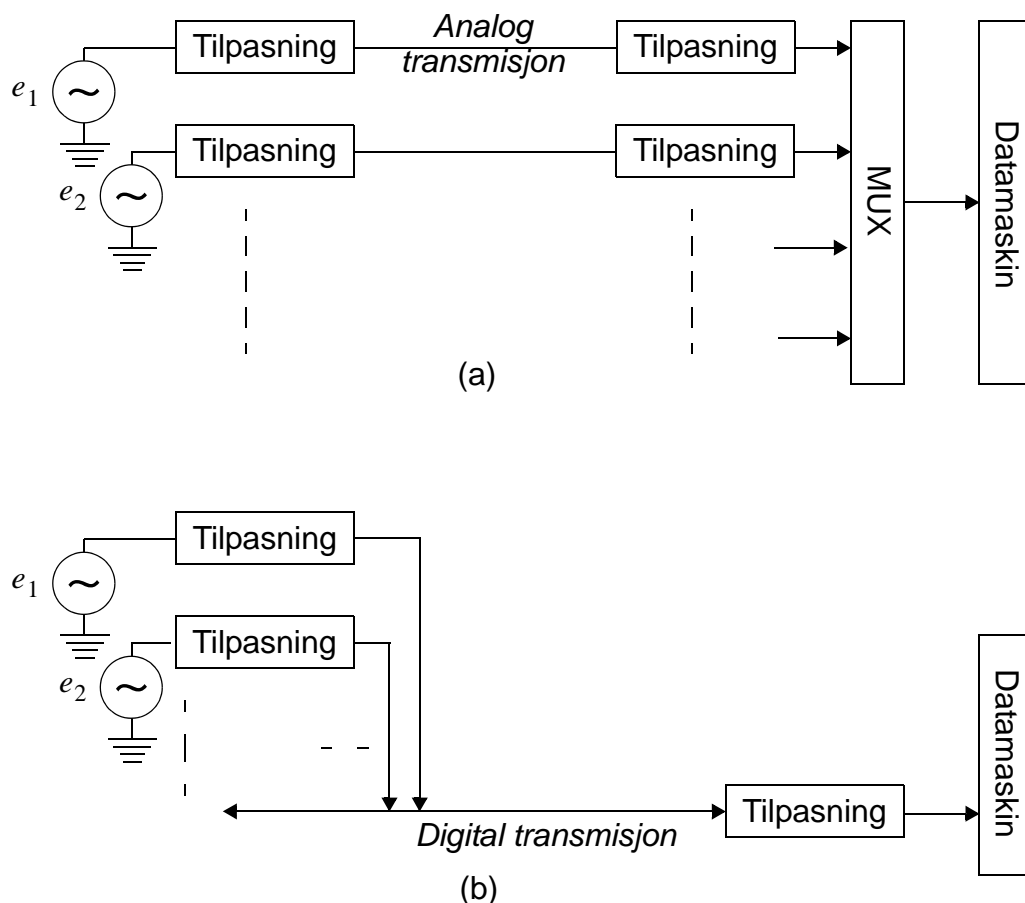
Ved innsamling av analoge data kan en etablere mange varianter av nettverk. I det aller enkleste tilfellet er signalet fra sensor tilpasset en dedikert transmisjonslinje av en eller annen type. En har da flere parallelle linjer inn til datamaskinen, én for hver sensor. Ved mottaksstedet går signalet gjennom et multiplekser- og signalbehandlingssystem (inkludert en A/D-omsetter) før det kan benyttes av datamaskinen (Figur 4.3a).

Signaleringen på transmisjonslinjen kan være analog eller digital. Ved analog signalering er det et kontinuerlig strøm- eller spenningssignal som overføres, mens det ved digital overføring er diskrete pulser. Ved svært høy båndbredde overføres gjerne digitale data ved hjelp av en eller annen modulasjonsteknikk.

Selve transmisjonslinjen kan være en alminnelig elektrisk ledning (et tråddpar) som er spesiallaget for formålet, en coaxialkabel, en fiberoptisk kabel eller en trådløs forbindelse.

Alternativt kan det koples flere sensorer til en og samme digitale transmisjonslinje (Figur 4.3b). Vi snakker da om et bussystem, en såkalt *feltbuss*, hvor hver av sensorene må tildeles tid slik alle får sendt sine måleverdier (dette er også et eksempel på multipleksing). Tradisjonelt har vi hatt svært mange systemer med parallelle, dedikerte transmisjonslinjer, men feltbussystemer tar etter hvert over fordi dette er mye billigere med hensyn på installasjonskostnader (ledningsforbruk og timeverk). Vi skal likevel konsentrere oss om analoge signalsystemer i resten av dette kapitlet, både fordi det





**Figur 4.3** (a) Tradisjonelt datainnsamlingsnettverk med dedikerte analoge linjer og analog multiplekser.  
(b) Feltbuss-system med felles digitalt medium mellom sensorer og den sentrale datamaskinen.

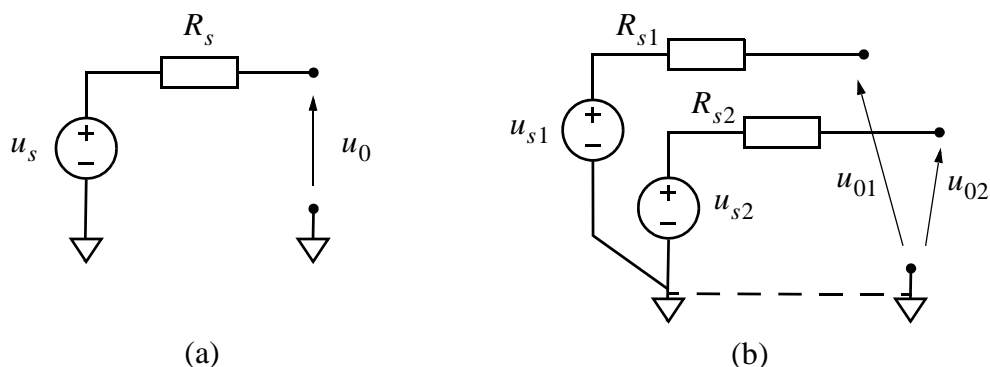
fortsatt finnes veldig mange slike installasjoner i industrien, og fordi selv et digitalt datainnsamlingssystem byr på mange av de samme analoge utfordringene i tilknytning til hver enkelt sensor; de analoge måleverdiene må jo leses inn i en datamaskin (i praksis en mikrokontroller) før de sendes ut på bussen. Dette tenker vi oss foregår i blokkene merket “Tilpasning” til venstre i Figur 4.3b.

### 4.3 Analoge signaler

Vi skal i fortsettelsen konsentrere oss mest om spenningssignaler, det vil si signaler der informasjonen er representert ved signalets spenning (et senere kapittel tar for seg strømsignaler). Vi bruker begrepene *signalgeneratoren* og *-kilden* synonymt om den enheten som produserer spenningssignalet. Signalgeneratoren kan med andre ord være en nivåmåler i en tank, en analog hastighetsmåler eller hva som helst annet som produserer et analogt signal.

### 4.3.1 Enpoledde signaler

I alle tilfeller kan denne kilden beskrives ved sin Thevenin-ekvivalent, som vist i figur 4.4a. Her er  $u_s$  spenningskilden og  $R_s$  den indre (utgangs-)impedansen i signalgeneratorens Thevenin-ekvivalent, mens  $u_0$  er signalgeneratorens utgangsspenning. Utgangsspenningen er referert til signaljord (representert med symbolet “ $\downarrow$ ”), det vil si at signalnivået defineres som spenningsforskjellen mellom signallederen og signaljord, med et positivt signalnivå hvis signallederen er mer positiv enn jordlederen. Kildeimpedansen  $R_s$  kan forøvrig være så liten at vi ofte ser bort fra den og tegner signalgeneratoren som en ren spenningskilde (dvs.  $R_s$  utelates fra modellen).



**Figur 4.4** (a) Enkel analog signalmodell - enpolet signalkilde som en Thevenin-ekvivalent. (b) Flere enpoledde signaler deler en felles singaljord.

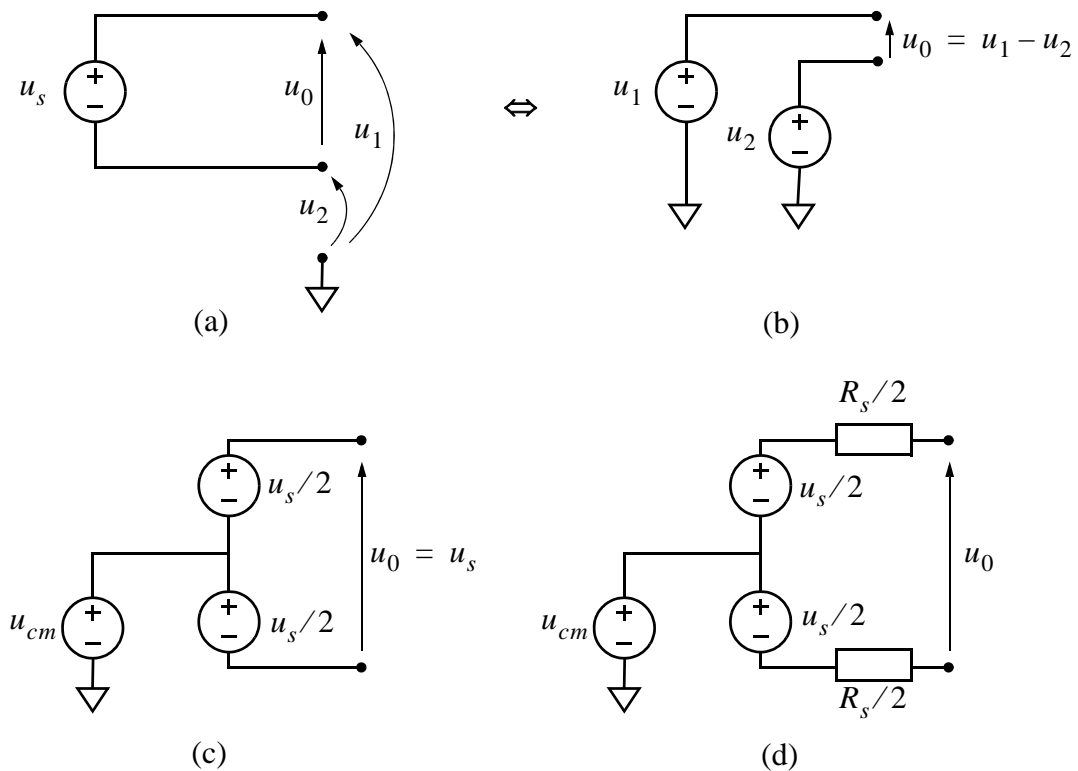
Hvis to eller flere signaler refereres til samme signaljord, trengs det i tillegg til den felles jordlederen bare én signalleder per signal. Dette er illustrert i figur 4.4b for to signalkilder med utgangsspenninger hhv.  $u_{s1}$  og  $u_{s2}$ , der den felles jordlederen er antydnet med en stiplet linje. I mange systemer er singaljord implisitt; hvis hele systemet for eksempel er omsluttet av en metallboks, kan selve boksen benyttes som jordreferanse og en trenger da ingen egen ledning for å representere signaljord<sup>1</sup>. Figur 4.4 representerer derfor det vi kaller en *enpolet* signalmodell; på engelsk brukes uttrykkene *unipolar*, *monopolar* eller *single-ended signals*.

### 4.3.2 Topolede (differensielle) signaler

Signalgeneratoren til venstre i figur 4.5a produserer et såkalt *topolet* eller *differensielt signal* (engelsk: *bipolar* eller *differential signal*). Vi har her for enkelhet skyld utelatt kildens resistans fra modellen. Signalnivået  $u_0$  er også her definert som spenningsdifferensen over signalgeneratorens spenningskilde, men i dette tilfellet er ikke spenningskilden koplet til noen fast referansespenning (signaljord) og vi må derfor føre fram begge signallederne for at mottakeren skal kunne tolke signalet. Som illustrert i figur 4.5b kan en differensiell signalgenerator betraktes som to (fiktive) spenningskilder<sup>2</sup>  $u_1$  og  $u_2$  som begge er referert til signaljord, men der nyttesignalet altså er definert som spenningsforskjellen mellom de to kildene, dvs.

$$u_0 = u_1 - u_2. \quad (25)$$

1. Denne løsningen er vanligvis bare aktuell i systemer med relativt lave krav til signalkvalitet og støyundertrykking.



**Figur 4.5** Topolet (differensiell) signalkilde – fire ulike modeller.

Et differensielt signal er i prinsippet uavhengig av en eksplisitt signaljord. På grunn av elektrisk støy fra omgivelsene og evt. andre forhold i prosessen vi måler på, kan signallederens spenningsnivå variere i forhold til signaljord, men de kan likevel overføre et gyldig differensielt signal så lenge støyen påvirker spenningene i begge signallederne *like mye og med samme fortegn*. Slik støy kalles *liketaktsstøy*, men vi bruker ofte det engelske uttrykket *common mode* også på norsk.

Common mode-spenningen kan modelleres som en fiktiv spenningskilde som altså påvirker begge signallederne *like mye og med samme fortegn*, mens det differensielle nyttesignalet per definisjon påvirker signallederne *like mye men med motsatte fortegn*. Vi sammenfatter dette i modellen vist i figur 4.4c, der signalkildens spenning  $u_s$  er delt i to symmetriske deler og common mode-spenningen er modellert som en spenningskilde  $u_{cm}$ . Spenningen på de to signallederne kan da uttrykkes som følger:

$$\begin{aligned} u_1 &= u_{cm} + u_s/2 \\ u_2 &= u_{cm} - u_s/2. \end{aligned} \quad (26)$$

- De to spenningskildene utgjør her til sammen én differensiell spenningskilde. Disse må ikke forveksles med det to énpolede signalgeneratorene i figur 4.4(b), som genererer to helt uavhengige signaler.

Hvis kildens utgangsresistans er så stor at den bør tas med i modellen, deler vi den i to like deler og tegner inn én halvdel i hver signalgrein som vist i figur 4.4d.

Modellene i figur 4.4c og d er eksempler på *balanserte* signalkilder, fordi alle kildens egenskaper er symmetrisk (balansert) fordelt mellom på de to signallederne.

### 4.3.3 Analoge signalformater

Det fins et vidt spekter av sensorer som gir oss elektriske signaler som representerer fysiske størrelser vi ønsker å måle. Alle disse enhetene har sine karakteristikk med hensyn på signaltipe, amplitude osv. For å kunne bruke dem effektivt i industriell sammenheng er det hensiktsmessig å konvertere samtlige signaler til et standard format før de koples til vårt styre- og overvåkingssystem. Dette setter oss nemlig i stand til å innhente og bearbeide mange ulike fysiske parametre med ett og samme system. Dermed forenkles systemets design og vedlikehold.

Gjennom årene er mange signalstandarder blitt utviklet; noen av disse er unike for spesielle land mens andre har fått global utbredelse. De mest populære standardene er (i vilkårlig rekkefølge):

Standarder for spenningssignaler:

- 0–5 V
- 0–10 V
- 1–5 V
- 2–10 V

Standarder for strømsignaler:

- 1–5 mA
- 0–20 mA
- 4–20 mA
- 10–50 mA.

Området 0–5 V er spesielt utbredt siden 5 V er en mye brukt foryningsspenning i mikrokontrollerbaserte systemer. Moderne mikrokontrollere kan ofte drives av enda lavere spenning, f.eks. 3,3 V, noe som også gjør spenningsområdet 0–3,3 V meget aktuelt. En viktig ulempe med disse spenningsområdene er imidlertid at det er svært vanskelig å avgjøre når en spenning er nøyaktig lik 0 V i et system der dette er den laveste forekommende spenningen. Derfor velger vi ofte å representere laveste signالنivå med en positiv spenning.

Eksempel:

Betrakt et termometer som er kalibrert slik at 0°C gir signalverdien 0 V og 100°C gir 5 V. Kretsen er forsynt fra en enkelt 5 V spenningskilde slik at utgangssignalet aldri kan være negativt. Hvis vi leser 0 V betyr det enten at vi er ved 0°C eller lavere, at termometerets utgangskrets er defekt, at selve sensoren er ødelagt og gir et feilaktig signal, eller vi kan rett og slett ha et ledningsbrudd. Vi er imidlertid ikke i stand til å avgjøre hvilken av disse situasjonene som har inntruffet.

Hvis vi i stedet bruker et 1–5 V-signal slik at 1 V tilsvarer 0°C, kan vi se at temperaturen begynner å gå under 0°C fordi spenningssignalet går til (evt. under) 1 V. Vi kan videre benytte en alarmgrense nær 0 V (dvs. en “ulovlig” spenningsverdi) til å varsle om at spenningsforsyningen svikter eller at en annen feil har inntruffet.

Dette fenomenet kalles *hevet nullpunkt* (eng.: *elevated zero*), og er den foretrukne metoden i industrien.

## 4.4 Digitalisering av analoge signaler

Figur 4.6 oppsummerer de viktigste elektroniske enhetene som inngår i et typisk system for innsamling av målesignaler. Som antydnet i figuren er det  $n$  analoge signaler



**Figur 4.6** Typiske enheter i et datainnsamlingssystem for  $n$  analoge signaler.

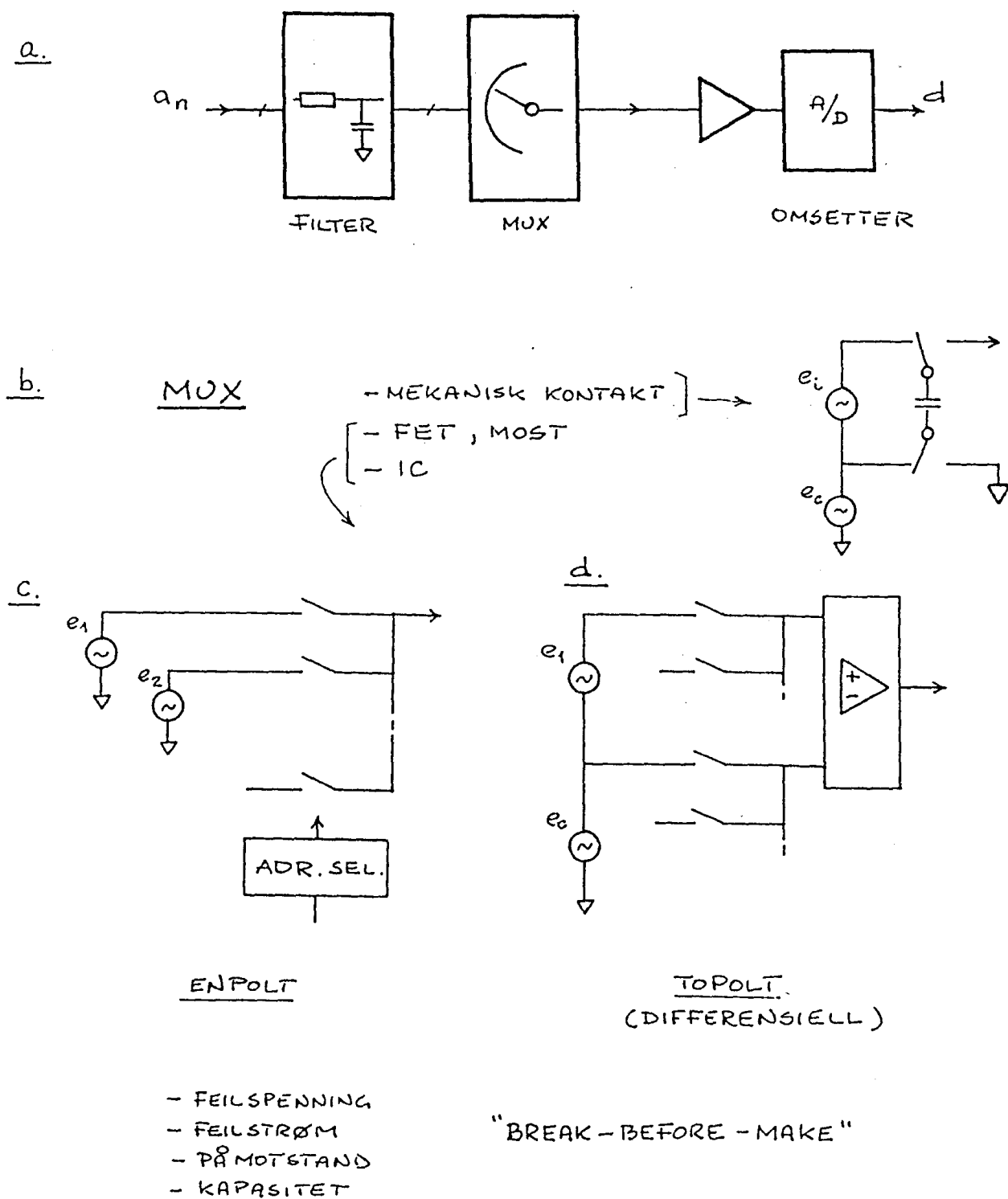
som filtreres med hvert sitt analoge filter, for deretter å bli matet inn i en analog multiplekser og så videre. I det følgende skal vi se nærmere på flere av disse enhetene og spesielle forhold rundt dem som vi må ta hensyn til ved design av systemet. Vi skal også ta for oss noen nyttige elementer og teknikker som ikke fremgår av figuren.

### 4.4.1 Filter mot nedfolding og overspenning

Avlesning via multiplekser er en form for tasting av signalet, og vi må sørge for at signalkomponenter høyere enn halve tastefrekvensen er fjernet ved lavpassfiltrering før multipleksingen slik at vi unngår nedfolding (dette forklares nærmere i avsnitt 4.4.7). Vanligvis greier vi oss med enkle RC-filtre foran multiplekseren, som vist i figur 4.7a. Filtermotstandene vil også virke som strømbegrensere om det kommer overspenning inn på signallinjene. Slik beskyttelse er viktig for å hindre at lokale feil på en inngangskanal ødelegger utstyret lenger innover i systemet eller forstyrrer signalinnlesningen på de kanalene som er i orden.

### 4.4.2 Analog multiplekser

Multiplekserens oppgave er å kople ett av flere (typisk 8 eller 16) analoge signaler til A/D-omsetteren. Dette kan gjøres med mekaniske kontakter, transistorer eller med spesielle integrerte multiplekserkretser. Utfordringen er å unngå å forurense målesignalene med elektrisk støy (feilspenninger og feilstrømmer) fra multipleksingen, spesielt dersom den foregår på et lavt signalnivå ( $\mu$  V-mV). Ved lave signalnivåer er mekaniske kontakter de beste fordi de har meget lav impedans, men de har begrenset operasjonshastighet (10 ms) og levetid (typisk noen titalls millioner koplinger). Halv-

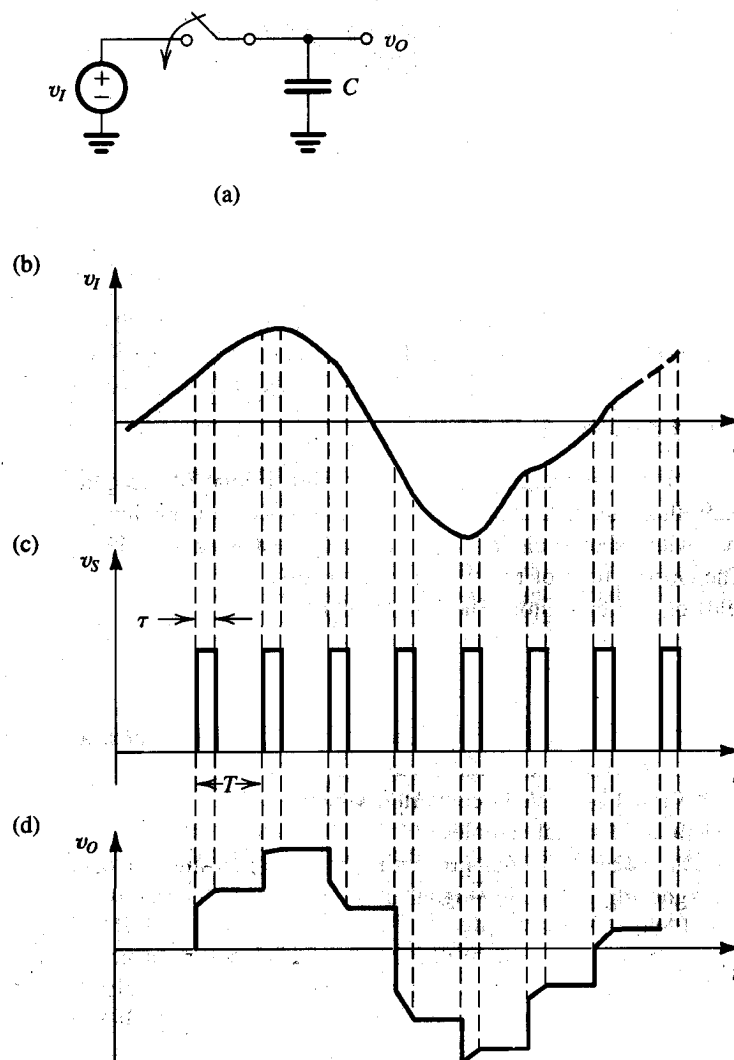


Figur 4.7 Innlesning av analogdata.

lederelementer har ikke disse begrensningene, men er nøye på at signalene ikke går utenfor kraftforsyningsgrensene, og de må også beskyttes bedre mot skadelige over-spenninger og -strømmer.

#### 4.4.3 Tast-og-hold-krets

Et underliggende prinsipp for digital signalbehandling er *tasting* (engelsk: *sampling*) av det analoge signalet. Figur 4.8 illustrerer dette konseptet. Kretsen i Figur 4.8 er kjent som en *tast-og-hold- (T/H)-krets*. Som antydnet består kretsen av en analog bryter som kan implementeres ved hjelp av transistorer, og en lagringskondensator. Ved tastetids-punktet sluttet bryteren i et relativt kort tidsintervall, *tasteintervallet*  $\tau$ , slik at kondensatoren lades opp til inngangssignalets øyeblikksverdi. Når bryteren åpnes, vil spenningen over kondensatoren holde seg konstant. Bryteren realiserer dermed taste-elementet, og kondensatoren holdeelementet.



**Figur 4.8** Prosess for periodisk tasting et analogt signal. (a) Tast-og-hold (T/H)-krets. Bryteren slutter i  $\tau$  s hver periode. (b) Inngangssignalets bølgeform. (c) Tastesignal (styringssignal for bryteren). (d) Utgangssignal (mates til A/D-omsetter).

Mellom tasteintervaller - det vil si i *holdeintervallene* - representerer spenningsnivået på kondensatoren signalnivået vi er ute etter. Dette spenningsnivået mates til inngangen på en A/D-omsetter som gir et N-bits binært tall proporsjonal med det tastede signalnivået (selve A/D-omsetteren er tema for et senere avsnitt). Noen typer A/D-omsettere trenger en viss tid på å omsette signalet til digital form, og i denne tiden må signalet holdes konstant for å oppnå et riktig resultat; dette er den grunnen til at taste-og-hold-kretsen benyttes.

Tast-og-hold-kretser har mange karakteristiske egenskaper, og vi skal ta for oss de viktigste:

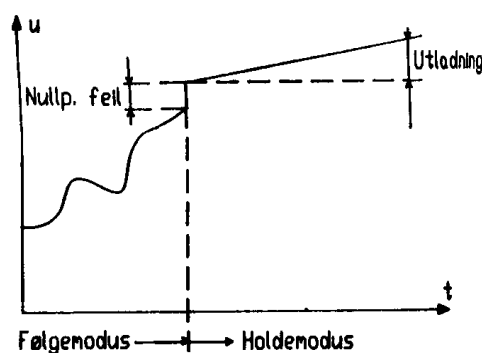
**Holdemodus** – Bryteren er åpen, og ideelt sett skal ikke utgangsverdien påvirkes av signalet på tasterens inngang.

**Følgemodus** – Bryteren er sluttet slik at holdeelementets utgang følger inngangssignalet.

**Innsvingningstid (Acquisition time)** – Den tid det tar for spenningen over kondensatoren å svinge seg inn til riktig verdi, med en gitt toleranse, når inngangsverdien endres i sprang fra nedre til øvre grense. Innsvingningstiden begrenses hovedsakelig av bryterens resistans, som sammen med kondensatoren danner en tidskonstant.

**Utladningshastighet (Droop rate)** – Selv om kondensatoren etterfølges av en forsterker, endres spenningen over den etter en tid selv om bryteren er åpen. Dette skyldes lekkasjestrømmer inne i kretsen. Utgangsspenningen kan både øke og minke, og endringen kan være ulineær med tiden. Utladningshastigheten oppgis i [V/s] ved romtemperatur, og fordobles når temperaturen øker 10°C.

**Tast-til-hold nullpunktsfeil (Sample to hold offset)** – I T/H-kretser brukes elektroniske brytere. De har en liten sprekondensator mellom selve bryteren og den elektroden som styresignalet er koblet til. Når styresignalet endrer seg, blir det derfor injisert en liten ladning gjennom den. Injeksjonsladningen flytter til holdekondensatoren og fører til spenningsendring over den. Nullpunktsfeilen er forskjellen mellom spenningen over holdekondensatoren rett før bryteren åpnes og den verdi den svinger seg inn til etter at bryteren er åpnet. Se figur 4.9.



**Figur 4.9** Tast-til-hold nullpunktsfeil.



#### 4.4.4 Flying Capacitor

Det er enklest om målesignalene er referert til en felles signalgjord, for da kan vi greie oss med en enpolet multiplekser. Dersom målesignalene flyter på en stor common mode-spenning  $e_c$  som vist i figur 4.7b og c, må vi ha to koplingselementer per målesignal, og en differensialforsterker må ta hånd om signalene etterpå. Dette common mode-problemet kan imidlertid løses elegant ved hjelp av en kondensator som mellomlagrer signalverdien ("flying capacitor") som vist i figur 4.7b. Det differensielle inngangssignalet  $e_i$  lader opp kondensatoren, og når bryterne slår over til høyre legger vi den ene siden på kondensatoren til jord og får dermed ut et enpolet signal, helt uavhengig av common mode-spenningen  $e_c$ .

#### 4.4.5 Selvsjekk

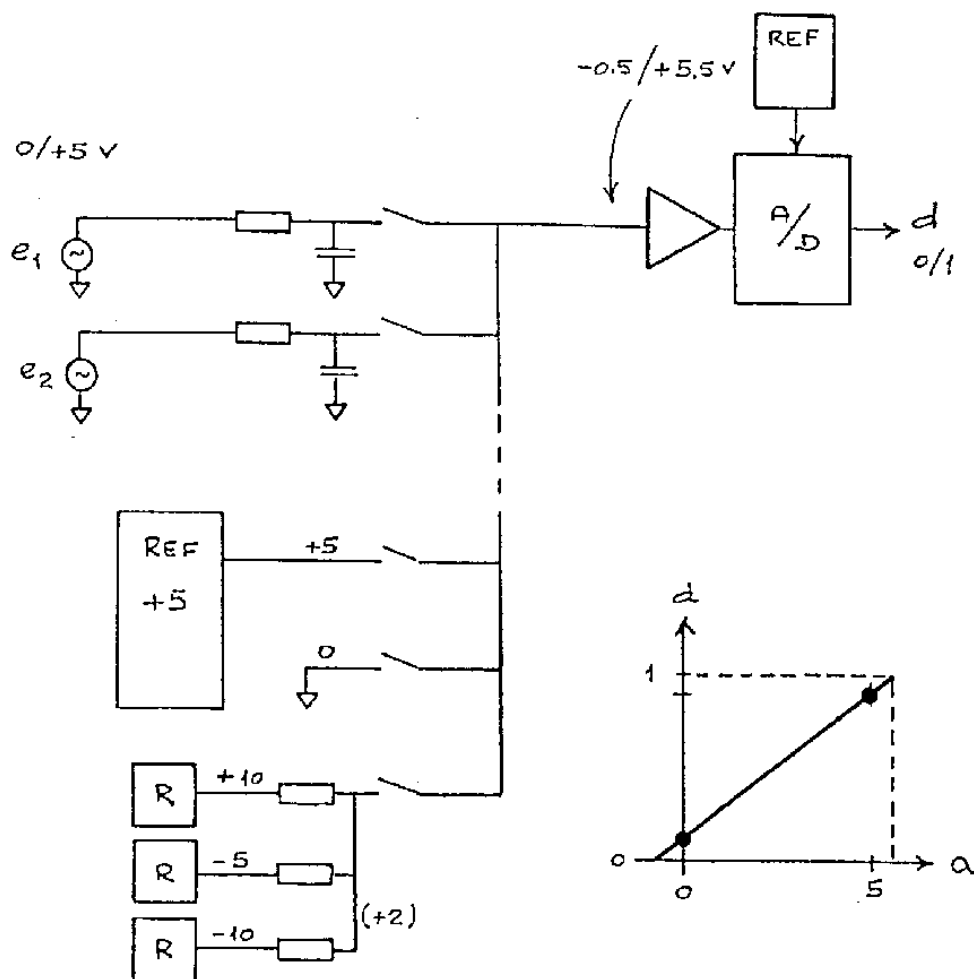
I målesystemer er det viktig at unøyaktigheter og feil blir oppdaget og korrigert for, eller rapportert av det sentrale datautstyret. I multipleksingen legger vi derfor inn faste og kjente signaler på enkelte kanaler, og datautstyret kan da med sine avlesninger gjøre en selvsjekk av det etterfølgende utstyret. Kretsene på den enkelte målekanal foran multiplekseren blir imidlertid uten overvåking, og det er derfor viktig at denne delen av systemet lages enkel og robust og med gode marginer mot støy og andre feil.

Figur 4.10 illustrerer noen av disse tankene: øverst i multiplekseren har vi en del nyttekanaler, mens vi nederst har lagt inn to kalibreringsspenninger på henholdsvis +5 V og 0 V. Måling på disse gir en sjekk av A/D-omsetteren og dens referansespenning. Her er det viktig at en ikke bruker samme referansespenningskilde for kalibreringssignalene og A/D-omsetteren fordi en feil i referansespenningen da ikke vil bli oppdaget.

I forbindelse med målesystemet er det ofte bruk for mange lokale spenningsregulatorer som gir matespenning til målekretsene. Det er vanligvis for kostbart å gi hver slik regulator en egen målekanal på multiplekseren. Vi kan imidlertid lage en billigere løsning som vist nederst i figuren, ved at flere signalspenninger midles sammen i et motstandsnettverk og koples til en enkelt multiplekserkanal. Dersom en av regulatorene gir ut unormal spenning, vil spenningen ut av nettverket avvike fra normalverdien, og feilsituasjonen kan oppdages.

#### 4.4.6 A/D-omsetters arbeidsområde

A/D-omsetteren bør ha et arbeidsområde som strekker seg en del utenfor det aktuelle signalområdet for inngangssignalene. Vi har da litt å gå på når systemet utfører selvkalibrering på grunnlag av innleste referansespenninger, og vi kan gjennomføre en digital signalmidling på støyfulle inngangssignaler selv om de ligger i kanten av vanlig signalområde. Dersom vi lot A/D-omsetteren gå i metning på støytoppene ("asymmetrisk metning") i signalet ville vi nemlig få en likerettereffekt, og dette ville i den digitale midlingen gi en forskyvning av middelverdien av signalet.



Figur 4.10 Selvsjekk

#### 4.4.7 Samplingsteoremet og nedfolding

Et grunnleggende spørsmål i sammenheng med signaldigitalisering er: Hvor ofte må vi taste for å fange opp all informasjonen i et analogt signal? Svaret på dette er gitt av følgende teorem:

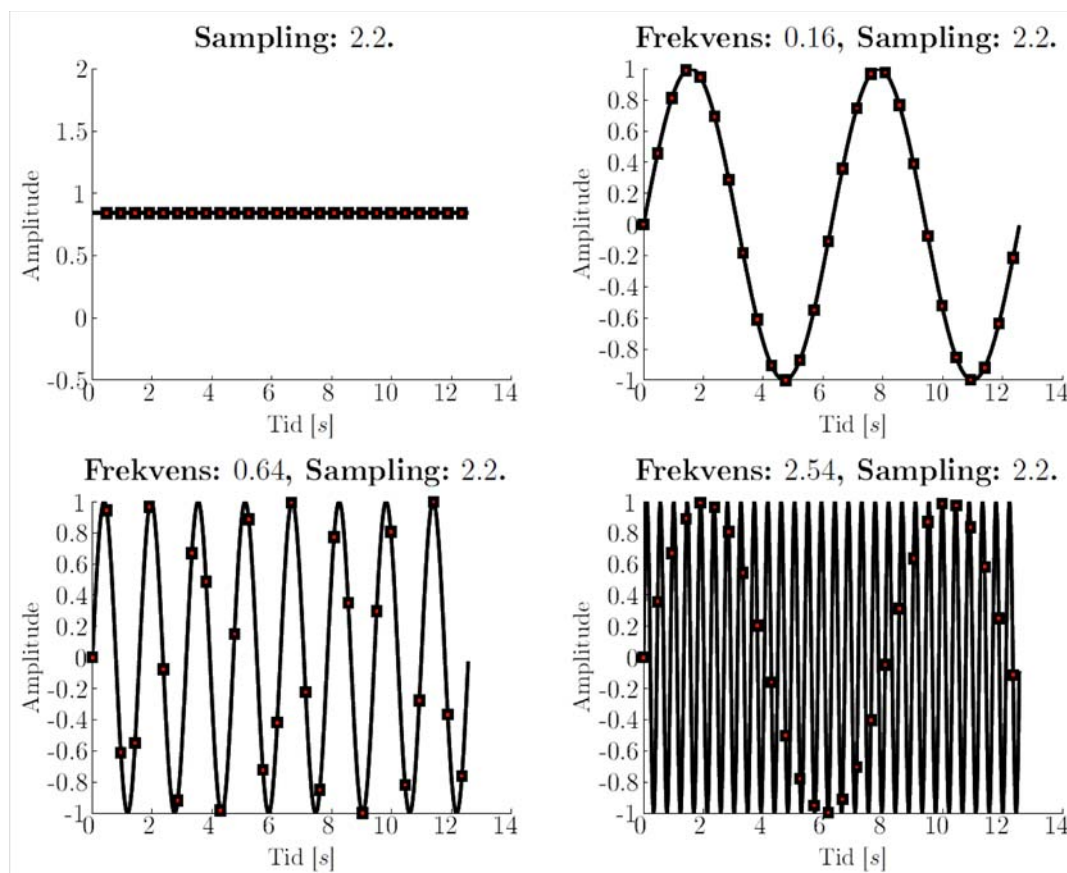
##### *Nyquist-Shannons samplingsteorem*

Et analogt signal kan representeres med det digitale signal, og rekonstrueres fra dette, hvis tastefrekvensen  $f_s$  er minst dobbelt så høy som den høyeste frekvensen<sup>1</sup>  $f_{max}$  i signalet,

$$f_s \geq 2f_{max}. \quad (27)$$

1. Denne formuleringen gjelder for signaler som inneholder relevante frekvenser helt ned til 0 Hz, noe som er tilfellet for de aller fleste kybernetiske systemer. For generelle signaler gjelder at tastefrekvensen må være minst dobbelt så høy som signalets båndbredde, dvs. forskjellen mellom høyeste og laveste frekvens i signalet.

Frekvensen  $f_{max}$  kalles for signalets Nyquistfrekvens. Hva som skjer hvis vi taster med for lav frekvens er illustrert i figur 4.11, der samme frekvens er brukt for å sample



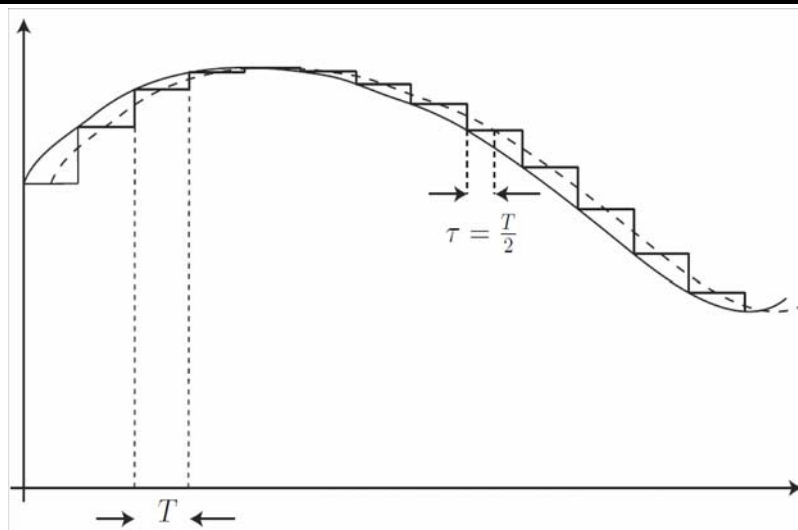
**Figur 4.11** Konsekvensen av å bruke samme tastefrekvens på signaler med ulike frekvenser.

sinus-signaler med ulike frekvenser. Nederst til høyre ser vi hva som skjer hvis vi taster et 2,54 Hz-signal med tastefrekvensen 2,2 Hz. Sinussignalet som oppstår i det digitaliserte signalet har en lavere frekvens enn originalsignalet. Dette fenomenet kalles *nedfolding* (engelsk: *aliasing*), og er noe vi i de aller fleste tilfeller må sørge for å unngå.

Merk at det ligger noen forutsetninger til grunn for samplingsteoremet. For det første må signalet være båndbreddebegrenset, det vil si at det ikke må inneholde noen frekvenser over en gitt  $f_{max}$ , og for det andre må vi i taster signalet uendelig mange ganger (dvs. uendelig lenge) for å kunne gjenskape det nøyaktig fra den digitale representasjonen. Den siste forutsetningen er det umulig å oppfylle i praksis. Båndbreddebegrensningen kan vi tilnærme ved å filtrere signalet med et analogt lavpassfilter som demper de høyeste signal- (og støy-) frekvensene. Et slikt filter kalles et *anti-nedfoldingsfilter* (eng.: *anti aliasing filter*). I alle tilfeller er det en god regel å taster signalet en del raskere enn det samplingsteoremet krever, for eksempel å velge  $f_s = 10f_{max}$ .

#### 4.4.8 Tasting gir tidsforsinkelse

En effekt av tasting er at hvis vi glatter ut det tastede signalet, får vi et signal som er tidsforsinket med en halv tasteperiode. Dette er illustrert i figur 4.12, og det kan være viktig å ta hensyn til denne forsinkelsen i design av reguleringsystemer.



**Figur 4.12** Tasting medfører tidsforsinkelse.

#### 4.4.9 A/D- og D/A-omsettere – sentrale begreper

Her skal vi raskt ta for oss noen viktige begreper med direkte relevans både for A/D- og D/A-omsettere. Analoge begreper gjelder nødvendigvis A/D-omsetters inngang og D/A-omsetters utgang, og motsatt for digitale begreper.

**Arbeidsområde** – angivelse av henholdsvis laveste og høyeste analoge verdi som omsetteren kan håndtere.

**Omfang** – differansen mellom høyeste og laveste analoge verdi som omsetteren kan håndtere. Omfanget er med andre ord størrelsen på arbeidsområdet.

**Oppløsning** – for en D/A-omsetter er oppløsningen den minste verdien vi kan endre utgangssignalet med; for en A/D-omsetter er det størrelsen på omsetters kvantiseringsintervall (se nedenfor).

Oppløsningen betegnes gjerne med enheten LSB (Least Significant Bit), fordi den representerer en spenningsendring tilsvarende nettopp en endring av omsetters minst signifikante bit. Hvis omsetteren har  $N$  bit og omfanget betegnes  $O$ , er oppløsningen gitt av

$$LSB = \frac{O}{2^N} [V]. \quad (28)$$

**Kvantisering** – det fenomenet at den digitale signalrepresentasjonen bare kan representere et fast og endelig sett av signalverdier.

Betrakt et analogt signal med signalnivå i området 0–10 V. La oss anta at vi ønsker å konvertere dette signalet til digital form, og at ønsket resultat er et 4-bits signal. Et 4-bits binært tall kan representere 16 forskjellige verdier fra 0 til 15. Det følger at oppløsningen av konverteringen vår vil bli  $10\text{ V}/15 = 2/3\text{ V}$ . Dermed vil et analogt signal på 0 V være representert med 0000,  $2/3\text{ V}$  vil være representert ved 0001, 6 V vil være representert med 1001, og 10 V vil være representert ved 1111.

Alle de ovennevnte signalverdiene var multipler av det grunnleggende inkrementet ( $2/3\text{ V}$ ). Et spørsmål melder seg angående konvertering av tall som faller mellom påfølgende inkrementelle nivåer. Betrakt for eksempel et analog nivå på 6,2 V. Dette faller mellom  $18/3\text{ V}$  og  $20/3\text{ V}$ . Siden det er nærmest  $18/3\text{ V}$  vil vi behandle det som om det var 6 V og kode det som 1001. Denne prosessen kalles kvantisering. Feil er åpenbart uunngåelig i denne prosessen, og slike feil kalles *kvantiseringsfeil*. Ved å bruke flere bit til å representere (kode) det analoge signalet, reduseres kvantiseringsfeilen, men dette krever en mer kompleks krets.

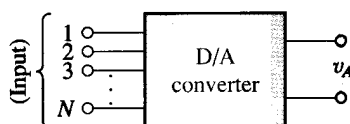
**Dynamisk område** – området er gitt av forholdet mellom omfanget og oppløsningen, regnet i desibel. Sagt på en annen måte, og noe upresist: det dynamiske området er forholdet mellom den største og den minste signalverdien omsetteren kan håndtere.

For en  $N$ -bits omsetter er det dynamiske området:

$$20 \cdot \log(1/2^{-N}) = N \cdot 20 \cdot \log(2) \approx 6 \cdot N \quad (29)$$

## 4.5 Kretser for D/A-omsetting

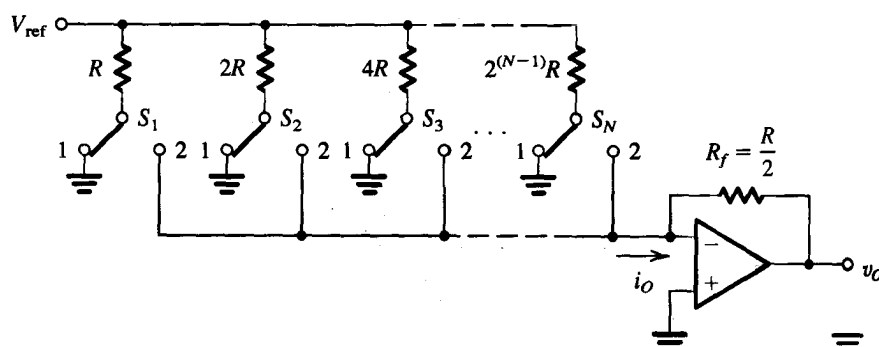
I figur 4.13 ser vi igjen D/A-omsetteren illustrert som en komponent på registernivå, her med  $N$  parallelle binære innganger. D/A-omsettersens inngangsdelen er en logisk registernivåkomponent på alle måter, og den finnes i alle tenkelige varianter med forskjellige antall bit, parallell eller seriell og synkron eller asynkron innlasting, og så videre. Utgangsdelen av kretsen er derimot selvsagt analog, og vi skal nå se på to konkrete kretsløsninger for realisering av denne analoge delen.



**Figur 4.13** D/A -omsetteren som registernivåkomponent.

### 4.5.1 Binærvektet stigenettverk

Figur 4.14 viser en enkel krets som realiserer en  $N$ -bits D/A-omsetter. Kretsen består av en referansespenning  $V_{\text{ref}}$ ,  $N$  binær-vektede motstander  $R, 2R, 4R, 8R, \dots, 2^{N-1}R$ ,  $N$  énpoledede vendebrytere  $S_1, S_2, \dots, S_N$ , og en op-amp med tilbakekoplingsmotstand  $R_f = R/2$ .



**Figur 4.14** En  $N$ -bits D/A-omsetter realisert med binærvektet resistivt stigenettverk.

Bryterne styres av et  $N$ -bits digitalt inngangssord  $D$ ,

$$D = \frac{b_1}{2^1} + \frac{b_2}{2^2} + \dots + \frac{b_N}{2^N} \quad (30)$$

der  $b_1, b_2$ , osv. er bitkoeffisienter som er enten 1 eller 0. Merk at bit  $b_N$  er det *minst signifikante bit* (LSB) og  $b_1$  er det *mest signifikante bit* (MSB). I kretsen i Figur 4.14 styrer  $b_1$  bryteren  $S_1$ ,  $b_2$  styrer  $S_2$ , og så videre. Når  $b_i$  er 0, er bryter  $S_i$  i posisjon 1, og når  $b_i$  er 1, er bryter  $S_i$  i posisjon 2.

Siden posisjon 1 for alle brytere er jord og posisjon 2 er virtuell jord (op-amp'ens negative inngang), vil strømmen gjennom hver motstand være konstant. Hver bryter styrer rett og slett hvor strømmen skal gå: til jord (når den tilsvarende bit er 0) eller til virtuelle jord (når den tilsvarende bit er 1). Strømmene som flyter inn på virtuell jord summeres opp, og summen flyter gjennom tilbakekoplingsmotstanden  $R_f$ . Den totale strømmen  $i_O$  er derfor gitt ved

$$i_O = \frac{V_{\text{ref}}}{R} b_1 + \frac{V_{\text{ref}}}{2R} b_2 + \dots + \frac{V_{\text{ref}}}{2^{N-1}R} b_N = \frac{2V_{\text{ref}}}{R} \left( \frac{b_1}{2^1} + \frac{b_2}{2^2} + \dots + \frac{b_N}{2^N} \right) \quad (31)$$

Med andre ord har vi at

$$i_O = \frac{2V_{\text{ref}}}{R} D, \quad (32)$$

og utgangsspenningen  $v_O$  er gitt av

$$v_O = -i_O R_f = -V_{ref} D \quad (33)$$

som er direkte proporsjonal med det digitale ordet  $D$ , med andre ord nettopp det vi ønsker.

Merk at D/A-omsetterens nøyaktighet avhenger kritisk av

1. nøyaktigheten til  $V_{ref}$ ,
2. nøyaktigheten til de binærvektede motstandene, og
3. bryternes egenskaper.

Når det gjelder det tredje punktet, bør vi understreke at disse bryterne håndterer analoge signaler, og dermed er deres karakteristikk av betydelig interesse. Mens offsetspenning og motstand ikke er av avgjørende betydning i en digital bryter, er disse parametrene av enorm betydning i analoge brytere.

En ulempe med det binærvektede motstandsnettverket er at for et stort antall bits ( $N > 4$ ) vil spredningen mellom de minste og største motstandene bli ganske store. Dette gjør det vanskelig å opprettholde nøyaktigheten i motstandsverdiene. En mer praktisk løsning er å benytte et resistivt nettverk kjent som  $R$ - $2R$  stigenettverk.

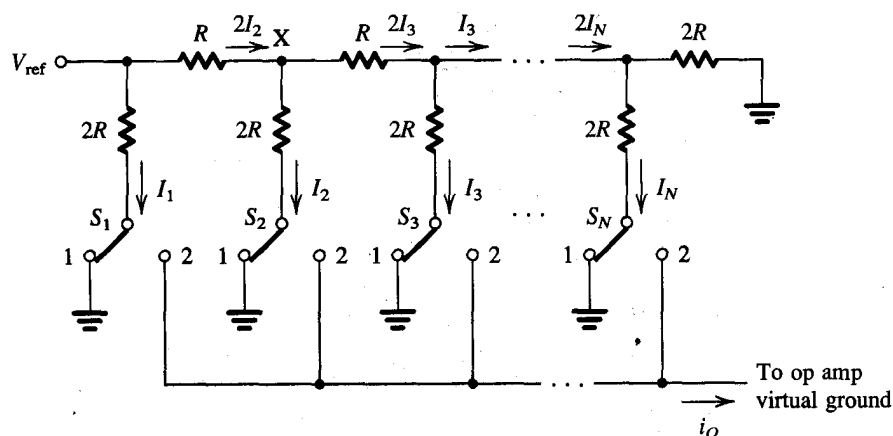
#### 4.5.2 $R$ - $2R$ -stigenettverk

Figur 4.15 viser den grunnleggende strukturen i en DAC basert på  $R$ - $2R$ -stigenettverk. På grunn av den begrensede spredningen i motstandsverdier er dette nettverket vanligvis foretrukket fram for den binærvektede løsningen omtalt ovenfor, spesielt for  $N > 4$ . Virkemåten til  $R$ - $2R$ -stigen er enkel. Ved å starte fra høyre og gå mot venstre kan det vises at motstanden til høyre for hver node i stigen, for eksempel noden merket  $X$ , er lik  $2R$ . Dermed er strømmen som flyter mot høyre, bort fra hver node, lik den som flyter nedover mot jord, og to ganger denne strømmen flyter inn i noden fra venstre side. Det følger at

$$I_1 = 2I_2 = 4I_3 = \dots = 2^{N-1}I_N \quad (34)$$

Strømmene som styres av bryterne er med andre ord binært vektet, akkurat som i det binærvektede stigenettverket. Utgangsstrømmen  $i_O$  er derfor gitt av

$$i_O = \frac{V_{ref}}{R} D \quad (35)$$



**Figur 4.15** Prinsippskisse av en D/A-omsetter basert på  $R$ - $2R$ -stigenettverk.

## 4.6 Kretser for A/D-omsetting

There exist a number of A/D conversion techniques varying in complexity and speed of conversion. In the following, we shall discuss two simple but slow schemes, and one complex (in terms of the amount of circuitry required) but extremely fast method.

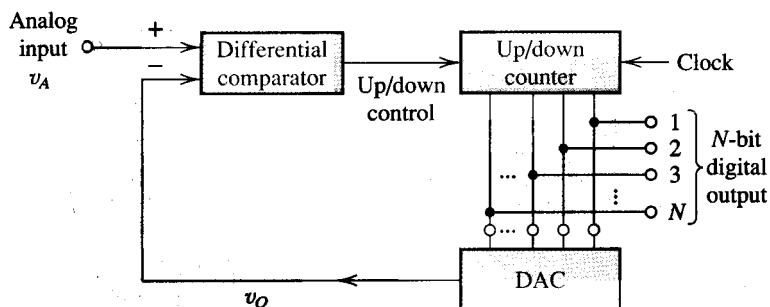
Det finnes en rekke teknikker for A/D-omsetting som varierer i kompleksitet og konverteringshastighet. I det følgende skal vi diskutere to enkle, men langsomme løsninger, samt en kompleks men svært rask krets.

### 4.6.1 Tilbakekoplet A/D-omsetter (servoomsetter)

Figur 4.16 viser en enkel A/D-omsetter som benytter en komparator, en opp-ned-teller, og en D/A-omsetter. Komparatoren gir en utgang som antar én av to forskjellige verdier: positiv når differensen mellom inngangssignalene er positiv, og negativ når differensen er negativ. En opp-ned teller er bare en teller som kan telle opp eller ned avhengig av det binære nivået påtrykt dens styringsterminal. Fordi A/D-omsetteren i Figur 4.16 benytter en DAC i en tilbakekoplet sløyfe, er det vanlig å kalle kretsen for en tilbakekoplet A/D-omsetter eller servoomsetter (eng.: *feedback-type A/D converter*). Den fungerer som følger: Med tellerverdien 0 i telleren, vil D/A-omsetterens utgang,  $v_o$ , være null, og komparatorutgangen vil være høy, noe som får telleren til å øke sin tellerverdi med én hv gang den påtrykkes en klokkepuls. Ettersom tellerverdien øker, stiger den analoge utgangsverdien til D/A-omsetteren. Prosessen fortsetter til utgangssignalet så vidt har passert verdien av det analoge inngangssignalet. Telleverkets verdi vil da være det digitale motstykket til den input analoge spenningen.

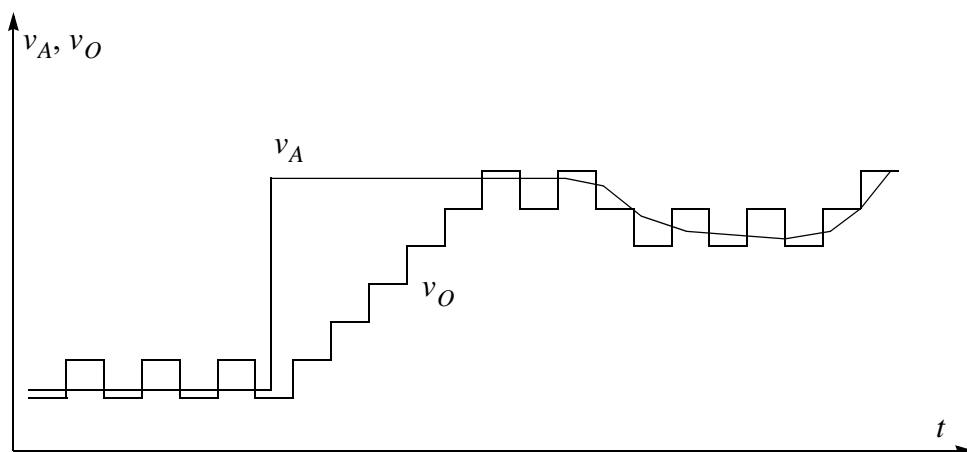
En spesiell egenskap ved denne omsetteren er at utgangen aldri vil falle til ro selv om inngangssignalet er konstant; når utgangssignalet har passert inngangsspenningens verdi på vei oppover, snur tellerens retning slik at den teller et skritt *nedover* igjen. Dette gjør at utgangen igjen faller under inngangsspenningen, slik at telleren neste gang teller et skritt *oppover*, og slik fortsetter omsetterens utgang å oscillere mellom det nærmeste kvantiseringsnivået over og under inngangssignalets verdi.





**Figur 4.16** En enkel tilbakekoplet A/D-omsetter.

Kretsen i Figur 4.16 kan bruke relativt lang tid på å “svinge seg inn” mot riktig signalverdi hvis den starter fra null. Denne omsetteren følger imidlertid små endringer i inngangssignalet ganske raskt. Figur 4.17 illustrerer servoomsetterens respons på et varierende inngangssignal.



**Figur 4.17** Servoomsetterens respons.

#### 4.6.2 Dobbel-rampe-omsetter

En svært populær A/D-omsetterløsning som kan gi høy oppløsning (12 - til 16-bit), er illustrert i Figur 4.18. Øverst til venstre i figuren ses den analoge inngangsspenningen  $v_A$  som skal digitaliseres. Kretsen omfatter også en analog integrator som beregner tidsintegralet av spenningen  $v_1$ . Bryteren  $S_1$  brukes til å velge om det er inngangssignalet  $v_A$  eller en fast referansespenning  $V_{ref}$  som skal integreres. Integratoren kan nullstilles ved at bryteren  $S_2$  lukkes et øyeblikk slik at kondensatoren  $C$  lades ut. Videre har kretsen en analog komparator som detekterer når integratorens utgangsspenning  $v_2$  passerer 0 V. Komparatorutgangen er koplet til en styringslogikk (“Control logic”), en digital tilstandsmaskin som styrer bryterne  $S_1$  og  $S_2$ . Styringslogikken styrer også en digitale teller (“Counter”) som presenterer det digitale sluttresultatet (“Output”).

Anta at inngangssignalet er negativt ( $v_A < 0$  V)<sup>1</sup>. Før starten av konverteringssyklusen lukkes bryter  $S_2$  slik at vi får  $v_1 = 0$  V. Telleren nullstilles også. Konverteringssyklusens fase I begynner med at vi (dvs. styringslogikken) åpner  $S_2$  og kopler integratorens inngang til det analoge inngangssignalet  $v_A$  gjennom bryteren  $S_1$ . Siden  $v_A$  er negativ, vil en strøm  $I = v_A/R$  strømme gjennom  $R$  i retning bort fra integratoren. Dermed stiger  $v_1$  lineært med et stigningstall på  $I/C = v_A/RC$ , som angitt i Figur 4.18 (b). Samtidig er telleren aktivert, slik at den teller pulser fra en klokke med fast frekvens. Denne fasen av konverteringsprosessen pågår i et fast tidsintervall av lengde  $T_1$ , og slutter når telleren når en tilsvarende fast tellerverdi som vi betegner  $n_{ref}$ . For en  $N$ -bit konverter settes vanligvis  $n_{ref} = 2^N$ . Hvis vi kaller toppverdien til spenningen på integratorutgangen  $V_{peak}$ , kan vi med referanse til Figur 4.18 (b) skrive

$$\frac{V_{peak}}{T_1} = \frac{v_A}{RC} \quad (36)$$

Fase II av konverteringssyklusen starter ved  $t = T_1$  ved på nytt å resette telleren og å kople integratorens inngang gjennom  $S_1$  til den positive referansespenningen  $V_{ref}$ . Strømmen inn på integratoren sifter retning og er nå lik  $V_{ref}/R$ . Derfor vil  $v_1$  avta lineært med stigningstall  $V_{ref}/(RC)$ . Samtidig startes telleren, og teller pulser fra en klokke med fast frekvens. Når  $v_1$  når verdien 0 V, stoppes telleren på en tellerverdi som vi kaller  $n_A$ . Hvis vi betegner varigheten av fase II med symbolet  $T_2$ , kan vi skrive (jfr. Figur 4.18(b)):

$$\frac{V_{peak}}{T_2} = \frac{V_{ref}}{RC} \quad (37)$$

Likning (36) og (37) kombineres, og vi får

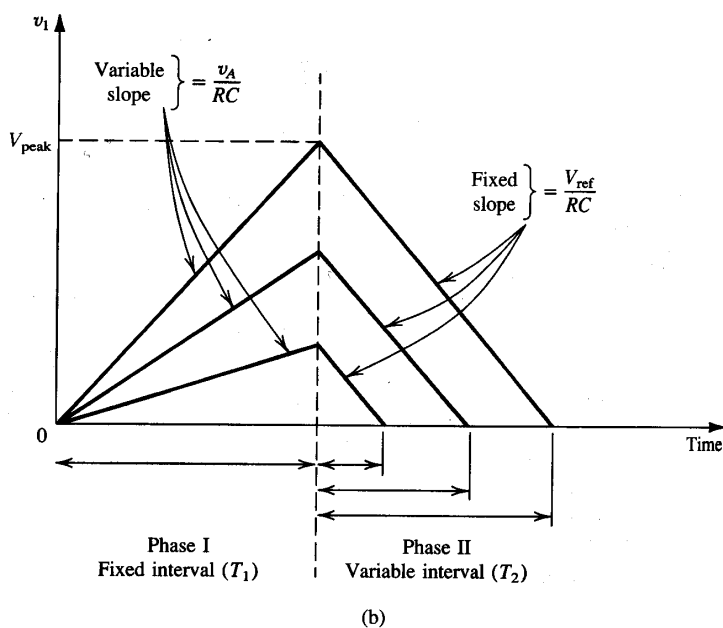
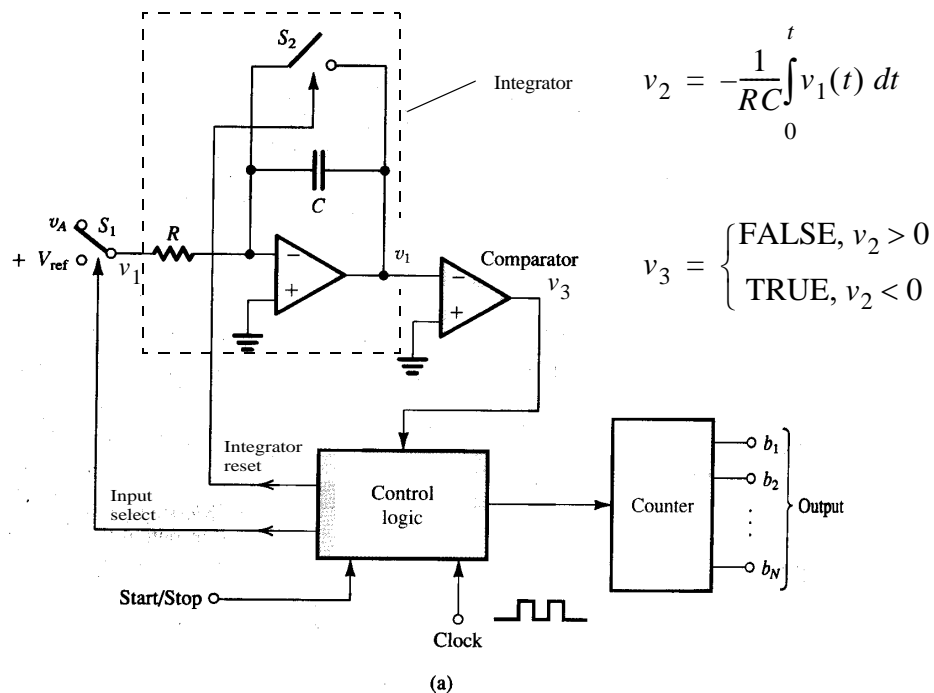
$$T_2 = T_1 \left( \frac{v_A}{V_{ref}} \right) \quad (38)$$

Siden tellerens verdi  $n_{ref}$  på slutten av fase I er proporsjonal med  $T_1$  og tellerverdien  $n$  på slutten av fase II er proporsjonal med  $T_2$ , har vi

$$n_A = n_{ref} \left( \frac{v_A}{V_{ref}} \right) \quad (39)$$

Tellerens verdi ved konverteringsprosessens slutt,  $n_A$ , er derfor den digitale ekvivalen-ten til inngangsspenningen  $v_A$ .

1. Kretsen kan lett modifiseres slik at den også kan behandle positive inngangsspenninger. Vi antar et negativt inngangssignal fordi det gir den enkelste kretsløsningen og gjør det lettere å få frem kretsens virkemåte.



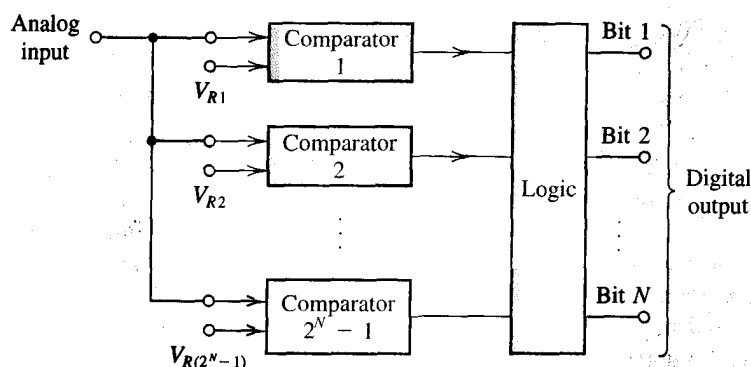
**Figur 4.18** The dual-slope A/D conversion method. Note that  $v_A$  is assumed to be negative.

Dobbel-rampe-omsetteren har stor nøyaktighet fordi dens ytelse *ikke* avhenger av den nøyaktige verdien til  $R$  og  $C$ . Imidlertid er den relativt langsom, fordi oppløsningen avhenger av  $n_{ref}$ ; jo flere bits oppløsning, jo lengre tid tar integrasjonen. Kretsen brukes derfor i systemer uten ekstreme krav til hurtighet, blant annet håndholdte måleinstrumenter ("multimetre").

Det finnes mange kommersielle implementasjoner av denne kretsen, som blant annet bruker ulike teknikker for å øke nøyaktigheten ytterligere eller for å øke kretsens arbeidsområde. På engelsk går denne dobbel-rampe-omsetteren blant annet under betegnelsene *Integrating ADC*, *Dual-slope* (eventuelt *Multi-slope*) *ADC* og *Dual Slope Integrator*.

### 4.6.3 Parallellomsetter

Den raskeste A/D-omsetteren vi skal studere her, er den *parallelle* omsetteren, eller *flash*-omsetteren, som er illustrert i Figur 4.19. Konseptuelt er parallellomsetteren veldig enkel. Den består av  $2^N - 1$  komparatorer som sammenlikner inngangssignalet med hver av spenningene som tilsvarer grensene mellom de  $2^N$  mulige kvantiseringsnivåene. Komparatorutgangene behandles av en rent kombinatorisk krets som produserer de  $N$  bit'en i det digitale utgangsordet. Merk at en komplett konvertering gjøres i løpet av en enkelt klokkesykel.

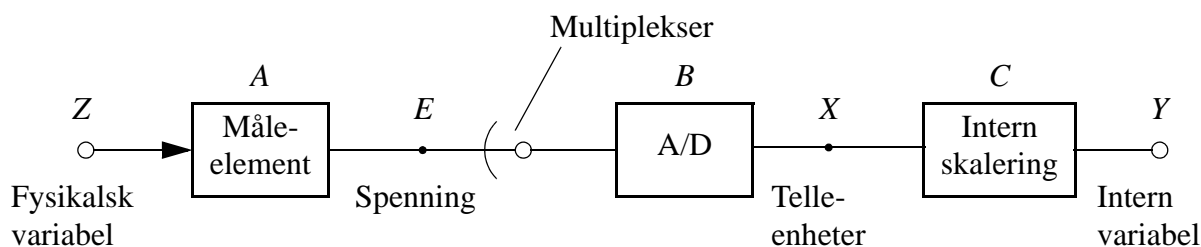


**Figur 4.19** Parallell- eller flash-A/D-omsetter.

Prisen for flash-omsetterens høye hastighet er en relativt kompleks kretsimplementasjon. Løsningen er likevel tilgjengelig i integrertkrets-versjon, og brukes i høyhastighetsanvendelser som oscilloskop, logikkanalysatorer og liknende.

## 4.7 Skalering og kalibrering

**Skalering** er en lineær transformasjon av for eksempel en måleverdi for tilpasning til et hensiktsmessig variasjonsområde.



**Figur 4.20** Omforminger fra fysikalsk variabel til intern representasjon.

Figur 4.20 gir et skjematisk bilde av de omforminger som skjer med en fysikalsk variabel til den er plassert i maskinen på en hensiktsmessig numerisk form. Generelt ønskes at  $Y$  skal være en lineær transformasjon av  $Z$ :

$$Y = K(Z - Z_0) + Y_0 \quad (40)$$

Vanligvis ønskes

$$Y = Z \quad (41)$$

som innebærer

$$\begin{aligned} K &= 1 \\ Y_0 &= Z_0 \\ Y_1 &= Z_1 \end{aligned} \quad (42)$$

der indeks 0 og 1 representerer henholdsvis nedre og øvre variasjonsgrense.

Måleelement og A/D-omsetter velges eller konstrueres slik at måleelementets utgangssignal,  $E$ , kan variere i *nesten* hele A/D-omsetters arbeidsområde for å utnytte omsetters oppløsning men samtidig unngå metning og resulterende tap av signalkvalitet. Om nødvendig konstrueres en krets som justerer signalets offset og forsterkning slik at vi oppnår dette.

Den interne skaleringen foregår i programvare, og går ut på å fastlegge funksjonen  $Y=f(X)$  i elementet  $C$  slik at likning (40), eventuelt (41), blir oppfylt, når måleelementets og A/D-omsetters egenskaper er gitt.

**Kalibrering** betyr generelt å fastlegge sammenhengen mellom den fysiske størrelsen  $Z$  som representeres av en variabel  $Y$  og korrekte numeriske verdier for denne variabelen. Kalibrering og skalering henger intimt sammen, men mens skaleringen gjerne skjer under systemutviklingen og går ut fra ideelle forhold og en ideell modell av målekjeden, vil kalibreringen ta sikte på å bestemme hvilke konstanter som *virkelig* inngår etter at skalering er foretatt og systemet er i bruk. Drift i forsterkere og passive kom-

ponenter kompenseres gjennom kalibrering. Kalibrering forutsetter målinger og sammenlikninger mot kjente fysiske størrelser og blir foretatt i et begrenset antall punkter. Forholdet mellom disse punkter antas kjent og blir ofte regnet som lineært. Siden kalibrering tar sikte på eliminering av ikke tilsiktede parametervariasjoner i målekjeden, blir kalibreringen ugyldig etter en tid og må derfor foretas med visse mellomrom. I sammenheng med skaleringen som gjennomgås i det følgende, går vi ut fra en ideell modell, og det er underforstått at systemet må kalibreres i tillegg.

#### 4.7.1 Skalering, lineære elementer

I første omgang antas at måleelement og A/D-omsetter er lineære. Som regel er A/D-omsetteren lineær, mens måleelementet (for eksempel en trykk- eller temperaturmåler) ofte kan være ulineært.

Hvis måleelementet er lineært, må også blokken  $C$  i figur 4.20 innebære en lineær funksjon, altså ren skalering. Med ulineært måleelement, kan blokken  $C$  også omfatte den inverse av den ulineære funksjonen slik at ulineariteten oppheves. Derved vil kravet om linearitet fra  $Z$  til  $Y$  bli oppfylt, i følge (40).

Funksjonene som representerer måleelement og A/D-omsetter dimensjoneres ut fra følgende regler:

Måleelementets inngang gis et variasjonsområde litt større enn og svarende best mulig til variasjonsområdet for  $Z$ . Måleelementets utgang  $E$ , sammen med en eventuell forsterker, gir ut en elektrisk spenning, standardisert for systemet og tilpasset multiplekserens og A/D-omsetterens arbeidsområde. Med lineært måleelement er

$$E = k_1(Z - Z_0) + E_0 \quad (43)$$

Går vi videre i rekken, har vi tilsvarende sammenheng mellom spenning  $E$  og den digitale utgang  $X$  for A/D-omsetteren:

$$X = k_2(E - E_0) + X_0 \quad (44)$$

$X$  er en digital størrelse i maskinen, gjerne oppfattet som et heltall.

Hvis  $X$  velges representert som et positivt heltall, vil et naturlig variasjonsområde være

$$0 \leq X < 2^N \quad (45)$$

hvor  $N$  er antall bits for omsetteren. Konstantene i (45) bestemmer området, og gir eksempelvis ved innsetting i (44):

$$X_0 = 0 \quad (46)$$

$$k_2 = \frac{2^N}{E_1 - E_0}$$

**Eksempel 4.1**

En 10-bits omsetter med arbeidsområde 0-10 V gir:

$$E_0 = 0$$

$$E_1 = 10$$

$$N = 10$$

$$\text{altså er } k = \frac{1024}{10} = 102,4$$

Sammenhengen mellom  $X$  og  $Y$  uttrykkes helt tilsvarende som de foregående blokker:

$$Y = k_3(X - X_0) + Y_0 \quad (47)$$

som kan trekkes sammen til:

$$Y = k_3X + B \quad (48)$$

Skaleringen innebærer fastleggelse av konstantene i likning (47) slik at de foran nevnte betingelser oppfylles. Kombineres likningene (43), (44) og (47), får vi

$$Y - Y_0 = k_3k_2k_1(Z - Z_0) \quad (49)$$

som sammenholdt med (40) gir

$$k_3 = \frac{K}{k_1k_2} \quad (50)$$

Med  $K = 1$ , svarende til likning (42), blir

$$k_3 = \frac{1}{k_1k_2} \quad (51)$$

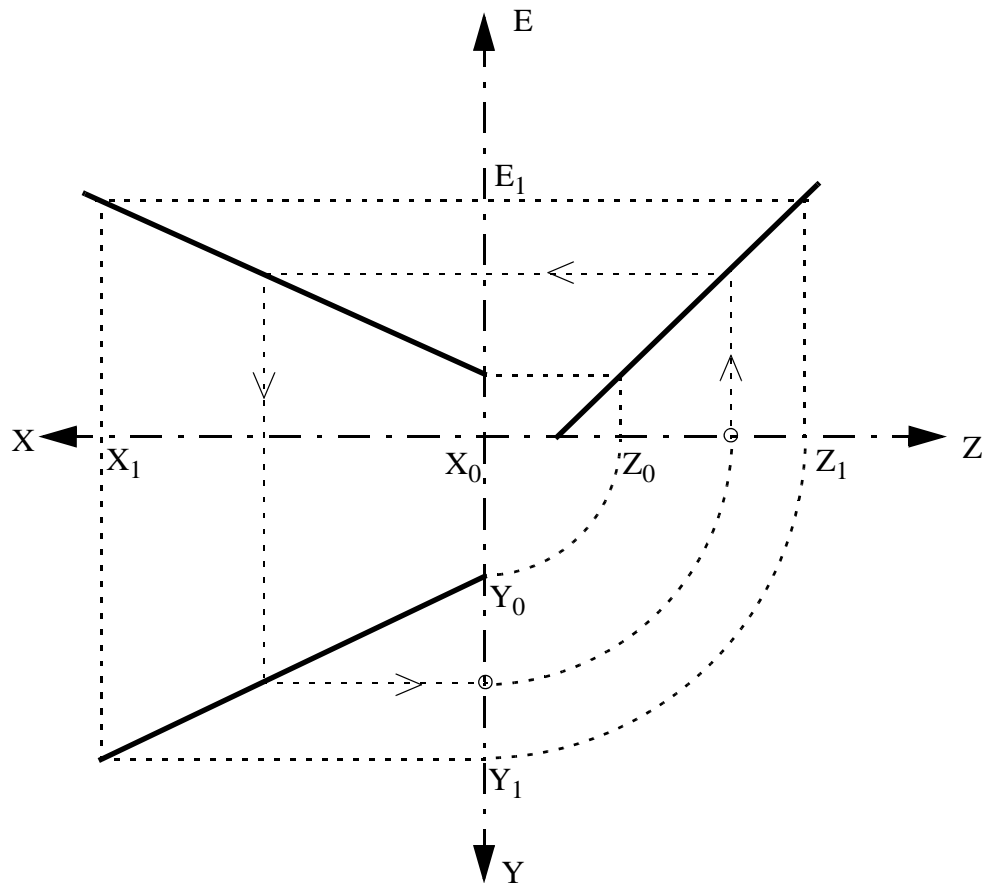
Overgangene kan illustreres i et sammensatt koordinatsystem som vist i figur 4.21.

Innsetting i (48) gir til slutt skaleringslikningen

$$Y = k_3X + B = \frac{K}{k_1k_2}(X - X_0) + Y_0 \quad (52)$$

Hvis (41) gjelder, blir skaleringslikningen enklere:

$$Y = \frac{1}{k_1k_2}(X - X_0) + Z_0 \quad (53)$$



**Figur 4.21** Overganger ved skalering fra målt variabel Z til regnestørrelse Y.

#### Eksempel 4.2

Trykkmåling:  $Z = p$  som varierer innen området  
 $0,2 \text{ kp/cm}^2 \leq p < 1,0 \text{ kp/cm}^2$

Måleelementet gir for dette trykkområde spenningsområdet  
 $1,0 \text{ V} \leq E < 9,0 \text{ V}$

Spenningen innleses med en 8-bits A/D-omsetter med utgang  $0 \leq X < 256$

Konstantene er (fra (43) og (44)):

$$k_1 = \frac{E_1 - E_0}{Z_1 - Z_0} = \frac{9,0 - 1,0}{1,0 - 0,2} = \frac{8}{0,8} = 10,0 \frac{\text{V}}{\text{kp/cm}^2}$$

$$k_2 = \frac{X_1 - X_0}{E_1 - E_0} = \frac{256 - 0}{8} = 32 \text{ V}^{-1}$$

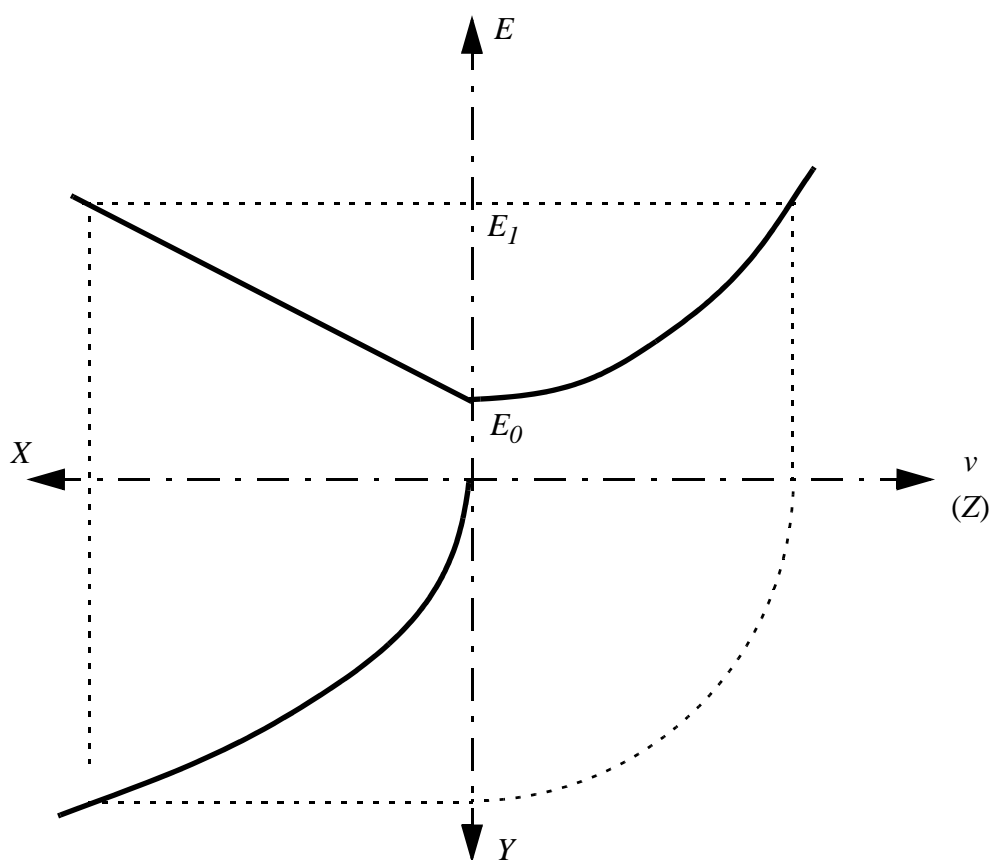


Hvis vi nå ønsker en intern representasjon med direkte numerisk samsvar med det målte trykk, slik at likning (40) gjelder, blir skaleringslikningen

$$Y = \frac{1}{320}X + 0,2 \text{ kp/cm}^2 \quad (54)$$

#### 4.7.2 Skalering, ulineært måleelement

Hvis måleelementet er ulineært men den ulineære funksjonen er én-entydig og dermed kan inverteres, er det fremdeles mulig å oppnå linearitet fra målt variabel  $Z$  til intern variabel  $Y$  ved å la blokken  $C$  omfatte en den inverse av den ulineære som måleelementet representerer.



**Figur 4.22** Overgang fra  $Z$  til  $Y$ , ulineært måleelement.

#### Eksempel 4.3

For måling av væskestrøm benyttes ofte en måleblende eller dp-celle. Disse elementer skaper et trykkfall proporsjonalt med kvadratet av strømningshastigheten. Elementets utgang kan være elektrisk, med spenning proporsjonal med trykkfallet over måleblenden. Utgangsspenningen  $E$  kan da uttrykkes:

$$E = \text{sign}(v)k_1v^2 + E_0 \quad (55)$$

hvor  $v$  er den fysikalske variable som ønskes innlest, altså  $v = Z$ . Figur 4.22 viser et diagram tilsvarende det som gjaldt for det lineære tilfelle. Dette gir oss en oversikt over overgangene frem til den interne variable  $Y$ .

Analogt det lineære tilfelle kan settes:

Måleelementets karakteristik:

$$E - E_0 = k_1 Z^2 \operatorname{sign}(Z) \quad (56)$$

A/D-omsetteren:

$$X - X_0 = k_2(E - E_0) = k_1 k_2 Z^2 \operatorname{sign}(Z) \quad (57)$$

Krever vi som tidligere likning (41) oppfylt, altså  $Y = Z$ , blir:

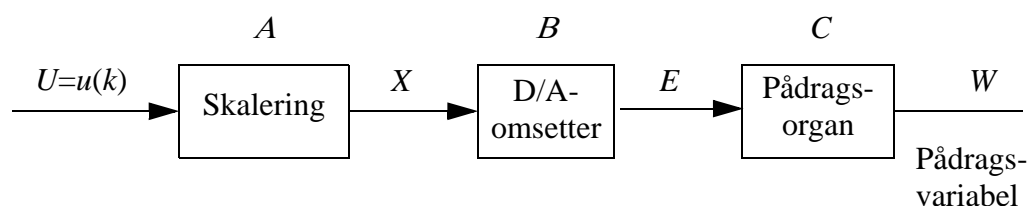
$$Y = \operatorname{sign}(X - X_0) \sqrt{\frac{X - X_0}{k_1 k_2}} \quad (58)$$

Blokk  $C$  omfatter altså en subrutine som beregner kvadratroten, hvilket selvsagt kunne innsees straks. I rutinen må velges korrekt fortegn, gitt av væskestrømmens retning, uttrykt ved signum-funksjonen i likning (58).

Det er ikke sikkert at måleelementets karakteristik er direkte inverterbar. For eksempel kan karakteristikken omfatte hysteres eller dødbånd. Slike tilfeller lar seg meget vanskelig forbedre med noen kompensering under innlesning.

#### 4.7.3 Skalering av ut-variable

Skalering må foretas ved utlesning på tilsvarende måte som ved innlesning. En intern variabel  $U$  som har en direkte numerisk relasjon til den aktuelle analoge pådragsvariable  $W$ , må omregnes til et signalområde passende for prosessdatautstyret (D/A-omsetteren og øvrige enheter som skal behandle signalet). La oss bruke det skjematisk bildet i figur 4.23 av en kanal hvor størrelsen  $U$  inn til blokk  $A$  skal være  $u(k)$ , der  $u$  er et digitalt signal og  $k$  er tids-indeks.



**Figur 4.23** Ut-kanal med overganger fra intern representasjon.

En likning som uttrykker skalering kan nå utvikles på samme måte som for innlesning. Utgangspunktet er:

$$W = K(U - U_0) + W_0 \quad (59)$$

Vanligvis skal vi ha

$$W = U, \quad (60)$$

som gir

$$\begin{aligned} K &= 1 \\ U_0 &= W_0 \\ U_1 &= W_1 \end{aligned} \quad (61)$$

der indeks 0 og 1 betegner henholdsvis nedre og øvre grenseverdi for det analoge pådragssignalets variasjonsområde.

Blokkene  $B$  og  $C$  beskrives med likninger tilsvarende innleseprogrammet, og dette fører til skaleringslikningen

$$X = k_3(U - U_0) + X_0 = k_3U + B \quad (62)$$

med parametre

$$\begin{aligned} k_3 &= \frac{K}{k_1 k_2} \\ B &= X_0 - k_3 U_0 \end{aligned} \quad (63)$$

hvor  $k_1$  og  $k_2$  er “forsterkningsfaktor” for henholdsvis D/A-omsetter og pådragsorgan, definert tilsvarende som i avsnitt 4.7.1. Hvis (60) gjelder, kan skrives noe forenklet:

$$X = \frac{1}{k_1 k_2}(U - W_0) + X_0 = k_3 U + B. \quad (64)$$



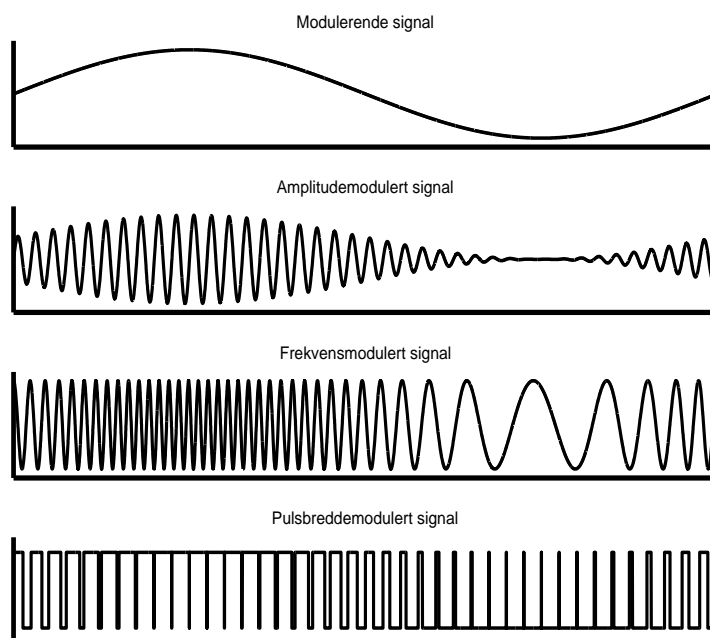
---

## KAPITTEL 5

# Modulasjon

### 5.1 Innledning og definisjoner

Som vi har vært inne på tidligere er signaler informasjonsbærerne i styrings- og reguleringsystemer, idet signalets spenningsverdi, evt. strømstyrke, for eksempel kan representere en måle- eller pådragsverdi. På grunn av signalmediets egenskaper og omkringsliggende støykilder kan det være vanskelige å overføre slike analoge signaler med tilstrekkelig nøyaktighet. Det kan da være nyttig å la informasjonen i det opprinnelige signalet bli representert av et *modulert* signal. Figur 5.1 viser eksempler på de tre prinsipielt ulike modulasjonsteknikkene vi skal se nærmere på i dette kapitlet.



**Figur 5.1** Eksempler på amplitude-, frekvens- og pulsbreddemodulasjon av et sinusformet nyttesignal.

Innen elektronikk og telekommunikasjon, som de fleste kybernetiske systemer bygger på, kan vi bruke følgende definisjoner:

**Modulerende signal** – et tidsvarierende signal som inneholder informasjon som skal overføres. Dette er vårt analoge nyttesignal.

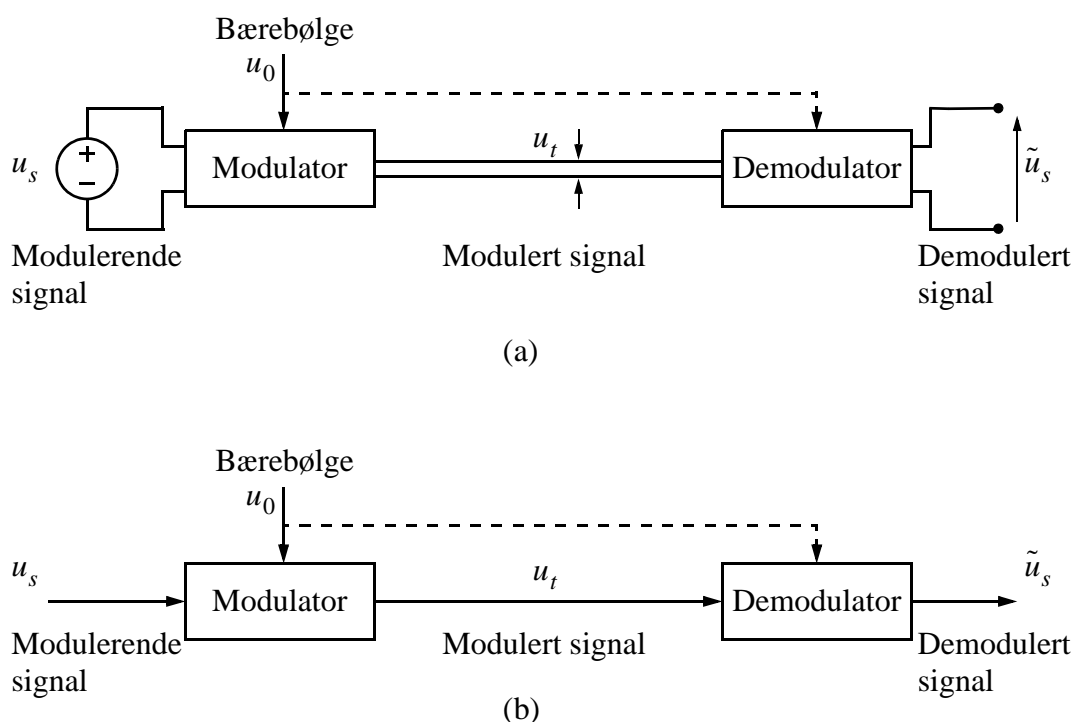
**Bærebølge** – et hurtigvarierende<sup>1</sup> periodisk signal som egner seg bedre til den aktuelle overføringen enn det modulerende signalet.

**Bærefrekvens** – bærebølgens frekvens.

**Modulasjon** – en prosess som består i å variere ett eller flere egenskaper hos bærebølgen som funksjon(er) av det modulerende signalet.

**Demodulasjon** – motsatt av modulasjon: en prosess som består i å gjenvinne informasjonen det opprinnelige (modulerende) signalet fra det modulerte signalet.

Informasjonen i et modulert signal er altså ikke representert som en momentan spennings- eller strømverdi, men som en annen karakteristisk og tidsvarierende størrelse i det modulerte signalet. Ved demodulasjon kan man siden komme tilbake til det opprinnelige signalet med den opprinnelige informasjonen i behold. Prinsippet er illustrert i figur 5.2. I noen tilfeller trengs tilgang på den umodulerte bærebølgen, eller iallfall noen av dens egenskaper (for eksempel bærebølgens frekvens eller fase), for å kunne demodulere signalet. Dette er symbolisert med en stiplet pil i figuren.



**Figur 5.2** Modulasjon og demodulasjon – prinsippskisse. (a) Eksemplet viser en signalkilde med utgangsspenning  $u_s$  som modulerer en bærebølge  $u_0$  og overføres som et modulert signal  $u_t$ . På mottakersiden demoduleres signalet til et nytt analogt signal  $\tilde{u}_s$ . Målet med prosessen er å gjenskape det opprinnelige signalet slik at  $\tilde{u}_s = u_s$ . (b) Forenklet prinsippskisse som fokuserer på informasjonsflyt og ikke signalenes elektriske karakteristikker.

1. Bærebølgens frekvens er vanligvis mye høyere enn båndbredden til det modulerende signalet.

Informasjonen bæres ved modulasjon av et signal med høyere frekvens enn det modulerende. Siden det er vanskelig å gjenvinne informasjonen hyppigere enn én gang for hver halvperiode av bærebølgen, får vi ved modulasjon en tasteffekt. Det er derfor viktig å velge bærefrekvensen en god del høyere enn nyttesignalets høyeste frekvens.

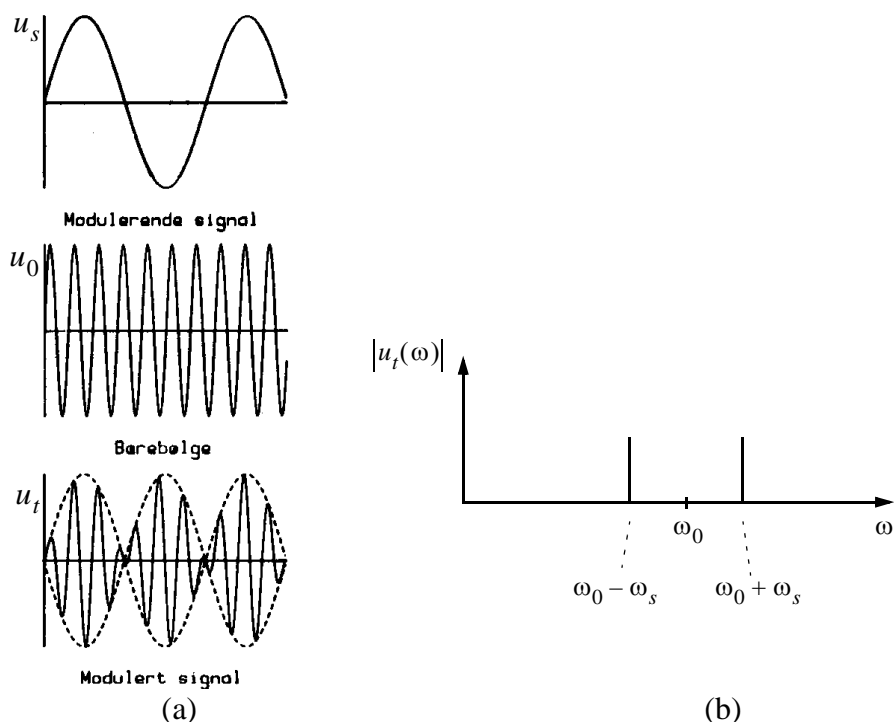
Det finnes en del målelementer som automatisk gir ut målesignal i form av et modulert signal, som så må demoduleres for å få tak i selve måleverdien. Det finnes et utall forskjellige typer og varianter av modulasjonsprinsipper. Vi skal i det følgende bare se på de som er viktigst innen instrumenteringsteknikken, nemlig amplitude-, frekvens- og pulsbreddemodulasjon.

## 5.2 Amplitudemodulasjon med undertrykt bærebølge

### 5.2.1 Modulasjon

Ved amplitudemodulasjon (AM) er bærebølgen et sinusformet signal med frekvens mye høyere enn det modulerende signalets båndbredde. Amplitudemodulasjon foregår ved at bærebølgens amplitude varieres slik at den til enhver tid er et mål for det modulerende signalets verdi. I Figur 5.3a er dette illustrert. Øverst i figuren finner vi det modulerende signalet  $u_s$ , for enkelhets skyld valgt som en lavfrekvent sinusbølge:

$$u_s = U_s \sin \omega_s t, \quad (65)$$



**Figur 5.3** (a) Amplitudemodulasjon med undertrykt bærebølge. (b) Det modulerte signalets spektrum.

mens bærebølgen er gitt av:

$$u_0 = U_0 \sin \omega_0 t \quad (66)$$

Amplitudemodulasjonen kan nå beskrives som en multiplikasjon av de to signalene:

$$u_t = U_s U_0 \sin(\omega_s t) \sin(\omega_0 t). \quad (67)$$

Ved å benytte den kjente formelen

$$\sin u \sin v = \frac{1}{2} [\cos(u - v) - \cos(u + v)]$$

kan likning (67) gjøre om som følger:

$$u_t = \frac{1}{2} U_s U_0 [\cos((\omega_0 - \omega_s)t) - \cos((\omega_0 + \omega_s)t)] \quad (68)$$

Det modulerte signalets spektrum er vist i figur 5.3b. Her ser vi, slik formelen over viser, at det inneholder frekvensene  $\omega_0 - \omega_s$  og  $\omega_0 + \omega_s$ . Selve bærebølgen er derimot borte. Denne modulasjonsformen kalles derfor amplitudemodulasjon med *undertrykt bærebølge*.

I det analyserte tilfellet var det modulerende signalet en sinusbølge, mens det i det generelle tilfellet vil være et tilfeldig varierende signal. Imidlertid kan ethvert signal betraktes som en sum av sinussignaler med ulike frekvenser, amplituder og fasevinkler. Hver av disse frekvenskomponentene vil oppføre seg i henhold til analysen over, og det resulterende spektret vil derfor bli summen av spektralbidragene fra alle frekvenskomponentene.

I tidsplanet ser resultatet av modulasjonen ut som nederst i figur 5.3a. Vi ser at signalets omhylningskurve representerer tallverdien til det modulerende signalet, mens signalets fase skifter  $180^\circ$  når inngangssignalet skifter fortegn.

Denne modulasjonsformen er hyppig forekommende innen instrumenteringsteknikken, spesielt i forbindelse med målebroer, induktive posisjonsmålere og andre måleelementer som baserer seg på bruk av vekselspanning.

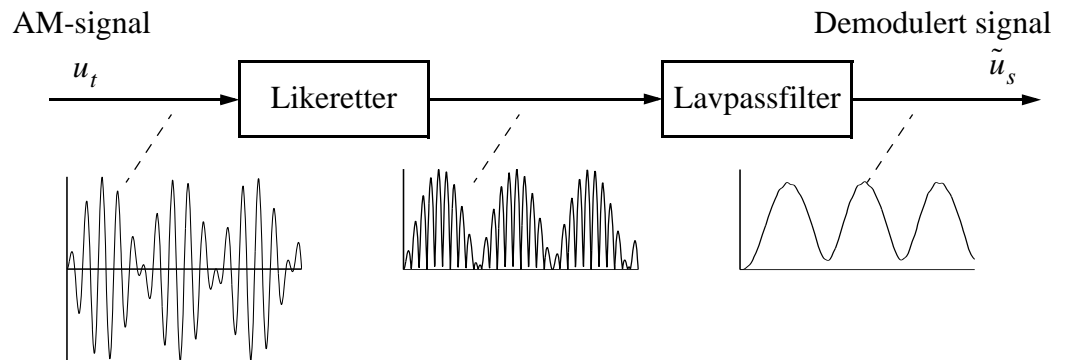
### 5.2.2 Demodulasjon

Vi skal nå se på noen ulike måter for å gjenvinne den opprinnelige signalinformasjonen fra et amplitudemodulert signal med undertrykt bærebølge.



### Gjenvinning av signalets tallverdi

Hvis vi bare er interessert i å gjenvinne det opprinnelige signalets tallverdi men ikke trenger dets fortegn, kan demodulasjonen skje enkelt ved å likerette og lavpassfiltrere, som vist i figur 5.4<sup>1</sup>. Slik demodulasjon kan enkelt gjøres med analoge kretser, og vi får da et analogt signal  $\tilde{u}_s$  som resultat. Alternativt kan det modulerte signalet testes, og demodulasjonen kan deretter foregå i programvare.



**Figur 5.4** Demodulasjon av AM-signal via likeretting og lavpassfiltrering.

### Synkron demodulasjon

Det er også mulig å gjenvinne det modulerende signalets fortegn fra et AM-signal med undertrykt bærebølge, men da må demodulasjonen være *synkron*. Dette gjøres ved igjen å multiplisere det modulerte signalet med et signal som er synkront med bærebølgen, dvs. har nøyaktig samme frekvens som denne. Vi benytter signalet  $u_d = U_d \sin(\omega_0 t)$ , og får:

$$\begin{aligned}\hat{u}_s &= u_t U_d \sin(\omega_0 t) \\ &= U_s U_o U_d \sin(\omega_s t) (\sin(\omega_0 t))^2.\end{aligned}\quad (69)$$

Vi benytter nå den kjente identiteten

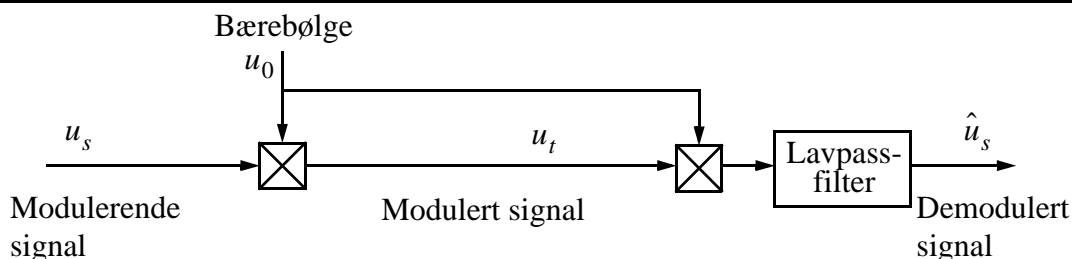
$$(\sin u)^2 = \frac{1 - \cos 2u}{2}$$

og får:

$$\hat{u}_s = \frac{1}{2} U_s U_o U_d (\sin(\omega_s t) - \sin(\omega_s t) \cos(2\omega_0 t)). \quad (70)$$

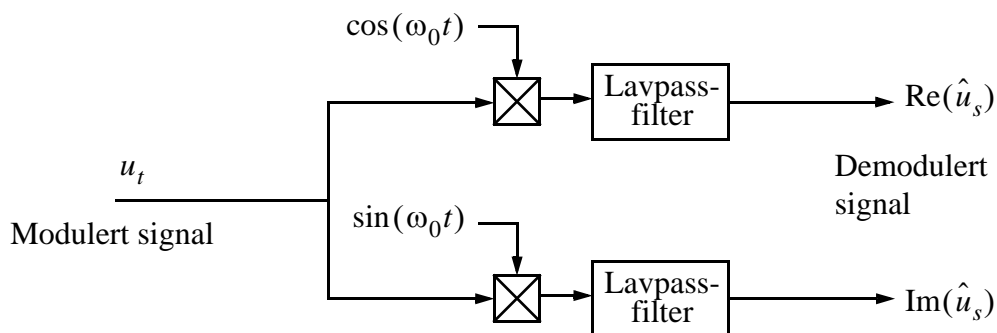
1. Det demodulerte signalet i figur 5.4 er ikke en nøyaktig gjengivelse av det modulerende signalets tallverdi. Dette kommer av at vi har valgt en relativt lav bærefrekvens for illustrasjonens skyld, og lavpassfilteret vil derfor også påvirke nyttesignalet. I praktiske anvendelser har vi  $\omega_0 \gg \omega_s$ , og kan da få en nærmest perfekt demodulasjon.

Det siste leddet består av et spekter omkring den doble bærebølgefrequensen ( $2\omega_0$ ), og kan fjernes med et lavpassfilter. Vi står dermed igjen med det opprinnelige signalet multiplisert med en konstant. Modulasjon og synkron demodulasjon etter dette prinsippet er vist blokkskjematisk i figur 5.5.



**Figur 5.5** Synkron modulasjon og demodulasjon. De kvadratiske blokkene representerer multiplikasjon.

For at synkron demodulasjon skal virke skikkelig, må mottageren ha tilgang på bærebølgen fra modulatoren, med riktig fase. Dersom mottageren har tilgjengelig informasjon om bærebølgens frekvens men ikke om dens fase, kan signalet demoduleres i en real- og imaginærkomponent, kalt *kvadraturkomponenter*. Blokkskjemaet for en kvadraturdemodulator er vist i figur 5.6. Inngangssignalet betraktes som et kom-



**Figur 5.6** Kvadraturdemodulator.

plekst signal med kjent frekvens men med en ukjent fase. Signalet sendes inn i to synkrone demodulatorer som er drevet av bærebølgefrequensens to kvadraturkomponenter, sinus og cosinus. Den ene demodulatoren produserer nå realdelen til signalet, mens imaginærdelen kommer fra den andre.

Matematisk sett vises dette ved regning med trigonometriske formler som før. De to komponentene kan i etterfølgende utstyr benyttes til å beregne inngangssignalets tallverdi og fase i forhold til den lokale bærebølgefase.

AM-modulasjon kalles også dobbelt sidebåndmodulasjon (DSB), med eller uten bærebølge. Det er mulig å filtrere bort det ene sidebåndet, og modulasjonen kalles da AM med enkelt sidebånd (SSB, *Single sideband*). Signalspekteret blir da bare halvparten så bredt.

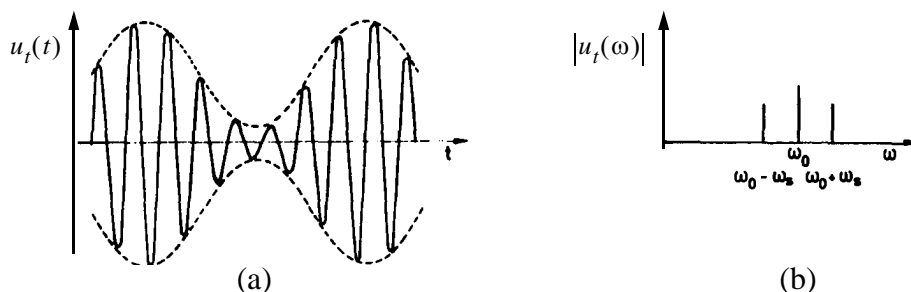
## 5.3 Amplitudemodulasjon med bærebølge

### 5.3.1 Modulasjon

Dersom bærebølgen adderes til det modulerte signalet, fås “vanlig” amplitudemodulasjon. Matematisk sett kan dette uttrykkes som følger:

$$u_t = U_s[\sin(\omega_s t) + A]U_0\sin(\omega_0 t), \quad (71)$$

der  $A \geq 1$  er en konstant. Vi ser at dette innebærer en nullpunktsforskyvning (offset) av det modulerende signalet slik at det justerte signalet aldri skifter fortegn. I tidsplanet kan resultatet bli som i Figur 5.7.



**Figur 5.7** (a) Amplitudemodulert signal med bærebølge. (b) Resulterende signalspektrum.

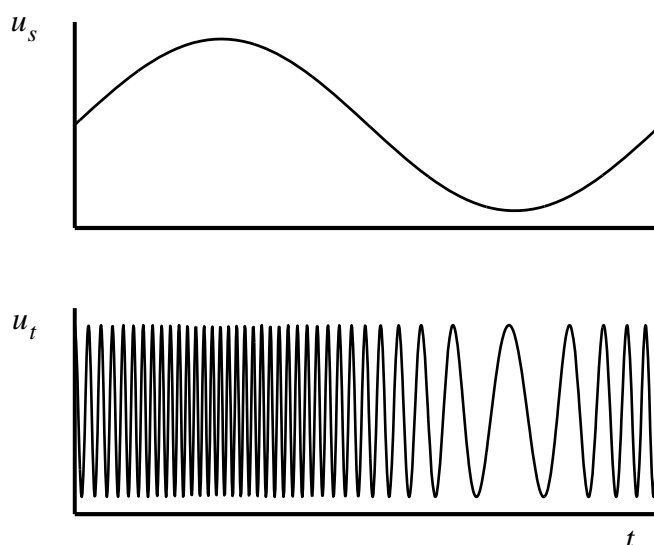
### 5.3.2 Demodulasjon

Det opprinnelige modulerende signalet kan nå gjenvinnes entydig fra det modulerte signalets omhylningskurve på liknende vis som i figur 5.4 ved likeretting og lavpass-filtrering, men vi må i tillegg foreta en nullpunktsforskyvning for å kompensere for den vi gjorde i (71). Demodulatoren er altså i dette tilfellet ikke avhengig av å kjenne bærebølgens frekvens og fase, og demodulatoren blir derfor mye enklere.

AM-modulasjon med bærebølge har vært mye nyttet for kringkastingsformål.

## 5.4 Frekvensmodulasjon

Amplitudemodulasjonens nøyaktighet er begrenset av drift i forsterkning og nullpunkt i de enkelte ledd av systemet. Dersom for eksempel dempingen i en overføringskabel øker, vil det modulerte signalet få lavere amplitude, og dette tolkes feilaktig som om det opprinnelige (modulerende) signalet har en lavere verdi. En teknikk som ikke har denne ulempen, er frekvensmodulasjon (FM). Prinsippet er illustrert i figur 5.8. Her



**Figur 5.8** Frekvensmodulasjon. Momentanverdien til nyttesignalet  $u_s$  bestemmer den momentane frekvensen til det modulerte signalet  $u_t$ .

formidles informasjonen som en funksjon av det modulerte signalets frekvens. Hvis bæreølgs nominelle frekvens er  $\omega_0$  og nyttesignalets momentanverdi er  $u_s(t)$ , kan det modulerte signalets momentane frekvens  $\omega_m$  uttrykkes som

$$\omega_m(t) = \omega_0 + A \cdot u_s(t) \quad (72)$$

der  $A$  er en konstant.

Det modulerte signalets frekvens er langt på vei uavhengig av unøyaktigheter i systemet, derfor begrenses nøyaktigheten i FM-overføringer kun av modulatorens og demodulatorens konformitet.

Siden all informasjonen ligger i et modulerte signalets nullgjennomganger, er signalets amplitude uvesentlig. Demodulatoren inneholder derfor et element som gjør signalet om til firkantsignal. Dermed dempes støy som ligger overlagret på signalet. Den støy-messige effekten av det kan godt merkes ved å sammenligne en radiostasjon i FM-båndet med en kanal på mellombølge, hvor amplitudemodulasjon benyttes.

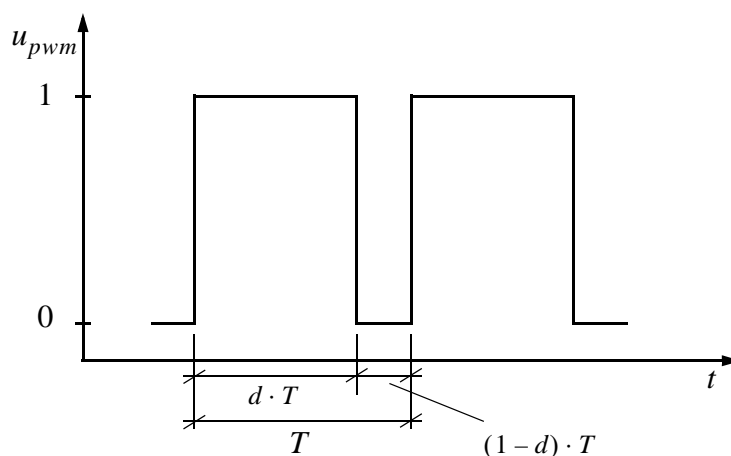
I instrumenteringssammenheng gjøres frekvensmodulasjonen ved hjelp av en spenningsstyrt oscillator, også kalt spenning-til-frekvensomsetter eller U/F (*eng.*: Voltage Controlled Oscillator, VCO). Tilsvarende skjer demodulasjonen i en frekvens-til-spenningsomsetter, F/U. Overføringsmediet mellom dem kan være ulineært, for eksempel optiske koblere og ulineære transformatorer, uten at dette i prinsippet forringer signalkvaliteten.

F/U og U/F-omsettere har en øvre og nedre frekvensgrense, et vanlig område er 10 til 100 kHz. Som regel er karakteristikken lineær, og nøyaktigheten kan være bedre enn 0,05%.

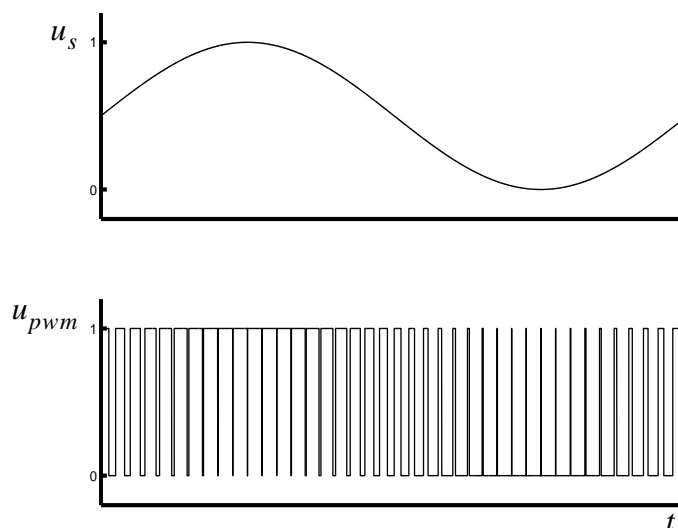
## 5.5 Pulsbreddemodulasjon

Den siste modulasjonsformen vi skal se på, er pulsbreddemodulasjon. Den engelske betegnelsen er Pulse Width Modulation, og forkortelsen PWM brukes ofte også på norsk.

I motsetning til AM og FM, som er basert på sinusformede bærebølger, er et pulsbred-demodulert signal et rent binært signal  $u_{pwm}$  med et fast pulsintervall  $T$  kalt *svitsjeperioden*, tilsvarende en fast svitsjefrekvens (“bærefrekvens”) på  $f = 1/T$ . Den informasjonsbærende parameteren er signalets såkalte *duty cycle*,  $d$ . Duty cycle er et tall i intervallet  $0 \leq d \leq 1$  som angir hvor stor andel av svitsjeperioden signalet er høyt (logisk “1”), dvs. den relative bredden til den “aktive” pulsen; derav navnet pulsbred-demodulasjon. Dette er illustrert i figur 5.9. Figur 5.10 viser et sinusformet nyttesignal og dets pulsbred-demodulerte ekvivalent.

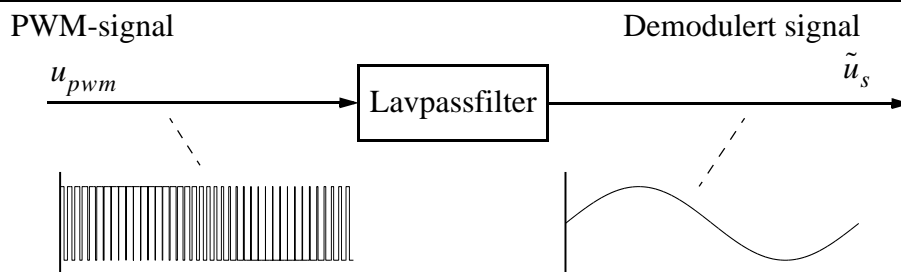


**Figur 5.9** PWM-signalets svitsjeperiode  $T$  og duty cycle  $d$ .



**Figur 5.10** Eksempel på pulsbred-demodulasjon.

Demodulasjon av PWM-signaler skjer ved å finne bredden på hver puls, som avhenger lineært av signalets middelvei i den aktuelle perioden. En tilnærming til middelveien finnes ved lavpassfiltrering. Filteret må slippe gjennom alle frekvenser i nyttesignalet, men effektivt stoppe det modulerte signalets svitsjefrekvens. Det er derfor viktig å velge en svitsjefrekvens som ligger langt over nyttesignalets båndbredde.



**Figur 5.11** Eksempel på pulsbredde-demodulasjon.

Pulsbreddemodulasjon er mye benyttet for pådragssignaler, blant annet fordi det er mye lettere å styre et pådrag av og på enn å styre det lineært. Et typisk eksempel er temperaturstyring: en termostat for av/på styring av varmeelementet er svært enkel og består bare av et bryterelement, mens en kontinuerlig, analog styring av varmeeffekten ville være både komplisert og lite energieffektiv. Temperaturstyring er et eksempel på en prosess som er treg (dvs. har lange tidskonstanter) i forhold til PWM-signalets modulasjonsfrekvens, og den demodulerende lavpassfiltreringen skjer derfor direkte i prosessen uten noe eksplisitt filterelement.

Ved pulsbreddemodulasjon av pådrag til mekaniske aktuatorer (motorer og liknende) velges gjerne svitsjefrekvenser på 20 kHz eller høyere for å unngå at det tidsvarierende spenningssignalet skaper hørbare vibrasjoner. Enkelte systemer som vi omgir oss med i dagliglivet "synder" mot denne regelen og skaper karakteristiske "summelyder". Dette gjelder for eksempel moderne togsett. Summingen er lettest hørbar fra perrongen ved oppbremsing og akselerasjon, men den kan også høres fra innsiden av togsettet. Et annet eksempel er billige batteridrevne bormaskiner og skrutrekkere.

## KAPITTEL 6

# Strømsløyfe

Blant de standardiserte signalformatene som ble nevnt i kapittel 4, er tradisjonelt 4-20 mA<sup>1</sup> blant de mest utbredte. I dette kapitlet skal vi se på fordelene som er forbundet med strøm som signaleringsstørrelse. Vi skal også se detaljert på hvordan en strømsløyfebasert sensor fungerer og hvordan en strømsløyfe dimensjoneres.

### 6.1 Hvorfor konverterer vi signaler til 4-20 mA?

Fordelen med et spenningssignal er enkelhet. Nesten alle forstår konseptet: signalet finnes som spenningsforskjellen mellom to ledninger, én positiv og én negativ. Elektronikken som driver spenningssignalet må ha energitilførsel, noe som kan skje via to ekstra ledere. Alternativt kan energitilførsel skje med én ekstra leder dersom for eksempel den negative signallederen brukes som “fellesjord” (dvs. som både signal- og kraftjord).

Ulempen med spenningssignaler er først og fremst presisjonstap pga. elektrisk støy fra omkringliggende utstyr og endelig inngangsimpedans i måleutstyret som skal ta imot signalet. Vi skal ta for oss disse faktorene i tur og orden.

#### 6.1.1 Robusthet mot støy

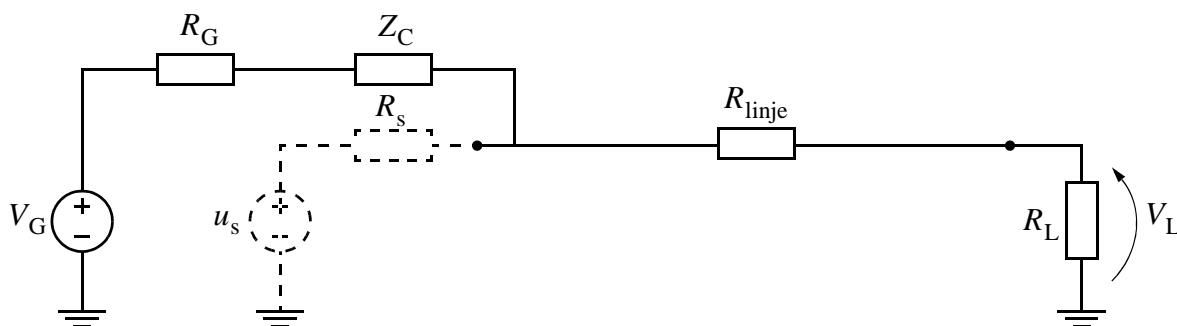
##### *Spenningssignalering krever kompromisser*

Figur 6.1 viser en enpolet signalkilde  $u_s$  med kildeimpedans  $R_s$ , koplet (med stiplet linje) til en signalleder. Signallederens totale serieresistans kaller vi  $R_{linje}$ . På lastsiden er linja koplet til en mottakerkrets med ekvivalent inngangsimpedans  $R_L$ , og det er spenningen  $V_L$  over denne impedansen som utgjør det mottatte signalet. I figuren er det også tegnet inn en støykilde  $V_G$  med indre motstand  $R_G$ . Denne støykilden, som f.eks. kan representere lysnettet (dvs. 220 V/50 Hz) eller en hvilken som helst annen

---

1. Ifølge standard notasjon skal et strømintervall som dette skrives “4 mA til 20 mA” eller “(4 til 20) mA” for å unngå tvetydigheter. Skrivemåten “4-20 mA” er derimot så godt innarbeidet at vi vil holde oss til denne.

kilde som påvirker spenningen på signallinja, har en koplingsvei mot mottakeren  $R_L$ . Kopplingsveien kan være av induktiv eller kapasitiv art (dvs. uavhengig av fysisk berøring) og er representert med koplingsimpedansen  $Z_C$ .



**Figur 6.1** Støykilde med koplingsimpedans  $Z_C$  mot en signallinje.

Vi bruker superposisjonsprinsippet for å studere støykildens betydning for spenningen på linja, og setter derfor  $u_s = 0$ . Vi skal gjøre en rent kvalitativ analyse, og tillater oss derfor også å sette  $R_s \rightarrow \infty$ . Dermed kan vi se helt bort fra de stiplede delen av figuren.

Ønsket vårt er at støykilden skal påvirke nyttesignalet vårt minst mulig, det vil si at støyspenningen  $V_L$  i figur 6.1 blir så liten som mulig. Vi kan sette opp følgende likning for kretsen i figuren:

$$V_L = V_G \cdot \frac{R_L}{R_G + Z_C + R_{linje} + R_L}. \quad (73)$$

Vi antar at  $R_G$  og  $R_{linje}$  er neglisjerbare i forhold til de øvrige impedansene i (73), og kan da gjøre følgende analyse:

Når  $Z_C \gg R_L$  får vi  $V_L \approx V_G \cdot \frac{R_L}{Z_C}$ , dvs.  $V_L$  er liten.

Når  $Z_C \ll R_L$  får vi  $V_L \approx V_G$ , dvs.  $V_L$  er stor.

Herav framgår at  $R_L$  må gjøres så liten som mulig og  $Z_C$  så stor som mulig.

Stor  $Z_C$  får vi ved å fjerne støykilden så langt fra mottakeren som mulig (dette reduserer den kapasitive og induktive koplingen). Dette kan imidlertid være svært vanskelig eller nærmest umulig. Derfor må vi etablere et system der signalmottakeren har *liten inngangsimpedans*  $R_L$ .

I systemer hvor man signalerer med spenning ønsker vi at det skal gå minst mulig strøm i linja, fordi linjas seriemotstand  $R_{linje}$  skaper et spenningsfall (en feilspenning) proporsjonalt med linjestrømmen. Det vil derfor være nødvendig at  $R_L$  har *meget høy verdi*. Ved spenningssignalsering står vi med andre ord foran to motstridende hensyn, og et kompromiss er nødvendig.



### ***Strømsignaler er bedre***

For å overvinne disse problemene er et strømsignal mer effektivt, og dette skal vi illustrere med et eksempel. Anta en situasjon der 0-20 mA brukes til å representere et målesignal for temperatur i området (0 til 100)°C. I vårt måleinstrument har vi en motstand med resistans  $R_L = 250\Omega$  som strømmen passerer gjennom, og vi måler spenningsfallet  $V_L$  over denne motstanden.

Ved 100°C passerer en strøm på 20 mA gjennom motstanden og gir oss en spenning på 5 V. Dette 5 V-signalet er uavhengig av motstanden i ledningene som bærer den (med spenningssignalerin ville en endring i ledningsmotstand føre til en endring i signalet ved måleinstrumentet og skape en målefeil). Dette betyr at *et strømsignal kan overføres lange avstander gjennom ledere av varierende og ukjent motstand uten tap av nøyaktighet*.

Den andre fordelen med strømsignaler er bedre støyimmunitet. Støysignaler skapes blant annet av tidsvarierende magnetfelter (ofte generert av nettkabler med vekselstrøm) som induserer strømmer i signalledningene på samme måte som i en generator. Hvis våre ledere er små og tett sammen (teoretisk sett på samme sted) genererer disse feltene like og motsatt rettede (med hensyn til sløyfen) strømmer i de to lederne, som kansellerer hverandre. I praksis vil et tvunnet trådpar av telefonkabelkvalitet fungere godt.

#### **6.1.2 Signal og kraftforsyning på samme linje**

Det som er sagt i kapittel 4 om *elevated zero* gjelder også for strømsignaler, men her gir det oss enda en gevinst.

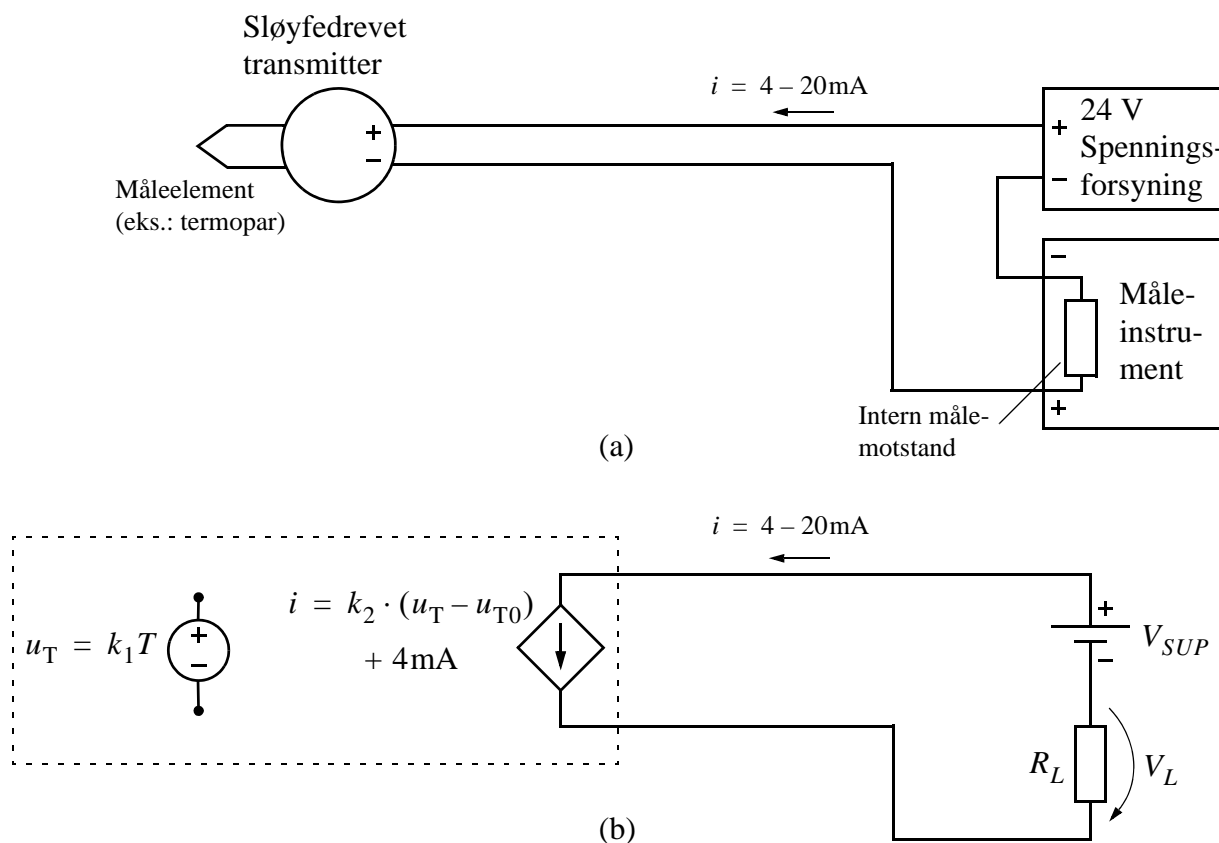
Hvis vi bruker et 0-20 mA signal, må vi forsyne elektronikken med energi på samme måten som når vi forsyner et spenningssignal, ved å bruke en eller to ekstra forsyningsledninger. Med et 4-20 mA signal flyter det imidlertid alltid minst 4 mA i kretsen. Det betyr at hvis vi klarer å holde strømforbruket til elektronikken vår under 4 mA, kan vi ha *strømforsyningen og signalet på samme trådpar*. Denne teknikken forenkler installasjonen, spesielt i store anlegg, fordi kun ett tvunnet trådpar trengs for å knytte en sensorenhet til et måleinstrument eller reguleringsmodul.

Et instrument som sender sitt målesignal som et 4-20 mA signal refereres ofte til som en *transmitter*, og hvis det henter sin energi fra selve strømsløyfen kalles det en *sløyferdrevet transmitter* (eng.: *loop powered transmitter*).

## **6.2 Strømsløyfens oppbygning og virkemåte**

### **6.2.1 Strømforsyningen**

Transmitteren er en *strømtrekkende* krets, noe som betyr at den vil forsøke å trekke en strøm fra en ekstern strømforsyning. Dette er vanligvis en 24 V likespenningsskilde som gjerne er integrert i måleinstrumentet som transmitteren er koblet til. Mens spenningstransducere koples i parallell med måleinstrumentet, koples strømtransmitteren i serie. Figur 6.2 viser en typisk strømsløyfekrets.



**Figur 6.2** Typisk 4-20 mA strømsløyfe med temeperaturtransmitter basert på et termopar. (a) Strukturen i sløyfa. (b) Kretsekvivalent.

Strømmen flyter pr. def. *fra positiv terminal* av strømforsyningen, gjennom transmitteren, gjennom den interne lastmotstanden i måleinstrumentet og tilbake til den negative terminalen på strømforsyningen. Spenningsfallet over den interne lastmotstanden er signalet som måleinstrumentet “tar imot”.

Sløyfens strømforsyning gir vanligvis all driftsstrøm til transmitter, mottaker og alle andre komponenter i sløyfen som krever en godt regulert likespenning. En spenning på 24 V er fremdeles mest populær i 4-20 mA prosessovervåkningssystemer, fordi 24 V også brukes i mange andre instrumenter og elektromekaniske komponenter som ofte finnes i industrielle miljøer. Lavere forsyningsspenninger, for eksempel 12 V, er også populære fordi de brukes i mange datamaskinsystemer.

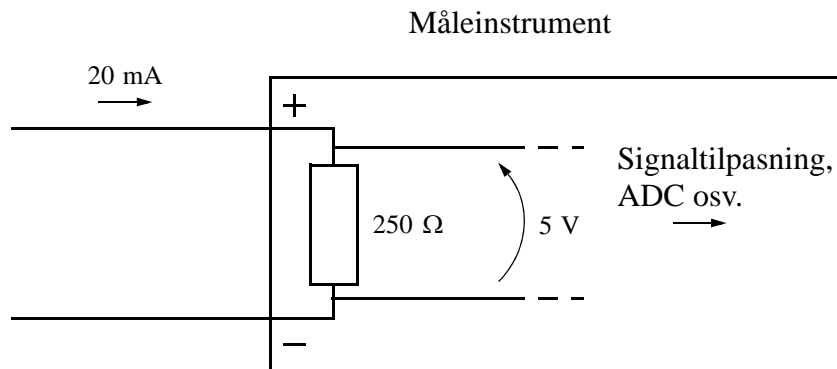
### 6.2.2 Sløyfens spenningsfall

En av de viktigste parametre for en prosessovervåkningsenhet (for eksempel måleinstrumentet i figur 6.3) – sløyfedrevet eller med separat strømforsyning – er den totale resistans (eller “belastningen”) den representerer i strømsløyfen. De fleste transmitters datablad spesifiserer den maksimale sløyferesistansen som transmitteren kan drive og samtidig gi et fullt 20 mA utgangssignal.

Ifølge Kirchhoffs spenningslov må summen av spenningsfallene rundt en sløyfe være lik forsyningsspenningen. Hvis et sløyfedrevet system har en 24 V spenningsforsyning, må derfor summen av spenningsfallene rundt sløyfen også utgjøre 24 V. I en 4-

20 mA strømsløyfe vil det maksimale spenningsfallet over hver komponent i sløyfen være lik komponentens resistans multiplisert med den maksimale strømmen, dvs. 20 mA. Eksempelvis vil 250  $\Omega$  -motstanden i figur 6.3 gi et maksimalt spenningsfall på

$$250 \Omega \cdot 20 \text{ mA} = 5,0 \text{ V}.$$



**Figur 6.3** Måleinstrument med målemotstand på 250  $\Omega$ . Dette gir 5 V målespenning ved maksimal sløyfestrøm.

### 6.2.3 Transmitterens ytelse

Med teorien ovenfor friskt i minne, betrakter vi følgende spørsmål: Hvis vi har et 24 V sløyfedrevet system der transmitterens minimum driftsspenning (dvs. det minste spenningsfallet over transmitteren som er tilstrekkelig til at den i henhold til databladet) er f.eks. 8 V og måleinstrumentets spenningsfall er maksimalt 5 V, hvor blir det av de “ekstra” 11 V?

Svaret er at *de ekstra 11 V i sin helhet må ligge over transmitteren*. Transmitteren må med andre ord kunne variere sin totale indre resistans, og bruke dette som et pådrag for å regulere strømmen i sløyfen slik at strømmen står i et korrekt forhold til den fysiske målevariabelen som skal formidles.

Vanligvis er både minimum og maksimum driftsspenning angitt i transmitterens datablad. Minimumsspenningen er den som kreves for å sikre at transmitterens elektronikk fungerer som den skal, mens maksimalspenningen begrenses av transmitterens maksimale effektutvikling (i praksis: hvor mye effekt transmitteren kan disipere før den blir for varm) og halvlederkomponentenes sammenbruddsspenning (dvs hvor mye spenning transmitteren “tåler” over strømsløyfeterminalene).

I transmitteren utvikles en varmeeffekt gitt av strømmen gjennom transmitteren multiplisert med spenningen over den. Den høyeste forventede strømmen er oftest, men ikke alltid, 20mA. Hvis vi for eksempel har et spenningsfall over transmitteren på 30 V ved en utgangsnivå på 30 mA, vil transmitterens effektutvikling være:

$$30 \text{ V} \cdot 30 \text{ mA} = 0,9 \text{ W}.$$

### 6.2.4 Ledningsmotstand

Fordi kobberledninger har en resistans som er direkte proporsjonal med lengden og også varierer med ledningsdiameteren, vil ikke denne diskusjonen være komplett uten å nevne ledningsmotstandens innvirkning på systemet.

I anvendelser der to eller flere sløyfemonitoreringsenheter (måleenheter) er forbundet over lange avstander (typisk 300 m til 600 m), brukes typisk 24 V spenningsforsyning. Mange transmittre trenger minimum 8 V driftsspenning, og i tillegg har en ofte 3 V til 4 V spenningsfall over hver måleenhet og 2 V til 4 V spenningsfall over ledninger og koplinger. Minimum forsyningsspenning kan da lett overskride 16 V. En spenning på 24 V gir dermed nødvendig margin.

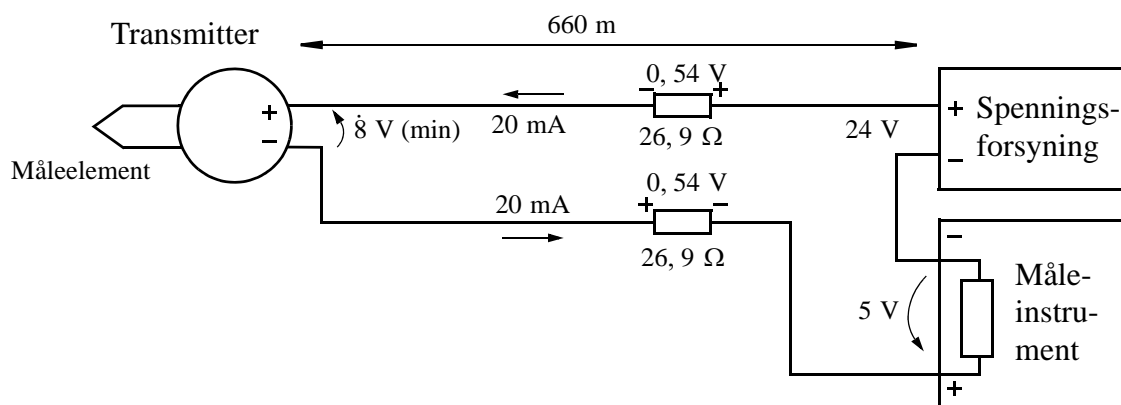
## 6.3 Regneeksempel

Spenningsfallet over en gitt ledningslengde finnes ved å multiplisere ledningens totale motstand med strømmen som passerer gjennom den. Ledningens totale motstand finnes ved å slå opp dens spesifikke motstand (vanligvis uttrykt i  $\Omega/1000$  m) i ledningens datablad. Figur 6.4 viser en transmitter tilkopleet et måleinstrument via en ledning på 660 m med en spesifikk motstand på  $40,8 \Omega/1000$  m. Ledningsresistansen én vei er da:

$$R_{660 \text{ m}} = 660 \text{ m} (40,8 \Omega/1000 \text{ m}) = 26,9 \Omega$$

Spenningsfallet én vei over ledningen når sløyfestrømmen er 20 mA, er da lik:

$$E = 20 \text{ mA} \cdot 26,9 \Omega = 0,54 \text{ V}$$



**Figur 6.4** Virkningen av ledningsmotstand.

Imidlertid må strømmen gå 660 m fra transmitteren til måleinstrumentet og deretter 660 m tilbake til transmitterens positive terminal, til sammen 1320 m. Total ledningsresistans er derfor

$$R_{1320 \text{ m}} = 26,9 \Omega \cdot 2 = 53,9 \Omega$$

og spenningsfallet i ledningen blir

$$E = 0,020 \text{ A} \cdot 53,9 \text{ } \Omega = 1,08 \text{ V}.$$

Hvis vi ser nedover sløyfen mot måleinstrumentet, ser transmitteren summen av spenningsfallet i ledningen på 1,08 V og spenningsfallet over måleinstrumentet på 5 V, som til sammen blir 6,08 V. Hvis transmitteren selv krever minst 8 V (som også utgjør et spenningsfall), trenger systemet i Figur 6.4 en spenningsforsyning på minst 14,08 V.

