

Innføring og støttelitteratur for IN1020 - Maskinvaredelen

Omid Mirmotahari

2018

Innhold

1	Introduksjon	5
1.1	Kompendiets oppbygning	5
1.2	Blooms taksonomi	6
1.3	Abstraksjonsnivå	7
1.4	Kompendium under arbeid	8
1.4.1	Bidragstere	8
2	Boolsk Algebra	9
2.1	Introduksjon	9
2.1.1	Mål: En 7-segment skjerm	10
2.1.2	Læringsmål	11
2.1.3	Nøkkeltermer	12
2.2	Sannhetsverditabell	12
2.2.1	Sannhetsverditabell med mange innganger og utganger	14
2.2.2	Sannhetsverditabellen for en 7-segment skjerm	15
2.3	Funksjonsuttrykk	15
2.3.1	Basis funksjoner	16
2.3.2	NAND og NOR funksjoner	18
2.3.3	Odde funksjoner: XOR og XNOR	19
2.3.4	Andre sammensatte funksjoner	19
2.3.5	Funksjonsuttrykkene for en 7-segment skjerm	25
2.4	Forenkling av funksjonsuttrykk med algebra	26
2.4.1	Regneregler – forenklet oversikt	27
2.4.2	DeMorgans Teorem	28
2.4.3	Rekkefølge og når er nok nok?	29
2.4.4	Forenkling av funksjonsuttrykk for en 7-segments skjerm	30
2.5	Karnaugh diagram	31
2.5.1	2-variabels Karnaugh diagram	32
2.5.2	3-variabels Karnaugh diagram	34
2.5.3	4-variabels Karnaugh diagram	38
2.5.4	XOR og XNOR i Karnaugh diagram	41
2.5.5	Forenkling av funksjonsuttrykkene for en 7-segment skjerm med Karnaugh Diagram	42
2.6	Don't Care - verdier	46

2.7	Videre henvisning	49
2.8	Oppgaver	50
2.8.1	Sannhetstabell	50
2.8.2	Funksjonsuttrykk	50
2.8.3	Forenkling av uttrykk med regneregler	50
2.8.4	Forenkling av uttrykk med Karnaugh Diagram	52
2.8.5	Porter og kretsdesign	52
2.8.6	NØTTER godt og blandet	52
3	Fra Logikk til Kretsimplementasjon	53
3.1	Introduksjon	53
3.1.1	Forutsetninger	53
3.1.2	Læringsmål	53
3.1.3	Nøkkeltermer	54
3.2	Basis kretselementer og porter	54
3.2.1	Sammensatte porter	56
3.2.2	Porter med flere innganger	57
3.3	Kretser med flere innganger og utganger	57
3.4	Kretsimplementasjon av 7-segment skjerm	58
3.5	Kretsanalyse	60
3.6	Bubble-push	60
3.7	Videre henvisning	60
3.8	Oppgaver	61
3.8.1	Porter	61
3.8.2	Kretimplementasjon	61
3.8.3	Kretsanalyse	61
3.8.4	Bubble-push	61
3.8.5	NØTTER	61
4	Kombinatorisk logikk	62
4.1	Introduksjon	62
4.1.1	Mål: En kombinert addisjon-subtraksjon krets	62
4.1.2	Forutsetninger	62
4.1.3	Læringsmål	62
4.1.4	Nøkkeltermer	62
4.2	Addisjonskrets - ADDER	63
4.2.1	Halvadder	63
4.2.2	Fulladder	63
4.2.3	Mentepropagering	63
4.3	Komparator	63
4.4	Dekoder og Enkoder	63
4.5	Multiplekser og de-Multiplekser	63
4.6	Aritmetisk Logisk Enhet - ALU	63
4.7	Videre henvisning	63
4.8	Oppgaver	63
4.8.1	Addisjon	63

4.8.2	Komparator	63
4.8.3	Dekoder og Enkoder	63
4.8.4	Multiplekser og de-Multiplekser	63
4.8.5	ALU	63
4.8.6	NØTTER	63
5	Sekvensiell logikk	64
5.1	Introduksjon	64
5.1.1	Mål: Lage en sekvensteller med en tilstandsmaskin	64
5.1.2	Forutsetninger	64
5.1.3	Læringsmål	64
5.1.4	Nøkkeltermer	64
5.2	Klokkesignal, synkrone og asynkrone kretser	65
5.3	Låsekrets	65
5.3.1	Dobbel-inverter	65
5.3.2	SR-Latch	65
5.3.3	D-Latch	65
5.4	Master-slave låsekretser - Flip Flop	65
5.4.1	D-Flip Flop	65
5.4.2	JK-Flip Flop	65
5.4.3	T-Flip Flop	65
5.5	Tilstandsmaskin	65
5.5.1	Tilstandsdiagram	65
5.5.2	Tilstandstabell	65
5.5.3	Redusering av tilstander	65
5.5.4	Kretsimplementasjon av tilstandsmaskiner	65
5.5.5	Ubrukte tilstander	65
5.6	Videre henvisning	65
5.7	Oppgaver	66
5.7.1	Klokkesignaler og synkronisering	66
5.7.2	Låsekretser	66
5.7.3	Tilstandsmaskin	66
5.7.4	NØTTER	66
6	Datamaskinarkitektur	67
6.1	Introduksjon	67
6.1.1	Mål 1: Hvordan forstår datamaskinen maskinkode?	67
6.1.2	Mål 2: Hvordan oppnå raske datamaskiner?	67
6.1.3	Forutsetninger	67
6.1.4	Læringsmål	67
6.1.5	Nøkkeltermer	67
6.2	Kommunikasjon mellom enheter og I/O	68
6.2.1	BUS	68
6.3	Eksekvering av instruksjoner	68
6.3.1	«Hent» - Fetch	68
6.3.2	«Tolk» - Decode	68

6.3.3	«Eksekver» - Execute	68
6.3.4	«Lagre» - Write Back	68
6.4	En-sykel implementasjon	68
6.5	Pipeline implementasjon	68
6.5.1	Speed-up	68
6.6	Komplikasjoner - Hazard	68
6.6.1	Ressurs hazard	68
6.6.2	Data hazard	68
6.6.3	Kontroll hazard	68
6.7	Videre henvisning	68
6.8	Oppgaver	68
6.8.1		68
7	Minnehierarki	69
7.1	Introduksjon	69
7.1.1	Mål: Hvordan minne påvirker en datamaskins hastighet.	69
7.1.2	Forutsetninger	69
7.1.3	Læringsmål	69
7.1.4	Nøkkeltermer	69
7.2	Ulike typer minne	70
7.2.1	Harddisk - HDD eller SSD	70
7.2.2	Hovedminne - RAM	70
7.2.3	Hurtigminne Cache	70
7.2.4	Register	70
7.3	Minnehierarki	70
7.3.1	Akseshastighet	70
7.3.2	Lagringskapasitet	70
7.3.3	Bruksområder	70
7.4	Minne-treff og Minne-Bom (Hit and Miss)	70
7.5	Minne mapping	70
7.5.1	Direct-mapped	70
7.5.2	Set-associative	70
7.5.3	Full-associative	70
7.6	Overskrivning og skrive strategier	70
7.6.1	Write Hit og Miss	70
7.6.2	Inkoherens	70
7.6.3	Erstatnings strategi	70
7.7	Videre henvisning	70
7.8	Oppgaver	71
7.8.1		71

Kapittel 1

Introduksjon

Dette kompendiet er i utgangspunktet skrevet som støtte til emnet IN1020 - Introduksjon til datateknologi, ved institutt for informatikk ved Universitet i Oslo Dette er et obligatorisk emne for studenter ved første semester på instituttet. Formålet med kompendiet er å gi en enkel og rask innføring i den første delen av emnet som omhandler maskinvare. Det er viktig å understreke at kompendiet forenkler enkelte teoretiske betraktninger og presenterer fagstoffet på et overordnet nivå. Vi håper kompendiet bidrar til en lettere tilegnelse og forståelse av fagstoffet.

Denne versjonen av kompendiet som nå gjøres tilgjengelig er et ikke ferdigstilt produkt. Kompendiet vil bli fortløpende bearbeidet og oppdatert gjennom høsten 2018. Vi har imidlertid valgt å gjøre dette utkastet tilgjengelig for studentene slik at de kan ha glede av kompendiet under sine studier allerede dette semesteret. Alle forslag til forbedringer av utkastet tas imot med stor takk på epost: omidmi@ifi.uio.no.

1.1 Kompendiets oppbygning

Ved utforming av kompendiet har vi valgt å fokusere på hovedemnene som behandles innenfor Maskinvare. Disse behandles hver for seg i egne kapitler i dette kompendiet. Hoveddelene er som følger:

1. Introduksjon
2. Boolsk Algebra og
3. Kretsdesign
4. Kombinatoriske kretser
5. Sekvensielle kretser
6. Datamaskinarkitektur
7. Minnehierarki

Hver kapittel vil ha følgende oppdeling:

- Sammendrag
- Introduksjon, forutsetninger og læringsmål

- Avsnitter om temaet
- Videre henvisning
- Oppgaver

For hver kapittel vil vi ha en gjennomgående rød tråd for enten en applikasjon eller en egenskap vi er på utkikk etter. Denne applikasjonen eller egenskapen fordrer at vi tilegner oss kunnskap steg for steg slik at vi har målet og motivasjonen i fokus. For å bryte opp tekster har vi inntatt grafiske innslag markert som følgende fargelagte bokser:

Oransj boks representerer forutsatt kunnskap

Eksempel 1.1

Gul boks representerer nummererte eksempler

Blå boks representerer viktig teori

Grønn boks representerer nyttige tips, visste du at...

Grå boks representerer praktiske anvendelser, applikasjoner med mer.

Rød boks representerer typiske feil og misoppfatninger

1.2 Blooms taksonomi

I hvert kapittel vil vi referere til læringsmål for emnet. Noen læringsmål er mer krevende enn andre. For å ha et systematisk forhold til dem har vi valgt å ta i bruk Blooms taksonomi for beskrivelse av læringsmålene i dette kompendiet.

I 1956 publiserte Benjamin Bloom med samarbeidspartnere Max Englehart, Edward Furst, Walter Hill og David Krathwohl et rammeverk for kategorisering av utdanningsmål[1]: Taksonomi for læringsmål, bedre kjent som Blooms taksonomi. Denne metoden er godt utbredt i alle undervisningsmiljøer.

Taksonomi består av seks hovedkategorier: Kunnskap, forståelse, anvendelse, analyse, syntese og vurdering. Kategoriene etter kategorien kunnskap ble presentert som ferdigheter og evner, med den oppfatning at kunnskap var den nødvendige forutsetningen for å ferdigheter og evner i praksis. I

tabell 1.1 har vi satt opp vår tolkning av Blooms taksonomi for dette emnet, og vi vil gjennomgående bruke denne til å referere til i læringsmålene i kompendiet.

Blooms taksonomi brukes ofte i formulering av oppgaver. Dersom oppgaven går ut på å beskrive eller definere vet du at dette antagelig er en enkel oppgave, lavt nede på skalaen av det som kreves av kunnskap. Dersom du bes om å analysere eller vurdere, vet du at dette er krevende, fordi nivået skal ligge høyt i følge taksonomien. For å få gode karakterer på et høyere studium, kreves det at du opererer i det minste delvis på de høyere nivåene. Beskrivelser og definisjoner er ikke nok.

Nivå	Kunnskap	Assosiert aktivitet
6	SKAPE? Vurdering: <i>Å kunne bedømme noe ut fra forskjellige kriterier.</i>	Bedømme, vurdere, drøfte, kritisere, diskutere, avgjøre forhold til indre/ytre kriterier. <i>Utvikle ferdigheter:</i> kombinere, improvisere, videreutvikle.
5	Evaluere? Syntese: <i>Å kunne trekke egne slutninger, utlede abstrakte relasjoner.</i>	Generalisere, organisere, oppsummere, trekke konklusjoner. <i>Komplekse ferdigheter:</i> Bearbeide, tilvirke, vise presisjon.
4	Analyse: <i>Å kunne se sammenhenger</i>	Dele opp, identifisere, sammenlikne, undersøke. <i>Vanemessige ferdigheter:</i> Gjennomføre, tilpasse, rette feil, utføre selvstendig.
3	Anvende: <i>Å kunne bruke kunnskap og forståelse i konkrete situasjoner.</i>	Fortolke, tilpasse, overføre, kunne bruke. <i>Imitasjon:</i> Gjenta, etterlikne, prøve, utøve.
2	Forstå: <i>Å kunne sammenfatte og gjengi kunnskap med egne ord.</i>	Forklare, tolke, vise til forskjeller, likheter og særtrekk. <i>Handlingsberedskap:</i> Forberede, vise interesse, benytte.

Tabell 1.1: *Blooms taksonomi*

1.3 Abstraksjonsnivå

Kunsten å håndtere komplekse systemer er at man systematiserer og strukturerer informasjon. Et moderne digitalt system er bygd opp av milliarder av transistorer og koblinger. Det er ikke noe én person kan klare å lage alene eller holde styr på. Derfor introduserer vi det som kalles for abstraksjonsnivå/-lag. I hovedsak går det ut på å oppnå et toppnivå som gjør at man kan få brukt systemet på en fornuftig måte. La oss ta en sammenligning. Se på jordkloden vår, den består av kontinenter, som igjen består av land, som igjen består av byer, som igjen består av områder, som igjen består av veinavn, som igjen består av nummer. Hver av disse kan man tenke på som et abstraksjonsnivå. Når man snakker om jordklodens bevegelser om månen, trenger vi ikke informasjon om alle veinavnene i alle landene. Det gjør at vi lettere kan betrakte jordkloden som én enhet. Tilsvarende gjør vi for datamaskinen også. Vi prøver å gjemme bort uvesentlig informasjon mellom abstraksjonsnivåene, på en måte kan vi si at vi antar at de byggeklossene vi bruker i et abstraksjonsnivå virker som de skal.

I dette kompendiet vil vi bruke 10 abstraksjonsnivåer. Oversikt over de ulike abstraksjonsnivåene er vist i tabell 1.2 på neste side. Det høyeste nivået 10 representerer abstraksjonsnivået hvor det er

Abstraksjonsnivå	Kortform beskrivelse	
10	Applikasjoner	Programvare
9	Algoritmer	
8	Programmeringsspråk	
7	Assembly språk	
6	Maskinkode	
5	Instruksjonsett arkitektur	
4	Mikro arkitektur	
3	Logikk, porter og registre	
2	Transistorer	
1	Fysikalske og kjemiske bindinger	

Tabell 1.2: *Oversikt over abstraksjonsnivåene for en typisk datamaskin.*

bruken og anvendelser man bearbeider, mens nivå 9 er det algoritmene disse applikasjonene utfører. For å kunne skrive algoritmer er vi nødt til å bruke et programmeringsspråk og de som utvikler nye programmeringsspråk jobber i nivå 8. Ofte er programmeringsspråk en måte for oss mennesker for å kunne kommuniserer og ikke nødvendigvis det datamaskinen forstår. For å få til det må programmeringsspråket oversettes til det maskinen forstår og til det er det henholdsvis nivå 7 og 6 som er gjeldende. Med maskinkode, milliarder av bare 0 og 1, kan maskinvaren styres. Nivåene 6 til 10 betegnes som programvare, mens nivåene 1 til 4 betegnes som maskinvare. Felles for begge er nivå 5 som tar for seg en arkitektur som er bygd opp av instruksjonsett. Det vil vi komme nærmere inn på i siste delen av kompendiet.

I avsnitt 1.1 på side 5 presenterte vi kompendiets oppbygning og den følger rimelig godt abstraksjonsnivåene 3, 4 og 5. Vi starter med å presentere boolsk algebra, logikk og kretsdesign som er i nivå 3. Selv om det ikke er pensum i emnet IN1020 vil det være henvisninger til mer dybdekunnskap om nivå 2 i avsnitt 2.7 på side 49.

Det er ikke slik at man kan bli en ekspert på et abstraksjonsnivå uten å ha god innsikt i nivåene over og under. For eksempel det å bli en dyktig programmerer (abstraksjonsnivå 8 og 9), må du ha god innsikt i hvordan datamaskinen lagrer informasjon og hvordan en CPU utfører instruksjoner og operasjoner (som er nivå 4 og 5). En CPU (engelsk forkortelse for Central Processing Unit), også kalt prosessor, er hovedregne-/prosesseringsenheten i en datamaskin som utfører instruksjonene gitt i et dataprogram, og den er det primære elementet som gjennomfører datamaskinens funksjoner. I tillegg må du ha god innsikt hvordan applikasjonene brukes av brukere (nivå 10++).

muligheter for å tegne tabellen som en omvendt trekant?

1.4 Kompendium under arbeid

Kompendiet versjon fra: 8. september 2018

1.4.1 Bidragsyttere

En stor takk til alle som har bidratt med gode innspill.

Kapittel 2

Boolsk Algebra

Dagens teknologi og elektronikk er i stand til å gjøre mange kompliserte oppgaver, og vi omgir oss daglig med digitale elektroniske hjelpemidler. Hvordan er det mulig å få dette til når alt bare egentlig er 0'ere og 1'ere? Hvordan kan vi sette sammen mange nok 0'ere og 1'ere til å lage en avansert datamaskin? Til dette brukes en matematisk teori som heter Boolsk Algebra, og gjennom funksjonsuttrykk kan vi designe avanserte kretser. En Boolsk krets består av binære porter som utfører en bestemt funksjon med 0'ere og 1'ere. En krets eller elektronisk krets er sammensatt av komponenter som drives av strøm og spenninger. I dette kapitlet tar vi for oss grunnleggende boolsk algebra med fokus på kretsdesign.

Forutsetter kjennskap til:

- To-tallsystemet.
- Binære tall.
- Konvertering mellom desimaltall og binære tall.

2.1 Introduksjon

Majoriteten av all elektronikk vi omgir oss med består av transistorer. I digital elektronikk kan vi forenklet se på transistorer som brytere. Disse bryterne er enten AV eller PÅ. Det er nettopp dette som er broen mellom boolsk algebra og digital elektronikk. Hvordan kan man sette opp slike binære¹ brytere til å gjøre fornuftige ting? Hvordan kan vi lage store kompliserte systemer bare ved å bruke 0 og 1. For å svare på disse spørsmålene og enda flere trenger vi å lære litt om hvordan vi kan lage en oppskrift, kjøreregel eller en formell metode. Det er her matematikken kommer til hjelp. I 1854 presenterte George Booleen ny teori for logikk [2], Boolean Algebra (norsk Boolsk algebra), som er grunnlaget for logisk resonnering. Logikk er regler som gir sammenheng og forutsigbare resultater. Det var først i 1938 at Claude Shannon, fra M.I.T., viste at denne type algebra kan løse

¹Binær brukes for et to-tallsystem.

problemer med kretsdesign [3]. Slik teknologien har utviklet seg siden 1938 har Boolsk algebra vist seg å ha en sentral rolle innenfor:

Analyse – med boolsk algebra kan man på en lett og effektiv måte å beskrive funksjonen eller virkemåten til en digital krets.

Design – med en gitt funksjonsbeskrivelse kan boolsk algebra bidra til å effektivisere og forenkle et kretsdesign.

En grundig gjennomgang av Boolsk algebra er en krevende og komplisert og faller utenfor dette faget. Vi skal imidlertid fokusere på anvendelsen av Boolsk algebra med sikte på å designe og analysere kretsdesign. Vi bruker derfor Boolsk algebra med 0 og 1, som ofte også kalles binær logikk.

De fleste lærebøker starter med å introdusere logikk og funksjoner for å bygge opp en plattform innen funksjonell beskrivelse av et system. Vårt fokus på kretsdesign og analyse gjør at vi vil ha en annen innfallsvinkel. Vi trekker inn teori der vi trenger den for å kunne beskrive eller forenkle vårt design. Dette gjør at vi kommer til å ha vesentlig mindre fokus på det teoretiske og matematiske aspektet for logikk, men heller mer av det praktiske og anvendbare. Vårt mål for dette kapitlet er å forstå og kunne beskrive en 7-segment skjerm med funksjonsuttrykk i Boolsk algebra.

Kretsdesignere ønsker alltid å starte med å forenkle verden, og et av de mest kjente tankegangene er å starte med en *black box*, som i Figur 2.1. Her er man opptatt av hvilke signaler denne boksen får (inngangssignaler) og hva boksen skal påvirke videre (utgangssignaler). Det som foregår inni boksen er i utgangspunktet at man gjennomfører en funksjonell håndtering av inngangssignal(ene), kalt en funksjon, som gir verdier til utgang(ene). Antall innganger og utganger varierer ut fra funksjon til funksjon.



Figur 2.1: *Et klassisk utgangspunkt for å designe en krets. Man starter med å tenke at en krets er en boks som skal utføre en funksjon og har et sett av inngang(er) og utgang(er).*

2.1.1 Mål: En 7-segment skjerm

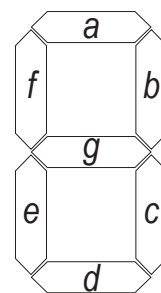
Vårt mål for dette kapitlet er å tilegne oss nok kunnskap om Boolsk algebra slik at vi kan beskrive funksjonen for å styre lysdiodene i en 7-segment skjerm. En typisk 7-segment skjerm er avbildet i Figur 2.2 på neste side og består av 7 lysdioder representert med a , b , c , d , e , f og g . Dette er den enkleste formen for skjerm og brukes ofte for å lese ut verdier i en prototype. En slik skjerm har faktisk 127 mulige måter å lyses opp på. Vårt mål er å bruke denne 7-segment skjermen til å vise desimaltallene $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$. Vi trenger da innganger, binære signaler, som kan representere de

angitte desimaltallene og utganger som signaliserer hvilke av lysdiodene som skal PÅ. Et forslag for et *black box* startpunkt er vist i Figur 2.3. Vi trenger fire binære innganger for å kunne representere 0-9 i desimaltall og kaller inngangene for (T_0, T_1, T_2, T_3) , hvor T står for *Tall* og indekseringen representerer signifikans². Det vi trenger å gjøre er å kunne beskrive funksjonen til denne svarte boksen og deretter implementere den som en krets. Før vi kan lage en krets må vi forstå hvordan vi kan endre utgangssignalene basert på verdien av inngangssignalene. Vi trenger å sette opp en tabell som viser oss hvilke verdier vi kan ha på inngangssignalene våre og hva er så deres korresponderende utgangssignaler.



Figur 2.3: En «black box» for logikk og styring av en 7-segment skjerm hvor inngangen er en 4-bits binært tall og hvor $(S_a, S_b, S_c, S_d, S_e, S_f, S_g)$ referer til lysdiodene.

Inndelingen videre i dette kapitlet er basert på de trinnene det er naturlig å gå igjennom for å kunne lage et funksjonsuttrykk som danner grunnlaget for et kretsdesign. I neste seksjon, Avsnitt 2.2 på neste side, starter vi med å beskrive en sannhetsverditabell som er en visuell måte å sette opp hvordan boksens innganger og utganger henger sammen, altså hva de ytre omgivelsene forventer fra boksens funksjon. Gitt en oppsatt sannhetsverditabell kan man gå videre til å beskrive boksens funksjon, se Avsnitt 2.3 på side 15, og deretter se på muligheter for å oppnå en mer effektiv funksjon gjennom forenklinger, se avsnitt Avsnitt 2.4 på side 26. Forenklinger av funksjoner gir direkte effektivisering av arealforbruk, strøm osv. Det er først under Kapittel 3 på side 53 vi ser på kretsdesignet av denne 7-segment skjermen.



Figur 2.2: En 7-segment skjerm hvor (a, b, c, d, e, f, g) referer til lysdiodene for det angitte stedet.

2.1.2 Læringsmål

Bloom's 1: Ha grunnleggende kunnskap om «black box» tankegang, oppsett av sannhetsverditabell, oppsett av Karnaugh diagram

Kunne gjengi regneregler, DeMorgans teorem og de basis funksjonene NOT, AND og OR.

²Vi definerer T_3 som den mest signifikante bit (MSB) og T_0 som det minst signifikante bit (LSB).

Bloom's 2: Forstå behovet for funksjonsuttrykk og forenkling av uttrykk.

Bloom's 3: Kunne anvende sannhetstabell for å beskrive en funksjon.

Bloom's 4: Kunne analysere en sannhetsverditabell og skrive opp et funksjonsuttrykk.

Bloom's 5: Kunne bruke Karnaugh diagram til å utlede forenklinger av funksjonsuttrykk.

Bloom's 6: Kunne bruke regneregler til å verifisere og forenkle funksjonsuttrykk.

2.1.3 Nøkkeltermer

Buffer, inverter, AND, OR, NOT, NAND, NOR, XOR, XNOR, Funksjonsuttrykk, Sannhetsverditabell, Sannhetstabell, Forenkling av uttrykk, DeMorgans Teorem, Regneregler, Karnaugh diagram.

2.2 Sannhetsverditabell

Sannhetsverditabell eller sannhetstabell er en måte å beskrive en funksjon sin virkemåte, hvor man i hovedsak ser på hvilke innganger som gir sant (1) eller usant (0) på utgangen. I en sannhetstabell setter man opp alle mulige kombinasjoner av verdiene for inngangssignalene og tilhørende verdier for utgangssignalene. For ett inngangssignal vil det være 2 mulige verdier, enten 0 eller 1, mens det for to inngangssignaler vil det være fire³ muligheter. Det er faktisk en sammenheng mellom hvor mange innganger det er og hvor mange mulige kombinasjoner av inngangsverdier vi kan ha. Denne relasjonen er matematisk beskrevet som 2^n relasjon, hvor n er antall innganger. Siden vi har binære signaler og hver inngang enten er 0 eller 1, kaller vi det for *bit* og ofte brukes antall inngangsbit til å gi navn til et system.

For en krets med tre innganger (3-bit) har vi da: $2^3 = 2*2*2 = 8$ mulige kombinasjoner av inngangsverdier, mens for en krets på fem innganger har vi $2^5 = 2*2*2*2*2 = 32$ mulige kombinasjoner.

Eksempel 2.2 Hvor mange kombinasjoner av verdier på inngangssignalene for en 32-bits krets?

Det vi vet er at en 32-bits funksjon gir: $2^{32} = 2 * 2 = 4 294 967 296$

³De fire mulighetene er 00 01 10 og 11.

Inn	Ut
0	(?)
1	(?)

Inn ₁	Inn ₂	Ut
0	0	(?)
0	1	(?)
1	0	(?)
1	1	(?)

(a) 1-inngang og 1-utgang

(b) 2-inngang og 1-utgang

Tabell 2.1: Sannhetstabellen for henholdsvis 1- og 2-innganger.

Det betyr at for en 32-bits krets vil kunne ha ≈ 4.3 milliarder mulige verdikombinasjoner av inngangssignaler ☹.

Enn så lenge tar vi for oss kretser med bare på ett utgangssignal og rent visuelt kan vi i en tabell/matrise sette opp alle inngangssignalene etterhverandre i hver sin kolonne adskilt med en strek før vi setter opp utgangssignalet. Matrisen fylles opp ved at radene er verdier (0,1) for de representative signalene. I Tabell 2.1 er sannhetsverditabellen for en én-inngang og en to-inngangs funksjon satt opp. Foreløpig har vi satt utgangene til å være ukjent (?) fordi de enda ikke er fastsatt eller deklareret⁴. Måten å lese en sannhetsverditabell består av å lese en og en rad nedover. Høytlesningen av Tabell 2.1a blir: *Inn 0 gir (?), mens Inn 1 gir (?)*, mens for Tabell 2.1b blir: *Inn₁ 0 og Inn₂ 0 gir (?)*. La oss først ta utgangspunkt i en 1-inngangs krets, Tabell 2.1a, og undersøke hvilke mulige utgangssignaler vi kan få. Ettersom utgangssignalet kan ha verdi enten 0 eller 1, åpner dette opp for å kunne få to utgangssignaler for hver verdi av inngangssignalet. Vi kan sette dette opp som en tabell vist i Tabell 2.2. Ser vi isolert på **Ut₁** virker det som om utgangssignalet er det

Inn	Ut ₁	Ut ₂
0	0	1
1	1	0

Tabell 2.2: Mulige utgangsverdier for en én-inngang krets.

samme som inngangssignalet, altså at den kopier inngangen til utgangen. **Ut₁** vil ha en funksjon vi kaller for *buffer*. Et buffer er en krets som gir ut det samme signalet som på inngangen, men buffere (forsterker/oppfrisker) signalet. Det blir ingen endring i logikken av en krets, men det er en vesentlig endring på det elektriske plan⁵. Tar vi for oss **Ut₂** er utgangen det motsatte av inngangen, altså når **Inn** = 0 er **Ut₂** = 1, denne funksjonen heter *inverter*.

Det er ikke flere mulige funksjoner som bare har en inngang, men derimot mange flere når vi har flere inngang. Tabell 2.3 på neste side lister opp alle mulige utgangskombinasjoner for en 2-inngangskrets. Som det fremkommer av tabellen er noen av utgangene representert ved kjente funksjoner (disse kommer vi tilbake). De utgangskombinasjonene som ikke har en kjent funksjon blir forklart i Avsnitt 2.3 på side 15 der vi viser hvordan vi kan skrive opp et funksjonsuttrykk for funksjonen. Sannhetsverditabell har en sentral rolle når det kommer til å sette opp en systematisk

⁴Her mener vi at funksjonen ikke er bestemt.

⁵For mer informasjon om dette må man ned i abstraksjonsnivået, nivå 2 i Tabell 1.2 på side 8, og betrakte transistorer med strøm og spenning.

Inn ₁	Inn ₂	Utgangskombinasjoner																
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	
Funksjon		AND							XOR	OR	NOR	XNOR						NAND

Tabell 2.3: Oversikt over alle mulige utgangsfunksjoner gitt 2-innganger.

beskrivelse for en black box er mest brukervennlig for bokser med få innganger. Vi kan bruke sannhetsverditabell for å sette opp matematiske uttrykk som beskriver funksjonen til black box, mer om dette i Avsnitt 2.3 på neste side.

2.2.1 Sannhetsverditabell med mange innganger og utganger

Det er ingen begrensning for hvor mange innganger og utganger en krets har når det kommer til å sette opp en sannhetsverditabell og heller ikke hva vi døper signalene til. Omdøpingen av signalene kaller vi for variabler og det er mest utbredt å bruke små bokstaver som inngangssignaler og store bokstaver som utgangssignal⁶. Rekkefølgen på inngangen spiller ingen rolle når det settes opp i en sannhetsverditabell, men man ønsker at bokstavene kommer i alfabetisk rekkefølge.

Eksempel 2.3 Hvordan settes en 3-inngang og 2-utgang sannhetsverditabell opp?

a	b	c	F ₁	F ₂
0	0	0	?	?
0	0	1	?	?
0	1	0	?	?
0	1	1	?	?
1	0	0	?	?
1	0	1	?	?
1	1	0	?	?
1	1	1	?	?

Det er en sammenheng mellom det å sette opp en sannhetsverditabell med hensyn på alle inngangskombinasjoner og telle på binært. Hvis du ser av Tabell 2.1 på side 13 for inngangsverdiene så kan du telle nedover radene. For eksempel er radene gitt som:

$$\begin{aligned}
 00_2 &= 0_{10} \\
 01_2 &= 1_{10} \\
 10_2 &= 2_{10} \\
 11_2 &= 3_{10}
 \end{aligned}$$

⁶Dette er sannhet med modifikasjoner, enkelte kretser har variabler som inneholder informasjon.

Denne sammenhengen kaller vi for *minterm* og det skal vi komme tilbake til senere.

2.2.2 Sannhetsverditabellen for en 7-segment skjerm

Nå som vi har vært igjennom hvordan en sannhetsverditabell kan settes opp med flere innganger og utganger er det mulig for oss å sette opp en tilsvarende tabell for en 7-segment skjerm. Basert på *black box* fremstillingen i Figur 2.3 på side 11 får vi innganger som er T_0 , T_1 , T_2 og T_3 , mens utgangene er a, b, c, d, e, f, g. For hver rad setter vi inn hvilke av lysdiodene som skal være på for den angitte raden. De lysdiodene som skal lyse får verdien 1 på utgangen og de som skal være av får 0 på utgangen. Sannhetsverditabellen for en 7-segment skjerm er vist i Tabell 2.4. Legg merke til at vi alle utgangene er 0 for inngangskombinasjoner for desimaltall 10 til desimaltall 15, som betyr at det ikke er noe lys på skjermen for disse verdiene.

Desimal	T_3	T_2	T_1	T_0	S_a	S_b	S_c	S_d	S_e	S_f	S_g
0 ₁₀	0	0	0	0	1	1	1	1	1	1	0
1 ₁₀	0	0	0	1	0	1	1	0	0	0	0
2 ₁₀	0	0	1	0	1	1	0	1	1	0	1
3 ₁₀	0	0	1	1	1	1	1	1	0	0	1
4 ₁₀	0	1	0	0	0	1	1	0	0	1	1
5 ₁₀	0	1	0	1	1	0	1	1	0	1	1
6 ₁₀	0	1	1	0	1	0	1	1	1	1	1
7 ₁₀	0	1	1	1	1	1	1	0	0	0	0
8 ₁₀	1	0	0	0	1	1	1	1	1	1	1
9 ₁₀	1	0	0	1	1	1	1	0	0	1	1
10 ₁₀	1	0	1	0	0	0	0	0	0	0	0
11 ₁₀	1	0	1	1	0	0	0	0	0	0	0
12 ₁₀	1	1	0	0	0	0	0	0	0	0	0
13 ₁₀	1	1	0	1	0	0	0	0	0	0	0
14 ₁₀	1	1	1	0	0	0	0	0	0	0	0
15 ₁₀	1	1	1	1	0	0	0	0	0	0	0

Tabell 2.4: Sannhetsverditabellen for en 7-segments display

2.3 Funksjonsuttrykk

En sannhetsverditabell med mange innganger er ofte en svært krevende prosess å sette opp og til tider vanskelig å se mønster eller finne ut av hva kretsen gjør. Boosk algebra er nyttig i denne sammenheng fordi den gir oss muligheten for å kunne beskrive en krets på en formell måte. Denne formelle måten er det vi kaller for et *funksjonsuttrykk*. Et funksjonsuttrykk inneholder nøyaktig den samme informasjonen som en sannhetsverditabell, men gjør det lettere å kunne bearbeide og effektivisere et kretsdesign. Først trenger vi å etablere hvordan vi kan sette opp et funksjonsuttrykk for en krets for så å forenkle uttrykket med regneregler.

Én sannhetsverditabell kan ha mange funksjonsuttrykk, men et funksjonsuttrykk har kun én sannhetsverditabell.

2.3.1 Basis funksjoner

Vi kan starte med den enkleste funksjonen, en 1-inngangs krets. Som vi så av Tabell 2.2 på side 13 er det to mulige kretser. Den ene er en buffer som da vil ha et funksjonsuttrykk som følgende:

$$Ut_1 = Inn \quad (2.1)$$

Et buffer er ikke særlig egnet for å kunne gjøre logikk funksjoner, så derfor blir også funksjonsuttrykket det samme som inngangssignalet. Det er på det elektriske plan at en buffer har en nyttig funksjon. Ser vi heller til en inverter, Tabell 2.2 på side 13 med Ut_2 , så kan funksjonsuttrykket skrives som følger:

$$Ut_2 = \overline{Inn} \quad (2.2)$$

Streken over \overline{Inn} betyr at det er den inverterte verdien av inngangssignalet. Det er flere måter å skrive inverteringen på, litt avhengig av hva man foretrekker. I digital elektronikk brukes følgende to skrivemåter:

$$\begin{aligned} Ut_2 &= \overline{Inn} \\ Ut_2 &= Inn' \end{aligned} \quad (2.3)$$

mens det i logikk skrives på formen:

$$Ut_2 = \neg Inn \quad (2.4)$$

En inverterende krets kalles for *inverter* og den boolske funksjonen betegnes som **NOT**. En inverter er en 1-inngangs krets og en av tre elementære boolske funksjoner. De to andre funksjonene heter **AND** (norsk: *OG*) og **OR** (norsk: *ELLER*) og har mer enn én inngang. Følger vi av konvensjonen for å skrive en sannhetsverditabell fra Avsnitt 2.2.1 på side 14 kan vi sette opp funksjonsuttrykket for en AND-funksjon som vist i Tabell 2.5 på neste side. I logikk er denne AND-funksjonen betegnet som *hvis og bare hvis* og med henvisning til 0 = usant og 1 = sant for en 2-inngangs AND gir følgende beskrivelse:

Alt burde skrives med små a,b,c, men F kan være stor.

Det er sant *hvis og bare hvis* begge inngangene er sanne. Ellers er det usant.

En generalisering av uttrykket er:

Det er sant *hvis og bare hvis* alle inngangene er sanne. Ellers er det usant.

Det er en noen likheter mellom den logiske funksjonen og det å tenke aritmetisk som *multiplikasjon*. Derfor betegnes AND-funksjonen som en \cdot og skrives som $a \cdot b$. I logikk omtales AND som union og i mengdelære med symbolet \wedge . Ser vi på verdien stemmer også følgende matematiske uttrykk i Tabell 2.6 på neste side. Fra sannhetsverditabellen i tabell Tabell 2.5 på neste side og Tabell 2.6 på neste side ser vi at det eneste som gir $F = 1$ er hvis begge inngangene er 1. Til forskjell for aritmetisk matematikk, så leses dette som en *og* en er lik en. Funksjonsuttrykket til en AND kan skrives på følgende måte:

a	b	F_{AND}
0	0	0
0	1	0
1	0	0
1	1	1

Tabell 2.5: Sannhetsverditabellen for 2-inngangs AND-funksjon med inngangene er a , b og utgangen F .

a	·	b	=	F_{AND}
0	·	0	=	0
0	·	1	=	0
1	·	0	=	0
1	·	1	=	1

Tabell 2.6: En visualisering av en 2-inngangs AND-funksjon satt opp som en sannhetsverditabell.

$$\begin{aligned}
 F &= a \cdot b \\
 F &= ab \\
 F &= a \wedge b \quad (\text{logikk})
 \end{aligned}
 \tag{2.5}$$

Den tredje og siste basis funksjonen er en OR-funksjonen. I logikk oversettes dette til *eller* og med henvisning til 0 = usant og 1 = sant for en 2-inngangs OR gir:

Det er sant hvis inngang 1 *eller* inngang 2 er sann. Ellers er det usant.

En generalisering av uttrykket er:

Det er sant hvis en av inngangene sanne. Ellers er det usant.

Det er en sammenheng mellom den logiske funksjonen og det å tenke aritmetisk som *addisjon*, men i mindre grad som for AND og multiplikasjon. Symbolet som brukes for OR er '+' og i logikk brukes symbolet '∨'. I Tabell 2.7 ser vi en slags sannhetsverditabellen for en OR-funksjon:

a	+	b	=	F_{OR}
0	+	0	=	0
0	+	1	=	1
1	+	0	=	1
1	+	1	=	1

Tabell 2.7: En visualisering av en 2-inngangs OR-funksjon satt opp som en sannhetsverditabell.

Som det fremkommer av tabellen blir ikke $1 + 1 = 2$, men $1 + 1 = 1$. Dette er den største forskjellen

mellom en aritmetisk addisjon og en logisk OR-funksjon. Høytlesingen av $1 + 1 = 1$ er en *eller* en gir en. Funksjonsuttrykket for Tabell 2.7 på side 17 er:

$$\begin{aligned} F_{OR} &= a + b \\ F_{OR} &= a \vee b \quad (\text{logikk}) \end{aligned} \tag{2.6}$$

Nå har vi vært igjennom de tre elementære boolske funksjonene og med disse kan vi nå bygge opp mer kompliserte funksjonsuttrykk.

Det er mulig å implementere en hvilken som helst funksjon, logisk eller aritmetisk, ved kun å bruke de tre elementære boolske funksjonene NOT, AND og OR.

2.3.2 NAND og NOR funksjoner

Over 90% av all digital elektronikk inneholder en krets som heter NAND. Grunnen til at denne kretsen er å foretrekke har med dets elektriske egenskaper å gjøre. En NAND-funksjon er en inverterende funksjon av AND og skrives:

$$\begin{aligned} F_{NAND} &= \overline{a \cdot b} \\ F_{NAND} &= \overline{ab} \end{aligned} \tag{2.7}$$

med tilhørende sannhetsverditabell som vist i Tabell 2.8:

a	b	F_{NAND}
0	0	1
0	1	1
1	0	1
1	1	0

Tabell 2.8: Sannhetsverditabellen for en 2-inngangs NAND.

Tilsvarende finnes den en NOR-funksjon som er den inverterte funksjonen av en OR. Funksjonsuttrykket for en 2-inngangs NOR er beskrevet som følger:

$$F_{NOR} = \overline{a + b} \tag{2.8}$$

med tilhørende sannhetsverditabell som vist i Tabell 2.9:

a	b	F_{NOR}
0	0	1
0	1	0
1	0	0
1	1	0

Tabell 2.9: Sannhetsverditabellen for en 2-inngangs NOR.

Det er mulig å implementere en hvilken som helst funksjon, logisk eller aritmetisk, ved kun å bruke enten bare NAND eller bare NOR funksjon.

Eksempel 2.4 Hva er sannhetsverditabellen for en 3-bits NAND og NOR?

a	b	c	F_{NAND}	F_{NOR}
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	0

2.3.3 Odde funksjoner: XOR og XNOR

Enkelte ganger ønsker vi å kunne ekstrahere ut gitte signaler og til det trenger vi såkalte ulineære eller odde funksjoner. I boolsk algebra har vi to unike funksjoner for dette, nemlig XOR og XNOR som utfyller hverandre godt. XOR er en exclusive-OR som betyr at den er sann for odde antall sanne innganger, men XNOR er den inverterte XOR som da gir sann når det er et partall av sanne innganger. Disse funksjonene har egne symboler og en 2-inngangs XOR og XNOR skrives som:

$$\begin{aligned} F_{\text{XOR}} &= a \oplus b \\ F_{\text{XNOR}} &= \overline{a \oplus b} \end{aligned} \tag{2.9}$$

hvor sannhetsverditabellen for XOR og XNOR er vist i tabell Tabell 2.10:

a	b	F_{XOR}	F_{XNOR}
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Tabell 2.10: Sannhetsverditabellen for 2-inngangs XOR og XNOR.

Tilsvarende som for alle de foregående funksjonene kan disse også ha flere innganger.

2.3.4 Andre sammensatte funksjoner

Vi kan se tilbake til sannhetsverditabellen i Tabell 2.3 på side 14 som listet opp alle mulige utgangskombinasjoner for en 2-variabels funksjon. I de foregående avsnittene har vi gått igjennom de kjente funksjonsuttrykkene. Hvordan kan vi uttrykke de resterende som egne funksjonsuttrykk? For

å få til dette må vi sette sammen kjente funksjoner for å kunne uttrykke et nytt funksjonsuttrykk. I Tabell 2.11 er tabellen revidert i forhold til variabel konvensjonen vi bruker⁷.

a	b	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Funksjon		AND						XOR	OR	NOR	XNOR					NAND	

Tabell 2.11: Logiske funksjoner gitt alle mulige inngangskombinasjoner for en 2-inngangs krets.

Når vi skal skrive et funksjonsuttrykk er vi kun interessert i hvilke tilfeller utgangssignalet gir 1. For hver inngangskombinasjon som gir 1 på utgangen vil vi få et ledd i funksjonsuttrykket som representerer inngangssignalenes verdi. Vi kan se på utgangsfunksjonen for F_1 i Tabell 2.11, som gir oss 0 på utgangen uansett hva inngangskombinasjonene er, det betyr at vi ikke får noe ledd som inneholder en kombinasjon av inngangssignalene og dermed blir funksjonen lik 0 uansett hva inngangene er:

$$F_1 = 0 \quad (2.10)$$

Ser vi på F_2 har vi bare ett tilfelle der utgangen er 1 og det er for inngangene $a = 1$ og $b = 1$. Det vil derfor gi oss kun ett ledd i funksjonsuttrykket og vi kan starte med å skrive det på følgende måte:

$$F_2 = 1 \text{ ved verdier av inngangssignalene } \{a = 1 \text{ og } b = 1\} \quad (2.11)$$

For å kunne beskrive verdien av inngangssignalene som variabler i et funksjonsuttrykk må vi definere hva verdien for en variabel er. Definisjonen er som følger:

Verdien 0 for et binært signal, (a) , skrives som $\overline{(a)}$.
Verdien 1 for et binært signal, (a) , skrives som (a) .

For å kunne skrive Likning (2.11) som et funksjonsuttrykk setter vi for definisjonen av verdien av inngangssignalene a og b til å bli:

$$F_2 = a \text{ og } b \quad (2.12)$$

Vi erstatter *og* med **AND** (\cdot) og setter inn i Likning (2.12) som gir følgende endelige funksjonsuttrykk:

$$F_2 = a \cdot b \quad (2.13)$$

Tilsvarende fremgangsmetode kan vi bruke for alle funksjonene i tabellen. Nedenfor er funksjonsuttrykket for F_3 vist:

$$F_3 = 1 \text{ ved verdier av inngangssignalene som } \{a = 1 \text{ og } b = 0\} \quad (2.14)$$

ved å sette inn variablene vil gi funksjonsuttrykket:

$$F_3 = a \cdot \bar{b} \quad (2.15)$$

⁷Vi har omdøpt Inn_1 til a og Inn_2 til b .

Ser vi på F_4 er det to tilfeller som gir utgangen lik 1. Den ene er for inngangskombinasjonen $a=1$ og $b=1$ (11) og den andre for $a=1$ og $b=0$ (10). For hver rad på utgangen som er 1 vil vi få et ledd i funksjonsuttrykket med variablene. Funksjonsuttrykket for F_4 vil da kunne formuleres som:

$$F_4 = 1 \text{ ved verdier av inngangssignalene som } \begin{cases} a = 1 \text{ og } b = 0 \\ a = 1 \text{ og } b = 1 \end{cases} \quad (2.16)$$

ved å sette inn for variablene får vi:

$$F_4 = 1 \text{ ved variabler som } \begin{cases} a \cdot \bar{b} \\ a \cdot b \end{cases} \quad (2.17)$$

Måten å flette dette sammen på er som følger (muntlig): «F er sann når a **AND** \bar{b} **ELLER** når a **AND** b , som gir da følgende endelige funksjonsuttrykk:

$$\begin{aligned} F_4 &= (a \cdot \bar{b}) + (a \cdot b) \\ F_4 &= a\bar{b} + ab \end{aligned} \quad (2.18)$$

Slik kan vi fortsette gjennom hele tabellen og de er gitt som oppgaver i slutten av dette kapittelet.

Eksempel 2.5 Hva er funksjonsuttrykket for F_6

Vi kan sette opp sannhetsverditabellen for F_6 , som blir:

a	b	F_6
0	0	0
0	1	1
1	0	0
1	1	1

Vi kan lese av de radene hvor utgangen, F_6 , er 1 som gir oss:

$$F_6 = 1 \text{ ved verdier av inngangssignalene som } \begin{cases} a = 0 \text{ og } b = 1 \\ a = 1 \text{ og } b = 1 \end{cases} \quad (2.19)$$

ved å sette inn for variablene får vi:

$$F_6 = 1 \text{ ved variabler som } \begin{cases} \bar{a} \cdot b \\ a \cdot b \end{cases} \quad (2.20)$$

hvor funksjonsuttrykket blir:

$$\begin{aligned} F_6 &= \bar{a} \cdot b + a \cdot b \\ F_6 &= \underline{\underline{\bar{a}b + ab}} \end{aligned} \quad (2.21)$$

Fra sannhetsverditabellen kan vi se at utgangen egentlig bare følger inngangen b . Kunne vi ha skrevet funksjonsuttrykket som $F_6 = b$?

La oss se på utgangen F_{15} som vi vet er en NAND funksjon hvor $F_{\text{NAND}}(\overline{ab}) \equiv \overline{(a \cdot b)}$. For vi et så enkelt uttrykk når vi leser ut av sannhetsverditabellen hvor utgangen er 1 for tre rader? Vi starter med å skrive opp det vi kan lese ut fra tabellen: funksjonen gir 1 for inngangskombinasjonene 00, 01 og 10, som da gir følgende:

$$F_{15} = 1 \text{ ved verdier av inngangssignalene som } \begin{cases} a = 0 \text{ og } b = 0 \\ a = 0 \text{ og } b = 1 \\ a = 1 \text{ og } b = 0 \end{cases} \quad (2.22)$$

setter inn i variablene:

$$F_{15} = 1 \text{ ved inngangsverdier som } \begin{cases} \bar{a} \cdot \bar{b} \\ \bar{a} \cdot b \\ a \cdot \bar{b} \end{cases} \quad (2.23)$$

som gir da følgende funksjonsuttrykk:

$$\begin{aligned} F_{15} &= (\bar{a} \cdot \bar{b}) + (\bar{a} \cdot b) + (a \cdot \bar{b}) \\ F_{15} &= \bar{a}\bar{b} + \bar{a}b + a\bar{b} \end{aligned} \quad (2.24)$$

Vi endte ikke opp om det samme funksjonsuttrykket? Det har med at det er mulig å forenkle funksjonsuttrykket og tilslutt få at $F_{15} = F_{\text{NAND}} = \overline{AB}$. Dette er også et godt eksempel på teorien tidligere:

Én sannhetsverditabell kan ha mange funksjonsuttrykk, men et funksjonsuttrykk har kun én sannhetsverditabell.

Som teorien sier så finnes det faktisk mange forskjellige måter å uttrykke en funksjon på, noen enklere og bedre enn andre. I elektronikkens verden er det ofte et sterkt ønske om å bruke så lite areal som mulig, som forenkles til vårt abstraksjonsnivå til å være så få ledd og variable som mulig. Vi skal i Avsnitt 2.4 på side 26 se på fremgangsmetoden for hvordan vi kan forenkle funksjonsuttrykk med hjelp av et sett regneregler. Før vi går videre til neste kapittel lister vi opp alle utgangsfunksjonene

for Tabell 2.11 på side 20 i Likning (2.25).

$$\begin{aligned}
 F_1 &= 0 \\
 F_2 &= ab \\
 F_3 &= a\bar{b} \\
 F_4 &= [a\bar{b} + ab] = a \\
 F_5 &= \bar{a}b \\
 F_6 &= [\bar{a}b + ab] = b \\
 F_7 &= \bar{a}b + a\bar{b} \\
 F_8 &= \bar{a}b + a\bar{b} + ab \\
 F_9 &= \bar{a}\bar{b} \\
 F_{10} &= \bar{a}\bar{b} + ab \\
 F_{11} &= [\bar{a}\bar{b} + a\bar{b}] = \bar{b} \\
 F_{12} &= \bar{a}\bar{b} + a\bar{b} + ab \\
 F_{13} &= [\bar{a}\bar{b} + \bar{a}b] = \bar{a} \\
 F_{14} &= \bar{a}\bar{b} + \bar{a}b + ab \\
 F_{15} &= \bar{a}\bar{b} + \bar{a}b + a\bar{b} \\
 F_{16} &= 1
 \end{aligned} \tag{2.25}$$

Eksempel 2.6 Hva er funksjonsuttrykket for følgende sannhetsverditabell?

a	b	c	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Vi kan starte med å finne ut av hvilke inngangsverdier som gir utgangen lik 1, altså når $F = 1$. I tabellen under har vi markert hvilke inngangsverdier som er av interesse. For hver inngangsverdi som er markert vil det gi oss et ledd i det endelige funksjonsuttrykket.

a	b	c	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Vi kan starte med å skrive funksjonsuttrykket som:

$$F = 1 \text{ ved } \begin{cases} a = b = 0 & \text{og } c = 1 \\ a = 0 & \text{og } b = c = 1 \\ a = 1 & \text{og } b = c = 0 \\ a = c = 1 & \text{og } b = 0 \end{cases} \quad (2.26)$$

som da kan skrives opp på følgende måte:

$$\underline{F = \bar{a}\bar{b}c + \bar{a}bc + a\bar{b}c + abc} \quad (2.27)$$

Det er ofte at man ikke har tilgang til informasjon om en krets og det fordrer til å kunne beregne baklengs for å finne informasjonen. På engelsk kalles dette for *reverse engineering*. Som et eksempel på dette kan vi anta at vi har et funksjonsuttrykk, men ikke sannhetsverditabellen. I eksemplet under tar vi for oss hvordan det kan gjøres.

Eksempel 2.7 Hva er sannhetstabellen til $F = \bar{a}\bar{b}c + \bar{a}bc + a\bar{b}c + abc$

Det vi vet er at $F = 1$ når minst en av de oppgitte leddene i funksjonsuttrykket er 1. Dermed kan vi dele opp funksjonsuttrykket slik at vi kan lettere oversette det til en sannhetsverditabell.

$$F = 1 \text{ ved } \begin{cases} a = b = c = 0 \\ a = c = 0 & \text{og } b = 1 \\ a = b = 1 & \text{og } c = 0 \\ a = b = c = 1 \end{cases} \quad (2.28)$$

Vi setter inn i sannhetsverditabellen der vi vet det skal være 1. For de resterende blir utgangen lik 0 og det er ikke like nødvendig å sette inn der funksjonen er 0.

a	b	c	F
0	0	0	1
0	0	1	
0	1	0	1
0	1	1	
1	0	0	
1	0	1	
1	1	0	1
1	1	1	1

2.3.5 Funksjonsuttrykkene for en 7-segment skjerm

Nå har vi lært å sette opp funksjonsuttrykk gjennom å lese ut fra en sannhetsverditabell. La oss ta utgangspunkt i sannhetsverditabellen til en 7-segment skjerm som vist i Tabell 2.4 på side 15 og skrive ut funksjonsuttrykkene for utgangene.

Desimal	T_3	T_2	T_1	T_0	S_a
0 ₁₀	0	0	0	0	1
1 ₁₀	0	0	0	1	0
2 ₁₀	0	0	1	0	1
3 ₁₀	0	0	1	1	1
4 ₁₀	0	1	0	0	0
5 ₁₀	0	1	0	1	1
6 ₁₀	0	1	1	0	1
7 ₁₀	0	1	1	1	1
8 ₁₀	1	0	0	0	1
9 ₁₀	1	0	0	1	1

Tabell 2.12: Sannhetsverditabellen for en S_a i en 7-segment skjerm.

Sannhetsverditabellen for S_a er satt opp i Tabell 2.12. Vi bryr oss kun om inngangskombinasjoner som gir oss desimaltallene 0 til 9 og har derfor forkortet sannhetsverditabellen når det gjelder antall rader. Vi leser ut fra tabellen:

$$F = 1 \text{ ved } \left\{ \begin{array}{ll} T_3 = T_2 = T_1 = T_0 = 0 & \\ T_3 = T_2 = T_0 = 0 & \text{og } T_1 = 1 \\ T_3 = T_2 = 0 & \text{og } T_1 = T_0 = 1 \\ T_3 = T_1 = 0 & \text{og } T_2 = T_0 = 1 \\ T_3 = T_0 = 0 & \text{og } T_2 = T_1 = 1 \\ T_3 = 0 & \text{og } T_2 = T_1 = T_0 = 1 \\ T_2 = T_1 = T_0 = 0 & \text{og } T_3 = 1 \\ T_2 = T_1 = 0 & \text{og } T_3 = T_0 = 1 \end{array} \right. \quad (2.29)$$

setter inn for variablene og får funksjonsuttrykket:

$$S_a = \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}\overline{T_2}T_1\overline{T_0} + \overline{T_3}\overline{T_2}T_1T_0 + \overline{T_3}T_2\overline{T_1}T_0 + \overline{T_3}T_2T_1\overline{T_0} \\ + \overline{T_3}T_2T_1T_0 + T_3\overline{T_2}\overline{T_1}\overline{T_0} + T_3\overline{T_2}\overline{T_1}T_0 \quad (2.30)$$

Tilsvarende for de resterende utgangene kan vi lese fra Tabell 2.4 på side 15 og får da følgende funksjonsuttrykk:

$$S_b = \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}\overline{T_2}\overline{T_1}T_0 + \overline{T_3}\overline{T_2}T_1\overline{T_0} + \overline{T_3}\overline{T_2}T_1T_0 + \overline{T_3}T_2\overline{T_1}\overline{T_0} \\ + \overline{T_3}T_2T_1T_0 + T_3\overline{T_2}\overline{T_1}\overline{T_0} + T_3\overline{T_2}\overline{T_1}T_0 \quad (2.31)$$

$$S_c = \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}\overline{T_2}T_1T_0 + \overline{T_3}T_2\overline{T_1}T_0 + \overline{T_3}T_2T_1\overline{T_0} + \overline{T_3}T_2T_1T_0 \\ + T_3\overline{T_2}\overline{T_1}\overline{T_0} + T_3\overline{T_2}\overline{T_1}T_0 \quad (2.32)$$

$$S_d = \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}\overline{T_2}T_1\overline{T_0} + \overline{T_3}\overline{T_2}T_1T_0 + \overline{T_3}T_2\overline{T_1}T_0 + \overline{T_3}T_2T_1\overline{T_0} \\ + T_3\overline{T_2}\overline{T_1}\overline{T_0} \quad (2.33)$$

$$S_e = \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}\overline{T_2}T_1\overline{T_0} + \overline{T_3}T_2T_1\overline{T_0} + T_3\overline{T_2}\overline{T_1}\overline{T_0} \quad (2.34)$$

$$S_f = \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}T_2\overline{T_1}T_0 + \overline{T_3}T_2T_1\overline{T_0} + T_3\overline{T_2}\overline{T_1}\overline{T_0} + T_3\overline{T_2}\overline{T_1}T_0 \quad (2.35)$$

$$S_g = \overline{T_3}\overline{T_2}T_1\overline{T_0} + \overline{T_3}\overline{T_2}T_1T_0 + \overline{T_3}T_2\overline{T_1}T_0 + \overline{T_3}T_2T_1\overline{T_0} + T_3\overline{T_2}\overline{T_1}\overline{T_0} \\ + T_3\overline{T_2}\overline{T_1}T_0 \quad (2.36)$$

Funksjonsuttrykkene for å få en 7-segment skjerm til å vise frem desimaltallene 0 til 9 ser ut til å kreve mange variabler og ledd, som igjen i sin tur vil lede til at vi trenger et stort areal (boks) for å implementere dette som en krets. Det er noen ledd som umiddelbart ser ut til å kunne reduseres, men hvordan er det med regneregler og forenkling. Dette ser vi nærmere på i Avsnitt 2.4.

2.4 Forenkling av funksjonsuttrykk med algebra

De to største motivasjonene for å forenkle et funksjonsuttrykk ligger i det å kunne designe kretser som tar mindre plass og reduserer kompleksiteten. Det er her teorien og den formelle beskrivelsen av logikk gjennom boolsk algebra gir oss muligheter til å kunne oppnå dette. I de neste avsnittene vil vi gå igjennom et forenklet sett av regneregler som er hentet ut fra ulike teoremer og postulater, i hovedsak tilpasset vår fokus på kretsdesign. Vi vil bruke Tabell 2.11 på side 20 og Likning (2.25) på side 23 gjennomgående som eksempler.

2.4.1 Regneregler – forenklet oversikt

I Tabell 2.13, er det listet opp en forenklet oversikt over de viktigste og nødvendigste regnereglene for å kunne forenkle funksjonsuttrykk. Vi skal ikke gå igjennom bevis for disse regnereglene, men heller anvende dem. Vi har valgt her å bruke små bokstaver (a,b,c) som representerer våre inngangssignaler. Mange av regnereglene kjenner man igjen fra vanlig algebra. Vi kan bruke disse

$a \cdot 1 = a$	$a + 0 = a$	(R1): Boundedness
$a \cdot 0 = 0$	$a + 1 = 1$	(R2): Boundedness
$a \cdot a = a$	$a + a = a$	(R3): Idempotency
$a \cdot \bar{a} = 0$	$a + \bar{a} = 1$	(R4): Completeness
$\bar{\bar{a}} = a$		(R5): Involution
----- $a \cdot b = \bar{b} \cdot a$	----- $a + b = b + a$	(R6): Commutative
$(a \cdot b) \cdot c = a \cdot (b \cdot c)$	$(a + b) + c = a + (b + c)$	(R7): Associative
$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$	$a + (b \cdot c) = (a + b) \cdot (a + c)$	(R8): Distributive
$\overline{a + b} = \bar{a} \cdot \bar{b}$	$\overline{a \cdot b} = \bar{a} + \bar{b}$	(R9): DeMorgans

Tabell 2.13: En oversikt over regneregler som er hentet ut fra boolsk algebra teori.

regnereglene til å forenkle funksjonsuttrykk som vi allerede i tidligere avsnitt har gått i gjennom. Det er ofte enklest å forenkle når vi har flere ledd i et funksjonsuttrykk og hvor hvert ledd inneholder alle variablene. Ser vi tilbake til Likning (2.25) på side 23 er de tre første funksjonene ikke mulig å forenkle mer, fordi de inneholder kun ett ledd. Ser vi på den fjerde funksjonsuttrykket kan dette forenkles til a , ene måten er å se det rett fra sannhetsverditabellen⁸, mens den andre er å bruke regnereglene til å komme frem til samme svar. Vi kan sette opp funksjonsuttrykket og forenklingsprosessen slik steg for steg:

$$\begin{aligned}
 F_4 &= [a\bar{b} + ab] \\
 &\text{braker regneregler R8 for å faktorisere ut } a: \\
 F_4 &= a \cdot (\bar{b} + b) \\
 &\text{ved å bruke regneregler R4 som gir } (\bar{b} + b) = 1, \text{ får vi:} \\
 F_4 &= a \cdot (1) \\
 &\text{ved å bruke regneregler R1 som gir } (a \cdot 1) = a, \text{ får vi:} \\
 F_4 &= a
 \end{aligned}
 \tag{2.37}$$

Vi har oppnådd $F_4 = a$ gjennom å ha forenklet uttrykket og kan påstå at den nye forenklete uttrykket bruker særdeles liten plass. I Kapittel 3 på side 53 vil vi presentere hvordan vi kan implementere kretser og der vil det være tydelig hvordan arealforbruket for disse funksjonsuttrykkene endrer seg som følge av en forenklingsprosess. Vi kan bruke regnereglene til å forenkle uttrykk i F_{15} fra Likning (2.24) på side 22 for å se om uttrykket blir slik som en NAND. Vi bruker da følgenede

⁸Etter mye trening og øvelse klarer man å se mønstre og sammenhenger i sannhetstabellen.

steg med tilhørende regneregler:

$$\begin{array}{ll}
 F_{15} = \bar{a}\bar{b} + \bar{a}b + a\bar{b} & \text{ser etter isolering av variable, velger } \bar{a} \\
 F_{15} = \bar{a} \cdot (\bar{b} + b) + a\bar{b} & \text{R8: } (\bar{a}\bar{b} + \bar{a}b) = \bar{a} \cdot (\bar{b} + b) \\
 F_{15} = \bar{a} \cdot (1) + a\bar{b} & \text{R4: } (b + \bar{b}) = 1 \\
 F_{15} = \bar{a} + a\bar{b} & \text{R1: } (a \cdot 1) = a \\
 F_{15} = (\bar{a} + a) \cdot (\bar{a} + \bar{b}) & \text{R8: } \bar{a} + (a \cdot \bar{b}) = (\bar{a} + a) \cdot (\bar{a} + \bar{b}) \\
 F_{15} = (1) \cdot (\bar{a} + \bar{b}) & \text{R4: } (a + \bar{a}) = 1 \\
 F_{15} = (\bar{a} + \bar{b}) & \text{R1: } (a \cdot 1) = a
 \end{array} \tag{2.38}$$

Dermed kan vi skrive $F_{15} = (\bar{a} + \bar{b})$, men dette er ser ikke ut som funksjonsuttrykket for en NAND som skrives på måten $(\bar{a}\bar{b})$. Disse to er faktisk like, altså er:

$$F_{15} = (\bar{a} + \bar{b}) = (\bar{a}\bar{b}) \tag{2.39}$$

Denne «konverteringen» er det matematikeren DeMorgan[?] som har bevist og fått et teorem oppkalt etter seg. DeMorgans teorem blir presentert i Avsnitt 2.4.2.

Eksempel 2.8 Forenkel følgende funksjonsuttrykk $F = \bar{a}b + a\bar{b} + ab$.

Vi bruker da følgende steg med tilhørende regneregler:

$$\begin{array}{ll}
 F = \bar{a}b + a\bar{b} + ab & \text{ser etter isolering av variable} \\
 F = \bar{a}b + a \cdot (\bar{b} + b) & \text{R8: } (ab + ac) = a \cdot (b + c) \\
 F = \bar{a}b + a \cdot (1) & \text{R4: } (a + \bar{a}) = 1 \\
 F = \bar{a}b + a & \text{R1: } (a \cdot 1) = a \\
 F = (a + \bar{a}) \cdot (a + b) & \text{R8: } a + (b \cdot c) = (a + b) \cdot (a + c) \\
 F = (a + b) & \text{R4: } (a + \bar{a}) = 1 \\
 F = a + b &
 \end{array} \tag{2.40}$$

Legg merke til at denne funksjonen er den samme som F_8 som i Tabell 2.11 på side 20 er satt opp som funksjon OR.

2.4.2 DeMorgans Teorem

Det å forenkle funksjonsuttrykk er ofte en kreativ prosess hvor man prøver og feiler en del før man oppnår et godt resultat. Selvom man kan alle regnereglene så er det ikke gitt at det er bare å anvende dem. Noen ganger må man utvide et funksjonsuttrykk eller se på det inverterte for å kunne forenkle det mer. Et av de mest velkjente og mest brukte teknikkene for å kunne implementere kretser er *DeMorgans Teorem*. Teoremet går i hovedsak ut på å beskrive inverterende funksjoner⁹ og som huskeregel kan du ha at hver gang du skal løse opp en inverterende funksjon så inverteres

⁹DeMorgans Teorem er veldig nyttig når man designer kretser, spesielt når man bruker NAND som er en inverterende funksjon.

den logiske funksjonen ('+' blir til '·') og hver variabel blir invertert. Som funksjon er teoremet som følger:

$$\begin{aligned}\overline{a + b} &= \bar{a} \cdot \bar{b} \\ \overline{ab} &= \bar{a} + \bar{b}\end{aligned}\tag{2.41}$$

Disse to uttrykkene henger tett sammen med kretsdesignene for NAND og NOR og relasjon til AND og OR. Spesielt kraftfullt er DeMorgans Teorem når det blir anvendt sammen med regneregler R5 med å dobbeltinvertere på begge sider av likhetstegnet.

Generelt:
Gjerne flere og enklere eksempler!

Eksempel 2.9 Vis at $Y = \overline{\overline{Y}}$ ved å bruke DeMorgans Teorem.

La oss ta utgangspunkt en tilfeldig funksjon som $Y = \bar{a}\bar{b}c + \bar{a}b$. Dette vil da gi følgende steg for å vise at $Y = \overline{\overline{Y}}$:

$Y = \overline{\overline{\bar{a}\bar{b}c + \bar{a}b}}$	setter inn for Y
$\overline{\overline{Y}} = \overline{(\bar{a} + b + \bar{c}) \cdot (a + \bar{b})}$	bruker DeMorgans for den ytterste inverteringen
$\overline{\overline{Y}} = (a \cdot \bar{b} \cdot c) + (\bar{a} \cdot b)$	bruker DeMorgans igjen
$\overline{\overline{Y}} = \bar{a}\bar{b}c + \bar{a}b$	som er lik utgangspunktet Y

(2.42)

2.4.3 Rekkefølge og når er nok nok?

I tillegg til regnereglene så er det også en rekkefølge for hvordan man forenkler uttrykk. Her er boolsk algebra meget likt vanlig algebra. Generelt er det anbefalt følgende rekkefølge som NOT-AND-OR:

- steg 1 Løs opp invertering (NOT) som binder AND eller OR-funksjoner, ofte brukes DeMorgans Teorem.
- steg 2 Løs opp parenteser som binder AND og OR-funksjoner, likt som i vanlig algebra «multiplisere» (AND) inn variabel.
- steg 3 Trekk sammen det som kan trekkes sammen, likt som i vanlig algebra hvor man summerer sammen (OR).

Det er vanskelig å vite når man har forenklet nok uten å involvere matematiske terminologi, som literaler og implikanter, for mer informasjon om dette kan du se i Avsnitt 2.7 på side 49. Det viktigste er at funksjonen ikke har endret sin logiske funksjon, så det er alltid anbefalt å kontrollsjekke utgangspunktet og det forenklete uttrykket i en og samme sannhetsverditabell.

Eksempel 2.10 Forenkle følgende funksjon: $F = a(\bar{b} + \bar{a}c)$

Vi følger rekkefølgen som beskrevet tidligere, først med oppløsning av eventuelle invertering som inneholder en OR-funksjon.

$F = a \cdot (\overline{b + ac})$	Utgangspunktet	
$F = a \cdot (\overline{b} \cdot \overline{ac})$	steg 1 (NOT): DeMorgans for å løse opp inverteringen	
$F = a \cdot [\overline{b} \cdot (\overline{a} + \overline{c})]$	steg 1 (NOT): DeMorgans for å løse opp inverteringen	
$F = a \cdot [\overline{b}\overline{a} + \overline{b}\overline{c}]$	steg 2 (AND): løser opp for parenteser	
$F = a\overline{b}\overline{a} + a\overline{b}\overline{c}$	steg 2 (AND): løser opp for parenteser	(2.43)
$F = a\overline{a}\overline{b} + a\overline{b}\overline{c}$	Regneregul R7 for å skrive leddene i alfabetisk orden	
$F = 0 \cdot \overline{b} + a\overline{b}\overline{c}$	Regneregul R4 som gir $a \cdot \overline{a} = 0$	
$F = 0 + a\overline{b}\overline{c}$	Regneregul R4 som gir $a \cdot \overline{a} = 0$	
$F = a\overline{b}\overline{c}$	Regneregul R1 som gir $a + 0 = a$	

Dette eksemplet finnes også som video med gjennomgang av fremgangsmetode og regelbruk.

2.4.4 Forenklig av funksjonsuttrykk for en 7-segments skjerm

Vi kan bruke regnereglene til å forenkle funksjonsuttrykkene for en 7-segments skjerm som vi listet opp i Likning (2.30) til Likning (2.36) på side 26. Nedenfor følger hver av funksjonsuttrykkene med tilhørende forenklingsprosedyre og referanse til regnereglene:

$$S_a = \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}\overline{T_2}T_1\overline{T_0} + \overline{T_3}\overline{T_2}T_1T_0 + \overline{T_3}T_2\overline{T_1}T_0 + \overline{T_3}T_2T_1\overline{T_0} + \overline{T_3}T_2T_1T_0 + T_3\overline{T_2}\overline{T_1}\overline{T_0} + T_3\overline{T_2}\overline{T_1}T_0$$

faktoreriserer ut $\overline{T_3}T_1$ for de merkede leddene med regneregul R8 og får:

$$S_a = \overline{T_3}T_1 \left[\overline{T_2}\overline{T_0} + \overline{T_2}T_0 + T_2\overline{T_0} + T_2T_0 \right] + \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}\overline{T_2}\overline{T_1}T_0 + T_3\overline{T_2}\overline{T_1}\overline{T_0} + T_3\overline{T_2}\overline{T_1}T_0$$

faktoreriserer for $\overline{T_2}$ og T_2 innvendig i parantes for de merkede leddene og får:

$$S_a = \overline{T_3}T_1 \left[\overline{T_2}(\overline{T_0} + T_0) + T_2(\overline{T_0} + T_0) \right] + \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}\overline{T_2}\overline{T_1}T_0 + T_3\overline{T_2}\overline{T_1}\overline{T_0} + T_3\overline{T_2}\overline{T_1}T_0$$

regneregul R4 gir oss at $T_0 + \overline{T_0} = 1$ og regneregul R2 gir $\overline{T_2} \cdot 1 = \overline{T_2}$, samlet får vi:

$$S_a = \overline{T_3}T_1 \left[\overline{T_2} + T_2 \right] + \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}\overline{T_2}\overline{T_1}T_0 + T_3\overline{T_2}\overline{T_1}\overline{T_0} + T_3\overline{T_2}\overline{T_1}T_0$$

ved å bruke regneregul R4 i parantesen får vi:

$$S_a = \overline{T_3}T_1 + \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}\overline{T_2}\overline{T_1}T_0 + \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}\overline{T_2}\overline{T_1}T_0$$

faktorerer for $T_3\bar{T}_2\bar{T}_1$ og får:

$$S_a = \bar{T}_3T_1 + T_3\bar{T}_2\bar{T}_1[\bar{T}_0 + T_0] + \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3T_2\bar{T}_1T_0$$

Funksjonen forenklet med regneregler gir da:

$$S_a = \underline{\underline{\bar{T}_3T_1 + T_3\bar{T}_2\bar{T}_1 + \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3T_2\bar{T}_1T_0}} \quad (2.44)$$

Tilsvarende kan forenklingen for de resterende funksjonen også gjøres. Merk at det finnes flere måter å foreta forenklingene på og det kan også være flere like gode forenklete funksjonsuttrykk. Nedenfor følger et forslag for å forenkle funksjonsuttrykkene de resterende utgangene for en 7-segment skjerm.

$$S_b = \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3\bar{T}_2\bar{T}_1T_0 + \bar{T}_3\bar{T}_2T_1\bar{T}_0 + \bar{T}_3\bar{T}_2T_1T_0 + \bar{T}_3T_2\bar{T}_1\bar{T}_0 + \bar{T}_3T_2T_1T_0 \\ + T_3\bar{T}_2\bar{T}_1\bar{T}_0 + T_3\bar{T}_2\bar{T}_1T_0 \quad (2.45)$$

$$S_b = \bar{T}_3\bar{T}_2 + \bar{T}_3T_2\bar{T}_1\bar{T}_0 + \bar{T}_3T_2T_1T_0 + T_3\bar{T}_2\bar{T}_1$$

$$S_c = \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3\bar{T}_2T_1T_0 + \bar{T}_3T_2\bar{T}_1T_0 + \bar{T}_3T_2T_1\bar{T}_0 + \bar{T}_3T_2T_1T_0 + T_3\bar{T}_2\bar{T}_1\bar{T}_0 + T_3\bar{T}_2\bar{T}_1T_0 \quad (2.46)$$

$$S_c = T_3T_0 + \bar{T}_3T_2\bar{T}_0 + T_3\bar{T}_2\bar{T}_1 + \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0$$

$$S_d = \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3\bar{T}_2T_1\bar{T}_0 + \bar{T}_3\bar{T}_2T_1T_0 + \bar{T}_3T_2\bar{T}_1T_0 + \bar{T}_3T_2T_1\bar{T}_0 \\ + T_3\bar{T}_2\bar{T}_1\bar{T}_0 \quad (2.47)$$

$$S_d = \bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3\bar{T}_2T_1 + \bar{T}_3T_2T_1\bar{T}_0 + \bar{T}_3T_2\bar{T}_1T_0$$

$$S_e = \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3\bar{T}_2T_1\bar{T}_0 + \bar{T}_3T_2T_1\bar{T}_0 + T_3\bar{T}_2\bar{T}_1\bar{T}_0 \quad (2.48)$$

$$S_e = \bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3T_1\bar{T}_0$$

$$S_f = \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3\bar{T}_2\bar{T}_1T_0 + \bar{T}_3T_2T_1\bar{T}_0 + T_3\bar{T}_2\bar{T}_1\bar{T}_0 + T_3\bar{T}_2\bar{T}_1T_0 \quad (2.49)$$

$$S_f = \bar{T}_3T_2\bar{T}_1 + \bar{T}_3\bar{T}_2\bar{T}_1 + \bar{T}_3T_2T_1\bar{T}_0 + \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0$$

$$S_g = \bar{T}_3\bar{T}_2T_1\bar{T}_0 + \bar{T}_3\bar{T}_2T_1T_0 + \bar{T}_3T_2\bar{T}_1T_0 + \bar{T}_3T_2T_1\bar{T}_0 + T_3\bar{T}_2\bar{T}_1\bar{T}_0 + T_3\bar{T}_2\bar{T}_1T_0 \quad (2.50)$$

$$S_g = \bar{T}_3T_2\bar{T}_1 + \bar{T}_3\bar{T}_2\bar{T}_1 + \bar{T}_3T_2T_1\bar{T}_0 + \bar{T}_3\bar{T}_2T_1 + \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0$$

2.5 Karnaugh diagram

Forenkling kompliserte funksjonsuttrykk med å bruke regneregler er en tidskrevende øvelse og ikke minst er det stor risiko for å gjøre feil. I dette avsnittet presenterer vi en visuell måte å sette opp en matrise som gjør det lettere å forenkling uttrykk.

I 1881 presenterte Allan Marquand et diagram for logikk med n termer[4], men det var ikke før i 1952 at Edward Veitch publiserte sitt arbeid for bruk av en tilsvarende diagram for forenkling av logikk uttrykk[5]. Året etter, i 1953, presenterte Maurice Karnaugh en forbedret versjon av Veitch diagram med fokus om forenkling av boolske funksjoner innen kombinatorisk kretslogikk[6]. Det er derfor vi kaller dette diagrammet for Karnaugh diagram (engelsk K-map).

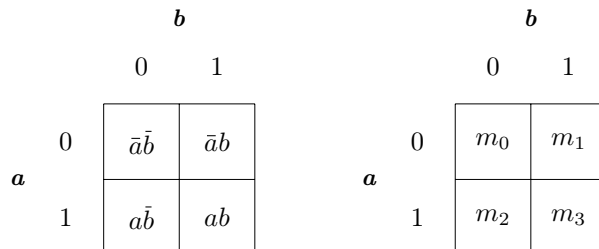
I de neste avsnittene introduserer vi 2-variabels, 3-variabels og 4-variabels Karnaugh diagrammer. Det er mulig å sette opp slike diagrammer for enda flere variabler, men det er ikke pensum for emnet.

2.5.1 2-variabels Karnaugh diagram

Utgangsfunksjonene for en krets med to innganger kan skrives i en sannhetsverditabell slik som i Tabell 2.14. Disse kalles for *minterm*¹⁰. Ved hjelp av Karnaugh diagram kan vi skrive opp sannhetsverditabellen som en matrise, slik som vist i Figur 2.4.

a	b	F	minterm
0	0	$\bar{a}\bar{b}$	m_0
0	1	$\bar{a}b$	m_1
1	0	$a\bar{b}$	m_2
1	1	ab	m_3

Tabell 2.14: *Utgangsfunksjonene for en 2-inngangs krets og korresponderende minterm.*

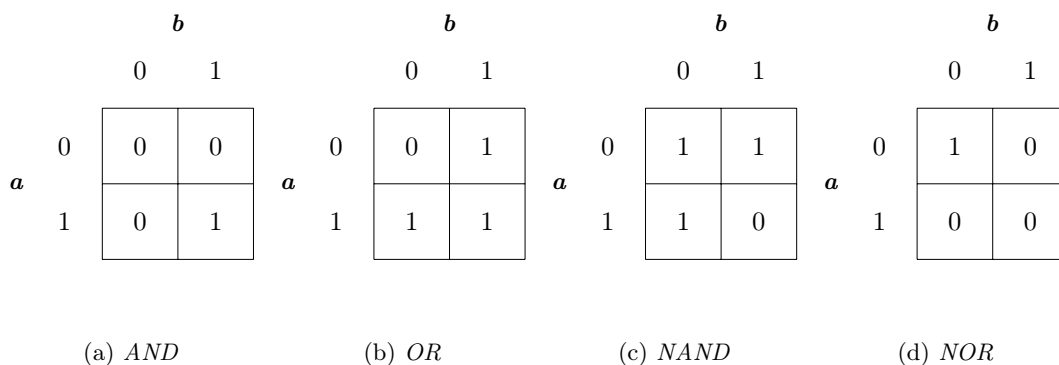


Figur 2.4: *En to variabel Karnaugh diagram.*

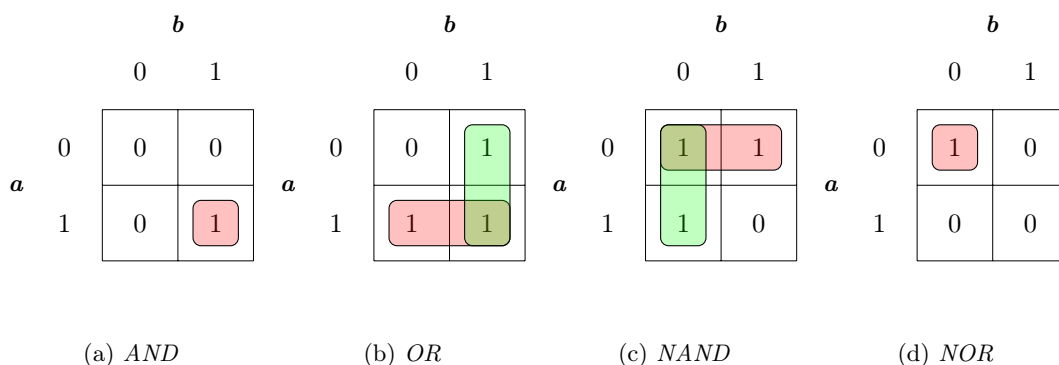
Radene representerer verdien til inngangssignalet a , mens kolonnene representerer inngangssignalet b . I hele første rad er inngangssignalet $a = 0$, som gir at utgangsfunksjonen får variabelen \bar{a} for begge kolonnene, mens andre rad er $a = 1$, som gir variabel a for begge kolonnene. Tilsvarende kan vi se for den første kolonnen hvor inngangssignalet $b = 0$, som gir \bar{b} for begge radene og andre kolonne som gir $b = 1$.

Utfyllingen av Karnaugh-diagrammet gjøres ved at man noterer hva utgangsfunksjonen er for de ulike kombinasjonene av inngangssignalene. Som et eksempel er Karnaugh-diagrammet for en AND,

¹⁰henvisning til Minterm er gitt i Avsnitt 2.7 på side 49.



Figur 2.5: Karnaugh-diagrammene for de ulike basis funksjonene.



Figur 2.6: Karnaugh-diagrammene for de ulike basis funksjonene med utlesing av funksjonsverdiene.

OR, NAND og NOR vist i Figur 2.5. Legg merke til at det er den samme informasjonen som i sannhetsverditabellen som skrives inn i Karnaugh diagrammet.

Forenklingen av et uttrykk ved å bruke Karnaugh diagram går ut på å kombinere flest mulig ruter med samme verdi etter gitte regler. Antall ruter man kan kombinere er 2^n , altså 1,2,4,8,16 osv. I tillegg kan kun rutene horisontal eller vertikal rekkefølge/nabo kombineres. Det er også mulig å gjenbruke ruter for å kombinere flere uttrykk. Hvert sett av ruter blir ett ledd i det endelige funksjonsuttrykket. Utlesningen for AND, OR, NAND og NOR er vist i Figur 2.6.

Funksjonsuttrykkene som lest ut fra Karnaugh diagrammet kan skrives på følgende måte:

$$\begin{aligned}
 F_{AND} &= ab \\
 F_{OR} &= a + b \\
 F_{NAND} &= \bar{a} + \bar{b} \\
 F_{NOR} &= \bar{a}\bar{b}
 \end{aligned}
 \tag{2.51}$$

Eksempel 2.11 Bruk Karnaugh diagram for F_{12} fra Tabell 2.11 på side 20

Funksjonsuttrykket $F_{12} = \bar{a}\bar{b} + a\bar{b} + ab$ og vi kan sette inn i en Karnaugh diagram som følgende:

		b	
		0	1
a	0	1	0
	1	1	1

Funksjonsuttrykket F_{12} kan leses ut som følger:

$$F_{12} = \bar{a} + \bar{b}$$

Sammenligner vi dette svaret med det som er oppgitt i Likning (2.25) på side 23, ser vi hvor vesentlig uttrykket er blitt forenklet. Vi har gått fra å oppgi $F_{12} = \bar{a}\bar{b} + a\bar{b} + ab$ til å kunne uttrykkes som $F_{12} = \bar{b} + a$.

2.5.2 3-variabels Karnaugh diagram

Vi kan sette sammen to 2-variabels Karnaugh diagram sammen for å lage en 3-variabels Karnaugh diagram. For å kunne få til en mest mulig effektiv forenkling må matrisen for en 3-variabels Karnaugh diagram konstrueres slik at det bare er en variabel som endres mellom hver rute. For å få til dette brukes *gray-code* i Karnaugh diagrammet for indeksering. I Figur 2.7 er en 3-variabels Karnaugh diagram vist med tilhørende mintermer. I Figur 2.8 på neste side har vi markert hvor

		bc						bc			
		00	01	11	10			00	01	11	10
a	0	$\bar{a}\bar{b}\bar{c}$	$\bar{a}\bar{b}c$	$\bar{a}bc$	$\bar{a}b\bar{c}$	a	0	m_0	m_1	m_3	m_2
	1	$a\bar{b}\bar{c}$	$a\bar{b}c$	abc	$ab\bar{c}$		1	m_4	m_5	m_7	m_6

Figur 2.7: En 3-variabels Karnaugh diagram

hver av de enkelte variablene er «sann».

		<i>bc</i>			
		00	01	11	10
<i>a</i>	0	$\bar{a}\bar{b}\bar{c}$	$\bar{a}\bar{b}c$	$\bar{a}b\bar{c}$	$\bar{a}bc$
	1	$a\bar{b}\bar{c}$	$a\bar{b}c$	abc	$ab\bar{c}$

(a) Markering av $a = 1$

		<i>bc</i>			
		00	01	11	10
<i>a</i>	0	$\bar{a}\bar{b}\bar{c}$	$\bar{a}\bar{b}c$	$\bar{a}bc$	$\bar{a}b\bar{c}$
	1	$a\bar{b}\bar{c}$	$a\bar{b}c$	abc	$ab\bar{c}$

(b) Markering av $b = 1$

		<i>bc</i>			
		00	01	11	10
<i>a</i>	0	$\bar{a}\bar{b}\bar{c}$	$\bar{a}\bar{b}c$	$\bar{a}bc$	$\bar{a}b\bar{c}$
	1	$a\bar{b}\bar{c}$	$a\bar{b}c$	abc	$ab\bar{c}$

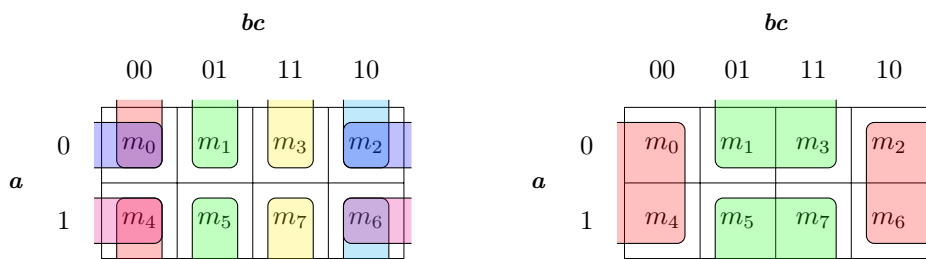
(c) Markering av $c = 1$

		<i>bc</i>			
		00	01	11	10
<i>a</i>	0	$\bar{a}\bar{b}\bar{c}$	$\bar{a}\bar{b}c$	$\bar{a}bc$	$\bar{a}b\bar{c}$
	1	$a\bar{b}\bar{c}$	$a\bar{b}c$	abc	$ab\bar{c}$

(d) Markering av $a = b = c = 1$

Figur 2.8: En 3-variabels Karnaugh diagram hvor «sanne» variabler er fargelagt med lik farge

Når det gjelder utlesningen av en 3-variabel Karnaugh diagram kan vi bruke samme fremgangsmetode som for en 2-variabels utlesning men med en litt vesentlig ny vri. Alle Karnaugh diagrammer er sirkulære, det betyr at hjørnerutene i hver sin ende er naboruter. Sett med Figur 2.7 på side 34 er m_0 naborute med $[m_4$ og $m_2]$ og tilsvarende for alle hjørnerutene. I Figur 2.9 er et utvalg av mulige kombinasjoner av hjørnene for en 3-variabels Karnaugh diagram vist.



Figur 2.9: Mulige hjørnekombinasjoner for en 3-variabels Karnaugh diagram

Eksempel 2.12 Bruk Karnaugh diagram for å forenkle funksjonen gitt i Eksempel 2.6 på side 23

Vi skriver opp sannhetsverditabellen med tilhørende mintermer:

a	b	c	F	minterm
0	0	0	0	m_0
0	0	1	1	m_1
0	1	0	0	m_2
0	1	1	1	m_3
1	0	0	1	m_4
1	0	1	1	m_5
1	1	0	0	m_6
1	1	1	0	m_7

Vi ser av tabellen at funksjonen er 1 for mintermene: $[m_1, m_3, m_4, m_5]$ som vil gi oss følgende 3-variabels Karnaugh diagram:

		<i>bc</i>			
		00	01	11	10
<i>a</i>	0		1	1	
	1	1	1		

For å gjøre det lettere å lese ut skriver vi bare opp der funksjonsuttrykket er 1 og kan lese ut følgende:

		<i>bc</i>			
		00	01	11	10
<i>a</i>	0		1	1	
	1	1	1		

Som gir da funksjonsuttrykket:

$$F = \bar{a}c + a\bar{b}$$

(Det er mulig å lese ut på andre måter også, men foreløpig er denne den mest effektive måten å gjøre det på.)

	<i>cd</i>			
	00	01	11	10
00	$\bar{a}\bar{b}\bar{c}\bar{d}$	$\bar{a}\bar{b}cd$	$\bar{a}b\bar{c}\bar{d}$	$\bar{a}bcd$
01	$\bar{a}b\bar{c}\bar{d}$	$\bar{a}bcd$	$a\bar{b}\bar{c}\bar{d}$	$a\bar{b}cd$
11	$ab\bar{c}\bar{d}$	$ab\bar{c}d$	$abcd$	$abc\bar{d}$
10	$a\bar{b}\bar{c}\bar{d}$	$a\bar{b}cd$	$ab\bar{c}\bar{d}$	$abcd$

	<i>bc</i>			
	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Figur 2.10: En 4-variabels Karnaugh diagram.

2.5.3 4-variabels Karnaugh diagram

Ved å sette sammen to 3-variabels Karnaugh diagrammer kan vi lage en 4-variabels Karnaugh diagram, se Figur 2.10. For å beskrive én rute trenger vi fire variabler, hvor hver av variablene er «sanne» er vist i Figur 2.11 på neste side. Tilsvarende som for en 3-variabels Karnaugh diagram har vi også her at alle hjørnene er naboruter. Det er slik at vi kan gruppere sammen alle fire hjørne ruter til én, altså slå sammen m_0 , m_2 , m_8 og m_{10} som gir oss uttrykket $\bar{b}\bar{d}$, se forøvrig Figur 2.12 på side 40

Det er en direkte relasjon mellom hvor mange ruter som er kombinert og hvor mange variabler det trengs for å beskrive området. Relasjonen kan skrives som følgende:

- 1 rute kombinert krever *alle variablene* for å beskrive den
- 2 ruter kombinert krever *alle variablene minus en* for å beskrive den
- 4 ruter kombinert krever *alle variablene minus to* for å beskrive den
- 8 ruter kombinert krever *alle variablene minus tre* for å beskrive den
- 16 ruter kombinert krever *alle variablene minus fire* for å beskrive den

		<i>cd</i>			
		00	01	11	10
<i>ab</i>	00	m_0	m_1	m_3	m_2
	01	m_4	m_5	m_7	m_6
	11	m_{12}	m_{13}	m_{15}	m_{14}
	10	m_8	m_9	m_{11}	m_{10}

(a) Markering av $a = 1$.

		<i>cd</i>			
		00	01	11	10
<i>ab</i>	00	m_0	m_1	m_3	m_2
	01	m_4	m_5	m_7	m_6
	11	m_{12}	m_{13}	m_{15}	m_{14}
	10	m_8	m_9	m_{11}	m_{10}

(b) Markering av $b = 1$.

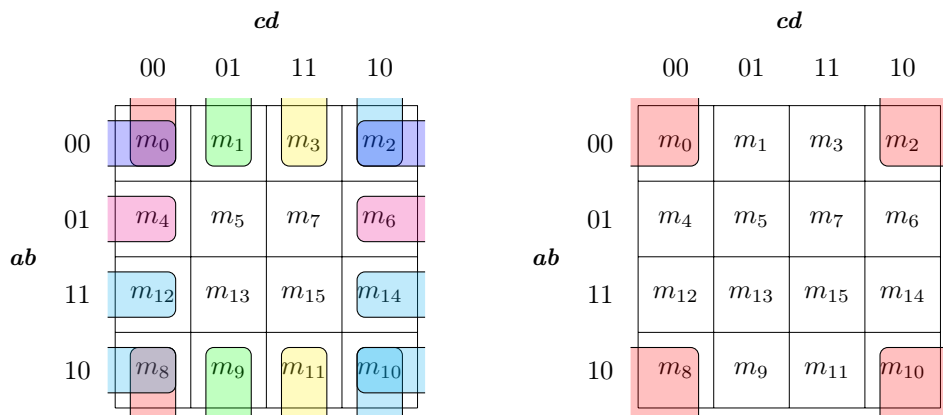
		<i>cd</i>			
		00	01	11	10
<i>ab</i>	00	m_0	m_1	m_3	m_2
	01	m_4	m_5	m_7	m_6
	11	m_{12}	m_{13}	m_{15}	m_{14}
	10	m_8	m_9	m_{11}	m_{10}

(c) Markering av $c = 1$.

		<i>cd</i>			
		00	01	11	10
<i>ab</i>	00	m_0	m_1	m_3	m_2
	01	m_4	m_5	m_7	m_6
	11	m_{12}	m_{13}	m_{15}	m_{14}
	10	m_8	m_9	m_{11}	m_{10}

(d) Markering av $d = 1$.

Figur 2.11: 4-variablers Karnaugh diagrammer som illustrerer hvor de representative variablene er 1.



Figur 2.12: Mulige hjørnekombinasjoner for en 4-variabels Karnaugh diagram

		<i>cd</i>			
		00	01	11	10
<i>ab</i>	00		1		1
	01	1	1	1	
	11		1	1	1
	10	1		1	

Fra Karnaugh diagrammet ser vi at vi kan bruke en XOR funksjon i kombinasjon med å kombinere de fire midterste 1'erne. Utlesningen av Karnaugh diagrammet kan derfor gjøre som følgende:

Får ikke frem Kmap i eksempel

2.5.5 Forenkling av funksjonsuttrykkene for en 7-segment skjerm med Karnaugh Diagram

Vi kan bruke en 4-variabel Karnaugh diagram for å forenkle funksjonsuttrykkene for vår 7-segment skjerm. Alle utgangsfunksjonene og deres representative Karnaugh diagram med tilhørende funksjonsutlesninger vist i Figur 2.15 på side 44 og Figur 2.16 på side 45. Noen av funksjonsuttrykkene har ledd som er gjenkjennbare fra forenklingene med regneregler. I Tabell 2.15 på neste side har vi listet opp funksjonsuttrykkene oppnådd gjennom forenkling med regning og

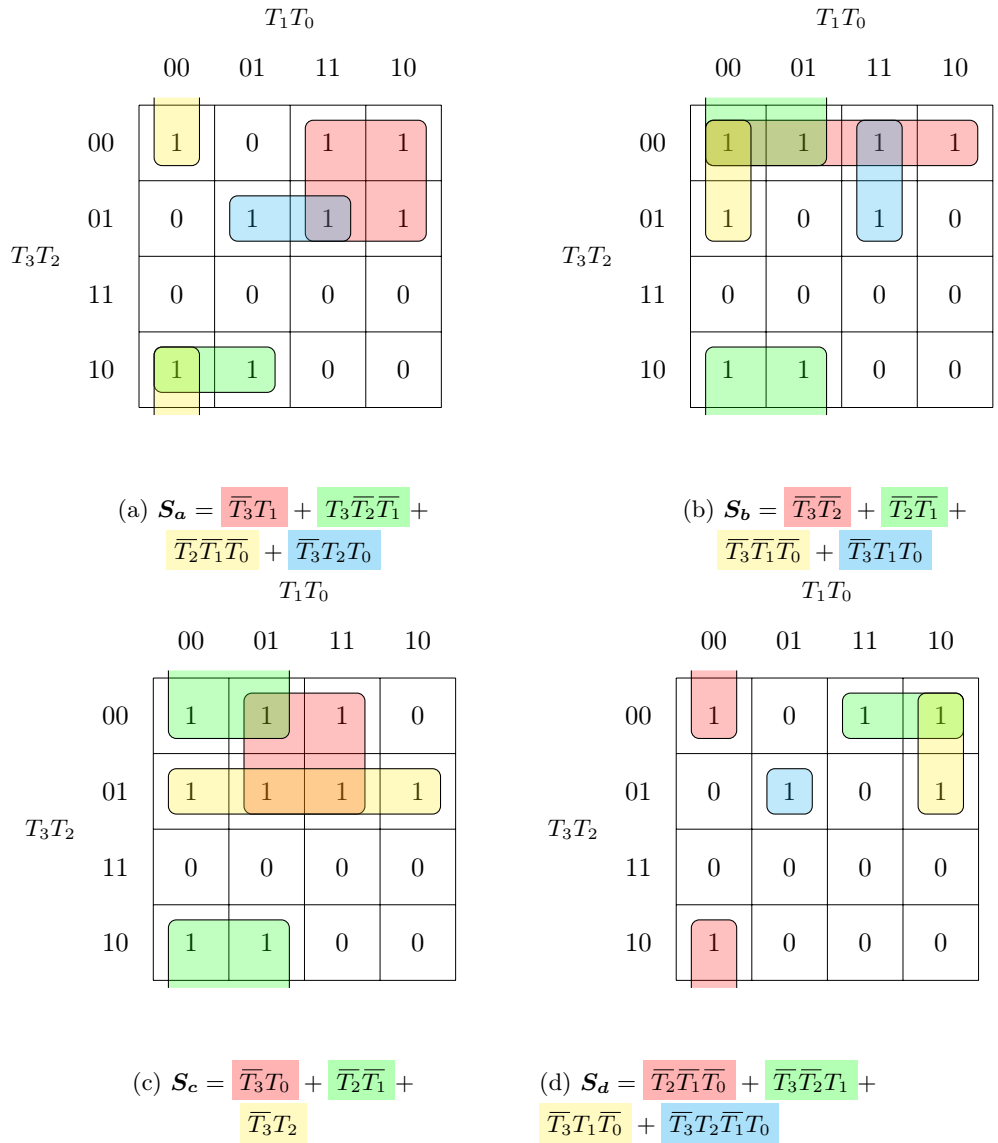
		<i>cd</i>			
		00	01	11	10
<i>ab</i>	00		1		1
	01	1	1	1	
	11		1	1	1
	10	1		1	

Figur 2.14: til bruk i eksempel eksempel 2.13 på side 41

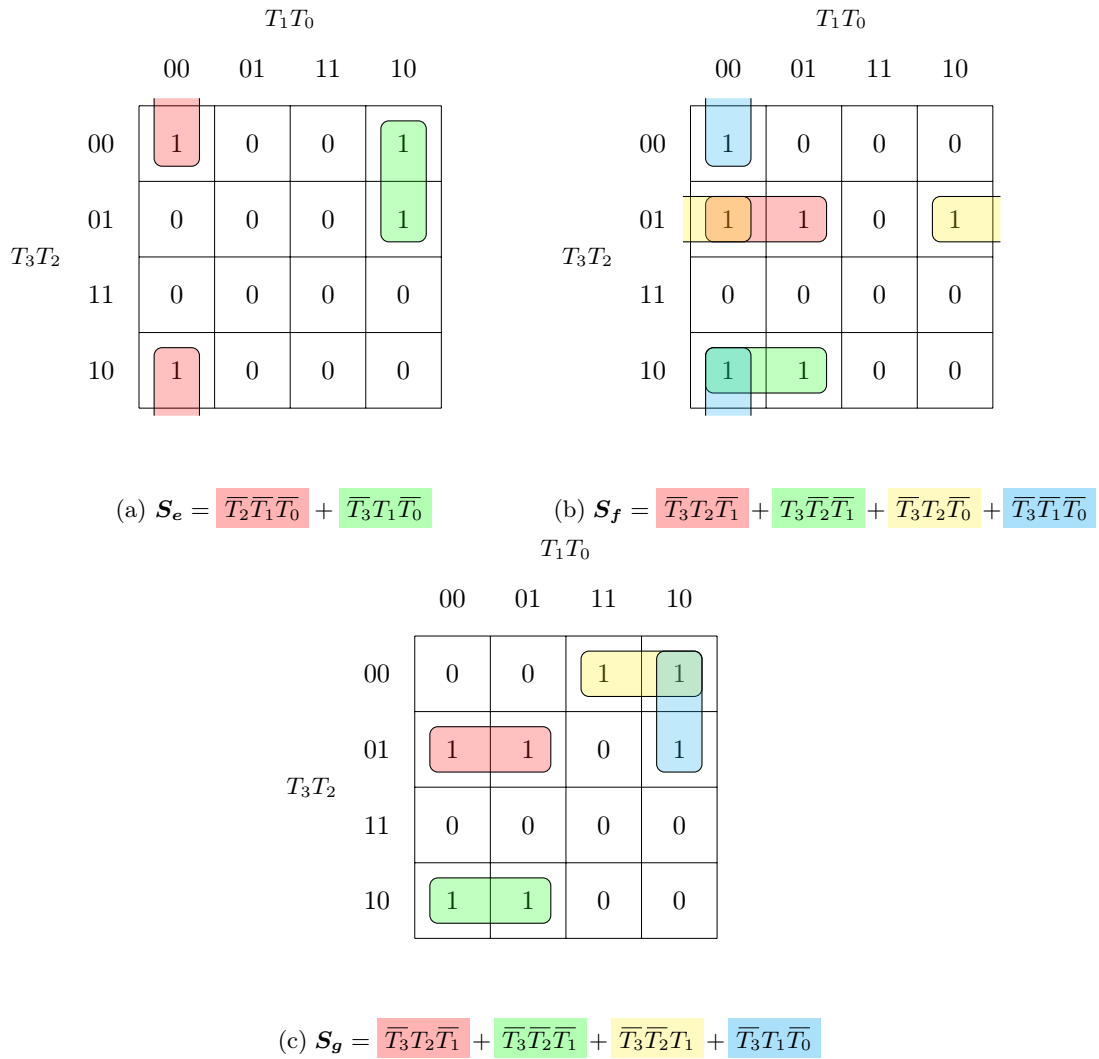
forenkling med Karnaugh diagram.

	Regneregler	Karnaugh diagram
S_a	$\bar{T}_3T_1 + T_3\bar{T}_2\bar{T}_1 + \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3T_2\bar{T}_1T_0$	$\bar{T}_3T_1 + T_3\bar{T}_2\bar{T}_1 + \bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3T_2T_0$
S_b	$\bar{T}_3\bar{T}_2 + \bar{T}_3T_2\bar{T}_1\bar{T}_0 + \bar{T}_3T_2T_1T_0 + T_3\bar{T}_2\bar{T}_1$	$\bar{T}_3\bar{T}_2 + \bar{T}_2\bar{T}_1 + \bar{T}_3\bar{T}_1\bar{T}_0 + \bar{T}_3T_1T_0$
S_c	$T_3T_0 + \bar{T}_3T_2\bar{T}_0 + T_3\bar{T}_2\bar{T}_1 + \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0$	$\bar{T}_3T_0 + \bar{T}_2\bar{T}_1 + \bar{T}_3T_2$
S_d	$\bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3\bar{T}_2T_1 + \bar{T}_3T_2T_1\bar{T}_0 + \bar{T}_3T_2\bar{T}_1T_0$	$\bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3\bar{T}_2T_1 + \bar{T}_3T_1\bar{T}_0 + \bar{T}_3T_2\bar{T}_1T_0$
S_e	$\bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3T_1\bar{T}_0$	$\bar{T}_2\bar{T}_1\bar{T}_0 + \bar{T}_3T_1\bar{T}_0$
S_f	$\bar{T}_3T_2\bar{T}_1 + \bar{T}_3\bar{T}_2\bar{T}_1 + \bar{T}_3T_2T_1\bar{T}_0 + \bar{T}_3\bar{T}_2T_1\bar{T}_0$	$\bar{T}_3T_2\bar{T}_1 + T_3\bar{T}_2\bar{T}_1 + \bar{T}_3T_2\bar{T}_0 + \bar{T}_3\bar{T}_1\bar{T}_0$
S_g	$\bar{T}_3T_2\bar{T}_1 + \bar{T}_3\bar{T}_2\bar{T}_1 + \bar{T}_3T_2T_1\bar{T}_0 + \bar{T}_3\bar{T}_2T_1 + \bar{T}_3\bar{T}_2\bar{T}_1\bar{T}_0$	$\bar{T}_3T_2\bar{T}_1 + \bar{T}_3\bar{T}_2\bar{T}_1 + \bar{T}_3\bar{T}_2T_1 + \bar{T}_3\bar{T}_1\bar{T}_0$

Tabell 2.15: Oversikt over de forenklete uttrykkene for en 7-segment skjerm ved å bruke regneregler og Karnaugh diagram.



Figur 2.15: De fire første Karnaugh diagrammene for en 7-segments skjerm.



Figur 2.16: De tre siste Karnaugh diagrammene for en 7-segments skjerm.

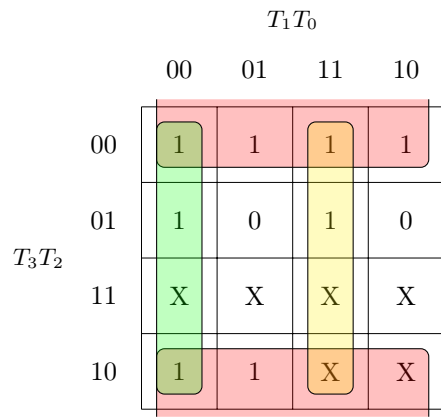
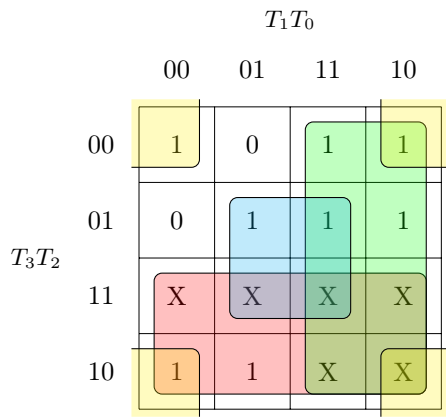
2.6 Don't Care - verdier

I enkelte funksjoner og kretser kan det være at man har noen inngangssignaler som man egentlig ikke trenger eller bryr seg om, disse kaller man for *Don't Care*. Ofte er *Don't Care* skrevet D , X , $?$ og kan brukes både for inngang og utgangssignaler. Vi kommer til å bruke symbolet X for don't care. Fordelen med å ha en don't care signal er at man kan betrakte den som en 0 eller 1 avhengig av hva som gir det mest forenklete uttrykket. Som eksempel kan vi se på vår 7-segment skjerm, hvor vi trenger minst 4-bit for å representere desimaltallene 0-9, men hva gjør vi da med de overskytende tallene vi kan representere (altså 10-15)? En mulighet er å si at vi ikke bryr oss om hva utgangen blir når det kommer inn inngangssignaler tilsvarende desimaltall 10-15 og kan derfor defineres utgangene som X . Vi har satt inn for don't care for inngangene m_{10} til m_{15} i en revidert sannhetsverditabell vist i Tabell 2.16. Ved å betrakte en del utganger som don't care vil som oftest gi et mer forenklet uttrykk.

T_3	T_2	T_1	T_0	S_a	S_b	S_c	S_d	S_e	S_f	S_g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X

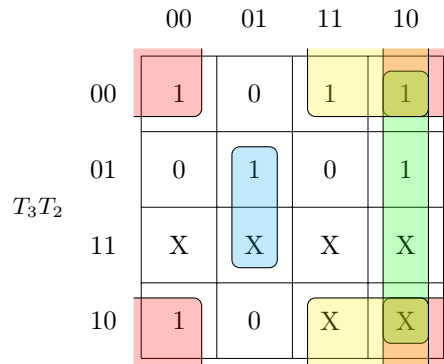
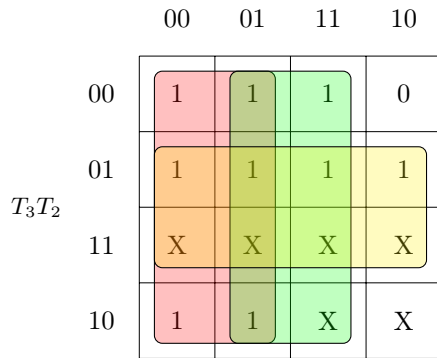
Tabell 2.16: Revidert tabell for en 7-segments display med bruk av don't care verdier.

I Figur 2.17 på neste side er de fire første, S_a, S_b, S_c, S_d , funksjonsuttrykkene for lysdiodene vist i et Karnaugh diagram med korresponderende funksjonsuttrykk listet opp under hvert av diagramene. Måten å lese ut fra et Karnaugh diagram som inneholder don't care verdier at at vi anser en X til å være enten en 0 eller en 1 avhengig av hva vi har behov for i den aktuelle ruten. Det betyr at hver rute med en X kan tolkes individuelt som en 0 eller en 1.



(a) $S_a = \overline{T_3} + T_1 + \overline{T_2}\overline{T_0} + T_2T_0$

(b) $S_b = \overline{T_2} + \overline{T_1}\overline{T_0} + T_1T_0$



(c) $S_c = \overline{T_1} + T_0 + T_2$

(d) $S_d = \overline{T_2}\overline{T_0} + T_1\overline{T_0} + \overline{T_2}T_1 + T_2\overline{T_1}T_0$

Figur 2.17: De fire første Karnaugh diagrammene for en 7-segments skjerm med don't care verdier.

		T_1T_0			
		00	01	11	10
T_3T_2	00	1	0	0	1
	01	0	0	0	1
	11	X	X	X	X
	10	1	0	X	X

		T_1T_0			
		00	01	11	10
T_3T_2	00	1	0	0	0
	01	1	1	0	1
	11	X	X	X	X
	10	1	1	X	X

(a) $S_e = \overline{T_2}\overline{T_0} + T_1\overline{T_0}$

(b) $S_f = T_2\overline{T_1} + \overline{T_1}\overline{T_0} + T_2\overline{T_0}$

		T_1T_0			
		00	01	11	10
T_3T_2	00	0	0	1	1
	01	1	1	0	1
	11	X	X	X	X
	10	1	1	X	X

(c) $S_g = \overline{T_3}T_2\overline{T_1} + \overline{T_3}\overline{T_2}\overline{T_1} + \overline{T_3}\overline{T_2}T_1 + \overline{T_3}T_1\overline{T_0}$

Figur 2.18: De tre siste Karnaugh diagrammene for en 7-segments skjerm med don't care verdier.

2.7 Videre henvisning

Vi kan vise til ulike mer avanserte tanker og ideer. F.eks. at man kan sette opp en sannhetsverdi tabell hvor rekkefølgen på inngangene ikke har noe å si.

- Transistorer og strøm
- Fan-in og fan-out
- Minterm og Maxterm - SOP og POS
- Bevis av postulatene
- Prime implicants
- Boolsk algebra og mengdelæring
- Om DeMorgan (person)
- Om Boole (person)
- Bubble push
- Omgjøring til kretser med bare NAND-porter.
- Logikk porter uten strøm, men med vann: http://www.blikstein.com/paulo/projects/project_water.html

Hacking med XOR: message i binær representasjon og xor med en nøkkel

<i>klartekst melding</i>	M	I	D	D	A	G
binærform (b)	01101	01001	00100	00100	00001	00111
nøkkel (n)	00001	01011	00101	00001	01011	00101
$b \oplus n$:	01100	00010	00001	00101	01010	00010
<i>kryptert melding:</i>	L	B	A	E	J	B

Tabell 2.17: *Kryptering av en melding «MIDDAG» ved å bruke krypteringsnøkkel n gir oss «LBAEJB».*

Kan du gjette hva nøkkelen er?

2.8 Oppgaver

2.8.1 Sannhetstabell

Hvor mange kombinasjoner av inngang er det for en 4-inngangs port?

Hvor mange kombinasjoner av inngang er det for en 10-inngangs port?

Hvor mange kombinasjoner av inngang er det for en 128-inngangs port?

Vis sannhetstabellen for en 4-inngangs AND.

Vis sannhetstabellen for en 4-inngangs OR.

Vis sannhetstabellen for en 4-inngangs NAND.

Vis sannhetstabellen for en 4-inngangs XOR.

2.8.2 Funksjonsuttrykk

Bruk samme fremgangsmetode som i seksjon Avsnitt [2.3.4](#) på side [19](#) for alle funksjonene i tabell [Tabell 2.11](#) på side [20](#).

2.8.3 Forenkling av uttrykk med regneregler

test

$$F = xy + xy'$$

$$F = (x+y)(x+y')$$

$$F = xyz + x'y + xyz'$$

$$F = (A+B)'(A'+B)'$$

$$F = ABC + A'B + ABC'$$

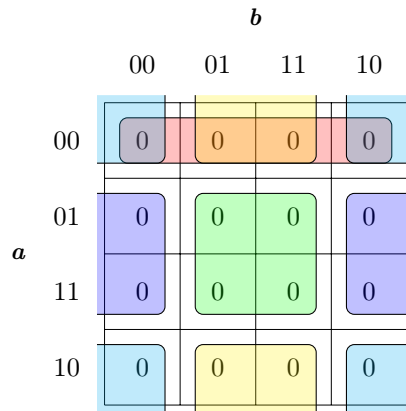
$$F = x'yz + xz$$

$$F = (x+y)'(x'+y')$$

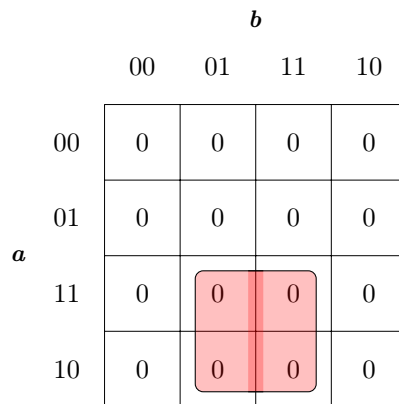
$$F = xy + x(wz + wz')$$

$$F = (BC' + A'D)(AB'+CD')$$

test



Figur 2.19: 4-variabels Karnaugh diagram



Figur 2.20: 4-variabels Karnaugh diagram

2.8.4 Forenkling av uttrykk med Karnaugh Diagram

Gitt Karnaugh diagrammet Figur 2.19 på side 51 hva er det forenklete funksjonsuttrykket?

Gitt Karnaugh diagrammet Figur 2.20 på side 51 hva er det forenklete funksjonsuttrykket?

2.8.5 Porter og kretsdesign

Implementer en 3-inngangs AND-port ved kun bruk av 2-inngangs porter

Implementer en 3-inngangs NOR-port ved kun bruk av 2-inngangs porter

2.8.6 NØTTER godt og blandet

Vis sannhetstabellen til funksjonen $F=xy + xy' + y'z$.

Hva er sannhetstabellen for en to bits binær addisjon?

Vis sannhetstabellen for en 4-inngangs NOT.

Implementer en 3-inngangs XOR-port ved kun bruk av 2-inngangs porter

Lag en 1-bits addisjonskrets. Vis sannhetstabellen, sett opp funksjonsuttrykket, forenkle uttrykket ved hjelp av regneregler. Tegn opp kretsen med porter.

Lag en 2-bits addisjonskrets.

Hvordan kan du modifisere addisjonskretsen i oppgave Avsnitt 2.8.6 til å kunne subtrahere?

Gitt tabell XXX, hvor mange funksjonsuttrykk kan du sette opp for F?

Kapittel 3

Fra Logikk til Kretsimplementasjon

En krets beskrives ofte gjennom et funksjonsuttrykk med boolsk algebra, men hvordan kobles disse boolske operasjonene sammen i en fysisk form? Vi har lært i boolsk algebra operasjoner, men hvordan implementerer man kretser basert på disse operasjonene? I dette kapitlet tar vi for oss veien videre fra logikk operasjoner til å implementere kretser.

3.1 Introduksjon

Vi startet tidlig med å introdusere tankegangen om «black box», Figur 2.1 på side 10, nå kan vi se på hvordan vi kan implementere en krets inne i denne boksen. Så vi går ned i dybden i abstraksjonsnivået. En funksjon består oftest av flere mindre funksjoner (subfunksjon), akkurat som funksjonen $F = AB + BC$ består av to mindre funksjoner, den ene subfunksjonen er AB og den andre BC . Vi bruker ofte terminologien «port» som en subfunksjon. I Figur 3.1 på neste side viser vi hvordan det kan se ut inne i funksjonen F med de to tilhørende subfunksjonene. Utgangen av subfunksjonene skal legges sammen som en OR-operasjon. **I elektronikkens verden kan man ikke koble signalveien sammen og anta at det blir en OR-operasjon¹.** Vi trenger derfor å lære oss først noen basis operasjoners kretsimplementasjon, det vil si vi må lære oss hvordan en NOT, AND og OR kan implementeres.

3.1.1 Forutsetninger

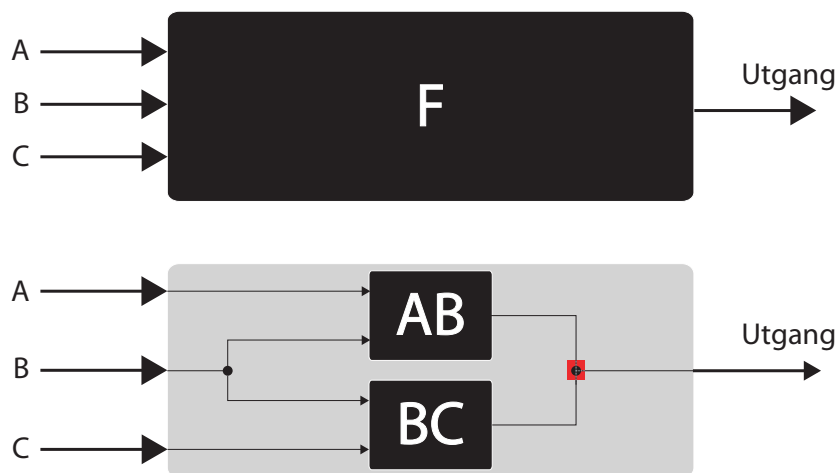
Det forventes at du har grunnleggende forståelse for Boolsk Algebra, logikk operasjoner og kunne sette opp funksjonsuttrykk. Du skal kunne forenkle funksjonsuttrykk både med regneregler og Karnaugh diagram.

3.1.2 Læringsmål

Bloom's 1: Ha grunnleggende kunnskap om «black box» tankegang, oppsett av innganger og utganger for en krets

Bloom's 2: Forstå måten å implementere en krets fra et funksjonsuttrykk.

¹en sannhet med modifikasjoner



Figur 3.1: Hvordan en funksjon kan implementeres med subfunksjoner. Det er funksjonen $F = AB + BC$ som er avbildet. Det røde punktet markerer at det er en feil.

Bloom's 3: Kunne anvende porter for å implementere en krets.

Bloom's 4: Kunne analysere en funksjonsuttrykk og krets for å oppnå mer effektiv implementasjon.

Kunne analysere en krets og skrive funksjonsuttrykket til kretsen

Bloom's 5: Kunne designe effektive kretser med bruk av færrest mulig porter




Bloom's 6: Kunne

3.1.3 Nøkkeltermer

Basis porter, sammensatte porter, kretsimplementasjon, analyse av kretser, bubble-push.

3.2 Basis kretselementer og porter

De tre basis logikk operasjonene NOT, AND og OR har egne symboler i digital elektronikk. Vi kaller dem for porter. I Tabell 3.1 på neste side er de tre basis funksjonene vist med tilhørende symbol, funksjonsuttrykk og sannhetsverditabell. NOT-porten er den eneste porten som kun kan ha én inngang, mens alle de andre portene kan ha flere innganger. Av elektriske hensyn så foretrekkes det å bruke så få innganger på porter som mulig, dette kalles for «fan-in». Vi skal nå se på hvordan vi bruker disse portene til å implementere en funksjonsuttrykk, som foreksempel $F = ab$. Det vi vet er at funksjonen er en AND-operasjon og symbolet for en AND-operasjon i kretsimplementasjon er gitt i midterste rad i Tabell 3.1 på neste side. Tilsvarende for en funksjon gitt som $F = a + b$, vet vi er en OR-operasjon hvor symbolet er gitt i den nederste raden i Tabell 3.1 på neste side. Vi kan

Port	Symbol	Funksjon	Sannhetstabell															
NOT	A  F	$F = \bar{a}$	<table border="1"> <thead> <tr> <th>a</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	a	F	0	1	1	0									
a	F																	
0	1																	
1	0																	
AND	A  B F	$F = a \cdot b$	<table border="1"> <thead> <tr> <th>a</th> <th>b</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	a	b	F	0	0	0	0	1	0	1	0	0	1	1	1
a	b	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR	A  B F	$F = a + b$	<table border="1"> <thead> <tr> <th>a</th> <th>b</th> <th>F</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	a	b	F	0	0	0	0	1	1	1	0	1	1	1	1
a	b	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																

Tabell 3.1: Oversikt over basis portene, med kretssymbol, funksjonsuttrykk og sannhetsverditabell.

bruke symbolene for disse operasjonene til å designe en krets med funksjonsuttrykk som har flere ledd. Vi kan ta som et eksempel funksjonsuttrykket $F = \bar{a}b + a\bar{b}$.

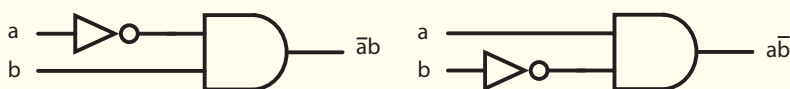
Eksempel 3.14 Implementer følgende funksjon: $F = \bar{a}b + a\bar{b}$

Vi kan først se på hvordan funksjonsuttrykket er bygget opp med hensyn på hvilke operasjoner den utfører. Fra funksjonen ser vi at det består av to ledd, altså subfunksjoner:

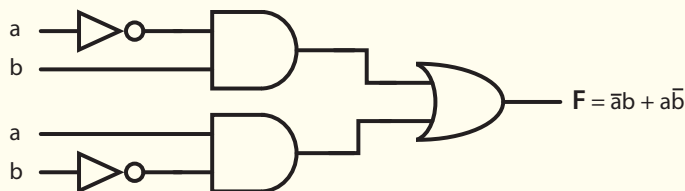
$\bar{a}b$ består av en invertering (NOT) av a og en AND-operasjon med b .

$a\bar{b}$ består av en invertering (NOT) av b og en AND-operasjon med a .

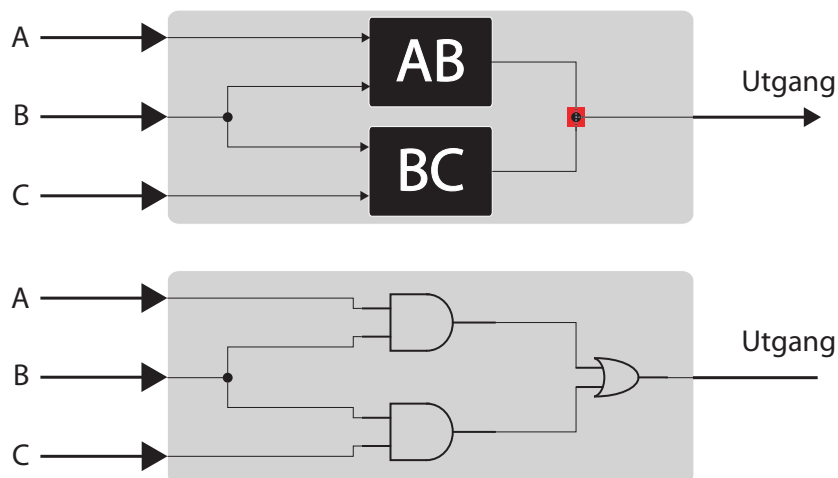
Vi kan implementere disse to subfunksjonene som følgende:



Måten å koble disse to subfunksjonene (leddene) sammen er å bruke samme operasjon som funksjonen har, altså en OR-operasjon « + » slik som figuren under viser:



La oss implementere funksjonen i Figur 3.1 på side 54 som er $F = AB + BC$. Dette gjøres ved å ha to AND-porter, den ene med inngang A og inngang B og den andre med inngang B og inngang C. Deretter kobles utgangen til hver av AND-portene til hver sin inngang på en OR-port. Utgangen av OR-porten blir da funksjonen F. Det endelige resultatet er vist i figurfigur 3.2.



Figur 3.2: Her er portene for henholdsvis AB og BC satt inn. Løsningen for det røde punktet er å bruke en OR-port.





Visste du at det å koble sammen alle inngangene til samme signal (A) for AND og OR gir oss en buffer, altså utgangen = inngangen.

Som nevnt tidligere kan enhver boolsk funksjon implementeres med å bruke disse basis portene. Når det kommer til de elektriske egenskapene til portene, så foretrekkes det to andre porter, NAND og NOR. Disse portene er lettere å bruke for syntese og vesentlig raskere i hastighet. I neste avsnitt presenterer vi NAND og NOR.

3.2.1 Sammensatte porter

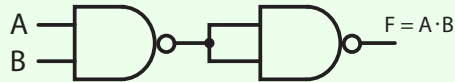
Tilsvarende AND og OR-portene har også NAND, NOR, XOR og XNOR egne symboler. Vi kaller XOR og XNOR for spesial porter, mens NAND og NOR for inverterende generelle porter. Disse er listet opp i Tabell 3.2 på neste side. En fordel til som NAND og NOR har hver for seg er at de alene kan implementere enhver boolsk funksjon. Naturligvis vil man bruke flere porter enn om man har tilgjengelig NOT, AND og OR.

Siden NAND og NOR er en inverterende operasjon, så vil det å koble sammen alle inngangene til samme signal (A) for NAND og NOR gir oss en inverter. På den måten kan man lage en

Port	Symbol	Funksjon	Sannhetstabell
NAND		$F = \overline{AB}$	TABELL
NOR		$F = \overline{A \cdot B}$	TABELL
XOR		$F = A \oplus B$	Tabell
XNOR		$F = \overline{A \oplus B}$	Tabell

Tabell 3.2: Oversikt over NAND, NOR, XOR og XNOR portene, med kretssymbol, funksjonsuttrykk og sannhetsverditabell.

AND eller OR. Se figuren under:



3.2.2 Porter med flere innganger

Alle portene (med unntak av NOT-porten) kan ha mange innganger og det tegnes inn så mange innganger som ønskes i symbolet. I lys av elektriske egenskaper så ønskes det å ha porter med så få innganger som mulig (fan-in). Det er en viss sammenheng med antall innganger og hvor stort areal porten bruker. Vi kan forenklet beregne arealforbruk på porter som at en port med en inngang bruker én arealenhet og en port med to-innganger bruker to arealenheter osv. Dette vil gi oss mulighet til å kunne sammenligne ulike funksjonsuttrykk².

3.3 Kretser med flere innganger og utganger

Hittil har vi stortsett gått igjennom kretser som har bare en utgang, men slik er nok ikke verden. La oss lage en krets som skal sammenligne to bit og gi oss svar på hvem av dem som er størst eller om de er like. Kretsen vil da ha to inngangssignaler, **A** og **B**, og tre utgangssignaler, (**A er størst**), (**B er størst**) og (**de er like**). Vi kan sette opp en *black box* som i Figur 3.3 på neste side.

For å kunne lage et funksjonsuttrykk for hver av utgangene trenger vi å sette opp en sannhetstabell som følger i Tabell 3.3 på neste side. I fra tabellen kan vi lese ut hvor hver av inngangene er '1' og deres representative funksjonsuttrykk blir da følgende:

$$\begin{aligned}
 F_{(A>B)} &= \overline{A} \cdot B \\
 F_{(A<B)} &= A \cdot \overline{B} \\
 F_{(A=B)} &= \overline{A} \cdot \overline{B} + A \cdot B = \overline{A \oplus B}
 \end{aligned}
 \tag{3.1}$$

²Vi ser bort i fra at det er forskjellige arealforbruk innbyrdes mellom portene. Altså om en AND-port bruker mer areal enn en OR-port.

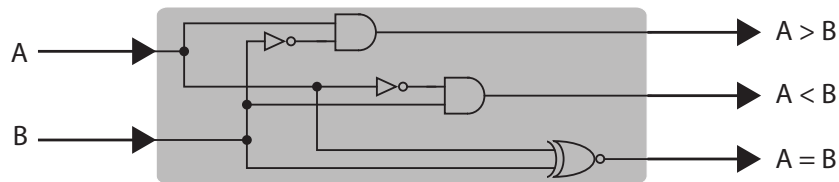


Figur 3.3: Oppstartssteget for å implementere en krets. Vi starter med å sette antall innganger og utganger.

A	B	$F_{(A>B)}$	$F_{(A<B)}$	$F_{(A=B)}$
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

Tabell 3.3: Sannhetsverditabellen for en enkel 2-bits komparator.

Med disse funksjonsuttrykkene kan vi fylle inn i boksen med de representative logiske portene som vist i Figur 3.4.



Figur 3.4: Kretsimplementasjonen med porter for funksjonene gitt i Likning (3.1) på side 57

3.4 Kretsimplementasjon av 7-segment skjerm

Implementeringen av 7-segment skjermen kan gjøres på flere måter. Vi opparbeidet oss mange funksjonsuttrykk gjennom Kapittel 2 for en 7-segments skjerm. Hvor godt et funksjonuttrykk er forenklet har mye å si iforhold til hvor stor kretsimplementasjonen blir. Tar vi utgangspunktet i arealenhet som beskrevet i Avsnitt 3.2.2 på side 57 så kan vi sette opp en oversikt over forskjellene mellom funksjonsuttrykkene. Tabell 3.4 på neste side viser hvor mange arealenheter hver av funksjonsuttrykkene trenger med hensyn på de ulike forenklingsmetodene vi har vist i Kapittel 2. Funksjonsuttrykkene oppgitt i kolonne *utgangspunkt* er hentet fra Likning 2.30 - Likning (2.36) på side 26. Metoden for utregningen er som følger for S_a :

$$\begin{aligned}
 S_a = & \overline{T_3}\overline{T_2}\overline{T_1}\overline{T_0} + \overline{T_3}\overline{T_2}T_1\overline{T_0} + \overline{T_3}\overline{T_2}T_1T_0 + \overline{T_3}T_2\overline{T_1}T_0 + \overline{T_3}T_2T_1\overline{T_0} \\
 & + \overline{T_3}T_2T_1T_0 + T_3\overline{T_2}\overline{T_1}\overline{T_0} + T_3\overline{T_2}\overline{T_1}T_0
 \end{aligned}
 \tag{3.2}$$

Det er 19 inverteringer (NOT) \Rightarrow 19 arealenheter, det er 8 stk av 4-inngangsport (AND) \Rightarrow 32 arealenheter, det er 8 ledd som betyr at vi trenger en 8-inngangsport (OR) \Rightarrow 8 arealenheter. Totalt vil dette utgjøre **59 arealenheter**. Ser vi på samme funksjonsuttrykk med forenkling ved hjelp av regneregler får vi fra Likning (2.44) på side 31:

$$S_a = \bar{T}_3 T_1 + T_3 \bar{T}_2 \bar{T}_1 + \bar{T}_3 \bar{T}_2 \bar{T}_1 \bar{T}_0 + \bar{T}_3 T_2 \bar{T}_1 T_0 \quad (3.3)$$

som gir oss 9 inverteringer (NOT), en 2-inngangsport (AND), en 3-inngangsport (AND), to 4-inngangsport (AND) og tilslutt har vi fire ledd som bruker en 4-inngangsport (OR). Tilsammen vil dette gi totalt **26 arealenheter**. Ved å forenkle har vi klart å redusere mer enn halvparten av arealet. Ser vi på metoden for Karnaugh diagram så får vi uttrykket gitt i Figur 2.15 på side 44:

$$S_a = \bar{T}_3 T_1 + T_3 \bar{T}_2 \bar{T}_1 + \bar{T}_2 \bar{T}_1 \bar{T}_0 + \bar{T}_3 T_2 T_0 \quad (3.4)$$

som gir oss 7 inverteringer (NOT), en 2-inngangsport (AND), tre 3-inngangsport (AND) og tilslutt har vi fire ledd som bruker en 4-inngangsport (OR). Tilsammen vil dette gi totalt **22 arealenheter**. Det er noe bedre enn med regneregler og nesten en tredjedel av utgangspunktet. Hvis vi tar med don't care verdier fra Figur 2.15 på side 44:

$$S_a = T_3 + T_1 + \bar{T}_2 \bar{T}_0 + T_2 T_0 \quad (3.5)$$

som gir oss 2 inverteringer (NOT), to 2-inngangsport (AND) og tilslutt har vi fire ledd som bruker en 4-inngangsport (OR). Tilsammen vil dette gi totalt **10 arealenheter**. Resten av funksjonsuttrykkene for en 7-segment skjerm er vist i Tabell 3.4.

	utgangs- punktet	Regne- regler	Karnaugh Diagram	don't care	Reduksjon
S_a	59	26	22	10	83 %
S_b	61	25	22	10	84 %
S_c	51	20	13	4	92 %
S_d	46	27	26	18	61 %
S_e	32	13	13	9	72 %
S_f	38	29	25	13	66 %
S_g	44	35	25	13	71 %
Totalt	331	175	146	77	77 %

Tabell 3.4: Oversikt over antall porter som trengs for å implementere funksjonsuttrykkene etter å ha forenklet med ulike metoder. Vi har forenklet telling av antall porter slik at det ikke er med gjenbruksmuligheter av signaler eller porter.

Vi har igjennom hele dette kapittelet gått igjennom eksempler for funksjonsuttrykket en 7-segment skjerm. I denne seksjonen samler vi sammen alle beregninger og funksjonsuttrykk og ser på hvordan denne kan implementeres. Vi henter inn 7-segment skjermen fra figur 2.2 på side 11 og tilhørende «black box» i fra figur 2.3 på side 11 og setter dem side om side i figur 3.5 på neste side. Funksjonsuttrykkene jobbet vi med i ?? på side ?? og i ?? på side ?? listet vi opp alle funksjonene for utgangene a,b,c,d,e,f,g.