

UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

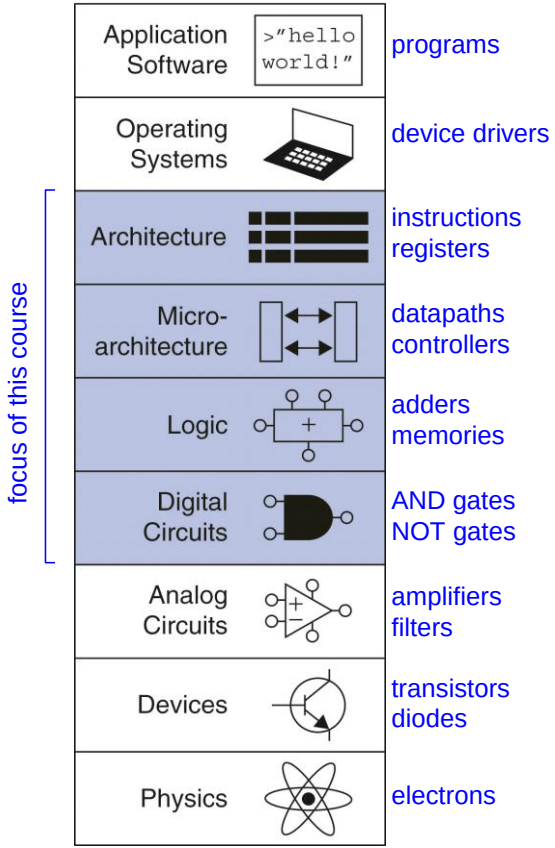
IN2060 – Forelesning 2

Design av kombinatorisk logikk (Kap. 2)

Tema - Forelesning 2

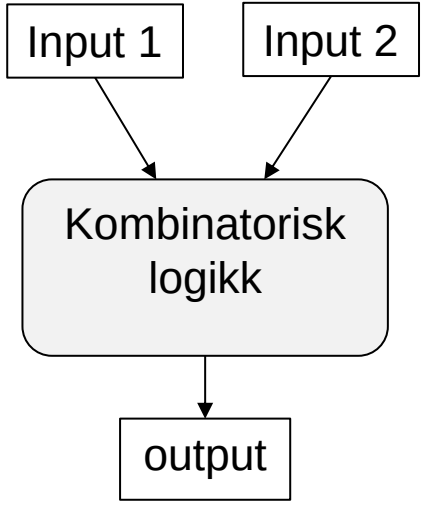
Kombinatorisk logikk

- Boolske funksjoner
- Sannhetstabeller
- Boolsk algebra
  - Aksiomer og teoremer
  - Litt bevis av teoremer
  - Forenkling
- Diverse:
  - Noen større kombinatoriske kretser
  - Portforsinkelse



Kombinatorisk logikk

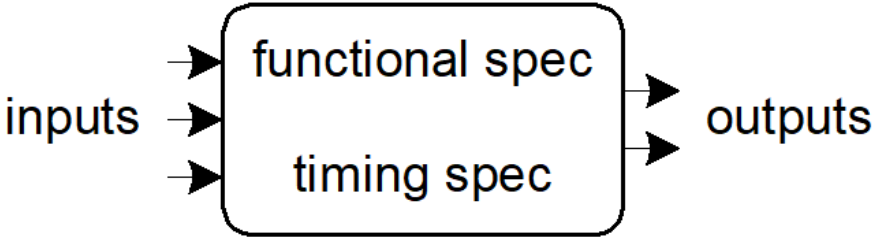
- For å effektivt kunne implementere logiske operasjoner trenger vi å
  - Spesifisere funksjonen ned på bit nivå
  - Teste / validere
  - Optimalisere / forenkle
- Vi skal gå gjennom flere prinsipper i dag som gir oss verktøy for dette.



Kombinatorisk logikk

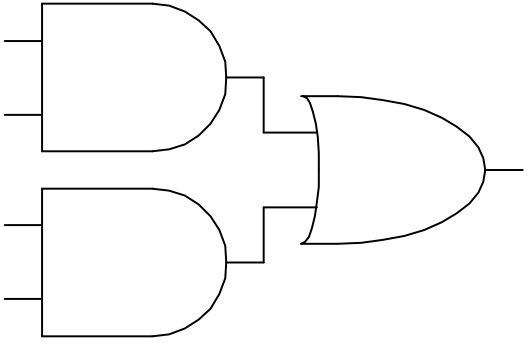
Kombinatorisk logikk består av:

- En eller flere digitale innganger
- En eller flere digitale utganger
- Funksjonsbeskrivelse
- En tidspesifikasjon.



Kombinatoriske logikk - definisjon

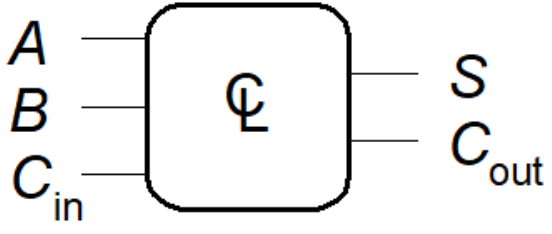
- Består kun av elementer/moduler som er kombinatoriske.
- Alle noder er enten en inngang eller er koblet til kun en utgang.
- Kan ikke inneholde tilbakekoblinger.
  - *Utgangen(e) er en funksjon av nåværende* innganger



Kombinatorisk krets uttrykt ved boolske funksjoner

- Funksjonell beskrivelse av kretsen
  - Utgangene = funksjon av innganger.

$$S = F(A, B, C_{in})$$
$$C_{out} = F(A, B, C_{in})$$



$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Kombinatorisk krets uttrykt ved Sannhetstabell

- En boolsk funksjon kan visualiseres/beskrives i en sannhetstabell
- Eksempel:
  - $F = x + y'z$
- En gitt funksjon har kun en sannhetstabell
- Men, en gitt sannhetstabell har uendelig mange funksjonsuttrykk

x	y	z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Oppramsing av alle mulige inngangskombinasjoner (2^N)

Noen *boolske* definisjoner

- **Komplement**: variabel med merke/strek over
  - A', B', C' eller $\bar{A}, \bar{B}, \bar{C}$
- **Literal**: Alle variabler (inkludert komplementene)
  - A, A', B, B', C, C' eller $A, \bar{A}, B, \bar{B}, C, \bar{C}$
- **Implikant**: produkt av literaler
  - $AB'C, A'C, BC$
- **Minterm**: *produkt* som inkluderer alle variablene
  - $ABC, ABC', AB'C'$
- **Maxterm**: *sum* som inkluderer alle variablene
  - $(A + B + C), (A + B + C'), (A + B' + C')$

Systematisk måte å gå direkte fra en sannhetstabell til et logisk uttrykk.

Sum-av-produkter (mintermer)

- Alle boolske funksjoner kan uttrykkes som en sum av mintermer.
  - Dvs. vi kan spesifisere en ønsket oppførsel som en sum av mintermer.
- Mintermene gjenspeiler radene fra sannhetstabellen
- Funksjonsbeskrivelsen for kretsen lages ved å OR'e mintermer som er 1.
  - Derav *sum* (OR) *av produkter* (AND).

Eksempel – sum-av-produkter

Finner *funksjonsbeskrivelsen* for kretsen/sannhetstabellen ved å OR'e mintermer som er 1.

A	*B*	*Y*	minterm	minterm name
0	0	0	$\overline{A}\ \overline{B}$	m_0
0	1	1	$\overline{A}\ B$	m_1
1	0	0	$A\ \overline{B}$	m_2
1	1	1	$A\ B$	m_3

Y = ?

Y = A'B + AB

Y = Σ(1, 3)

Produkt-av-summer (makstermer)

Vi har tilsvarende for makstermer.

- Alle boolske funksjoner kan også uttrykkes som et produkt av maxtermer.
- Funksjonsbeskrivelsen lages ved å AND'e de makstermene som har verdi 0 (FALSE).
  - Dvs. produkt (AND) av summer (OR).

Eksempel: Produkt-av-summer

Finner *funksjonsbeskrivelsen* ved å AND'e de makstermene som har verdi 0 (FALSE).

A	*B*	*Y*	maxterm	maxterm name
0	0	0	A + B	M_0
0	1	1	A + $\overline{B}$	M_1
1	0	0	$\overline{A}$ + B	M_2
1	1	1	$\overline{A}$ + $\overline{B}$	M_3

Y = ?

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

$Y = \Pi(0, 2)$

Eksempel: Sannhetstabell til funksjonsutrykk

- Når er en piknik aktuell? P
  - Dersom det **ikke er insekter** (I'), og
  - Dersom det **ikke regner** (R')
- Da har vi det vi trenger for å sette opp sannhetstabellen.

$I_{nsekter}$	R_{egn}	P_{iknik}
0	0	1
0	1	0
1	0	0
1	1	0

Minterm vs Maxterm

I	R	P	minterm
0	0	1	I'R'
0	1	0	I'R
1	0	0	IR'
1	1	0	IR

$P = I'R'$

$= \sum(0)$

I	R	P	maxterm
0	0	1	(I + R)
0	1	0	(I + R')
1	0	0	(I' + R)
1	1	0	(I' + R')

$P = (I + R')(I' + R)(I' + R')$

$= \prod(1,2,3)$

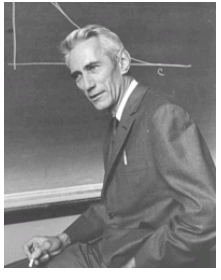
Merk! Vi har funnet to forskjellige funksjonsutrykk for den samme sannhetstabellen. Gir identisk utgang, dvs. I'R' = (I + R')(I' + R ')(I' + R').

BOOLSK ALGEBRA

Hvorfor Boolsk Algebra ?

- Det fantes allerede et helt felt innen matematikken som kun opererte med to verdier, *true* and *false*.
- Shannon viste i 1937 at *Boolsk algebra* var et veldig nyttig verktøy for digitale systemer.

- Muliggjør
  - *design* av komplekse digitale systemer
  - *analyse* av komplekse digitale systemer
  - *forenkling* av logiske funksjoner og den fysiske implementasjon.



(C.E. Shannon)

Boolsk algebra

- Reglene ligner på vanlig algebra, men er noe «enklere» siden vi kun opererer med to verdier (0, 1)
- Baserer seg på
  - **Aksiomer**: antakelser vi ikke prøver å bevise, bare godtar (konstruksjoner).
  - **Teoremer:** som vi beviser fra aksiomene.
    - Teoremene er de nyttige reglene som vi kan bruke til å manipulere/forenkle uttrykk.

Aksiomer

- Aksiomene tar vi for gitt.
- Aksiomene definerer de *Boolske operasjonene* AND, OR og NOT.

Number	Axiom	Name
A1	$B = 0$ if $B \neq 1$	Binary Field
A2	$\overline{0} = 1$	NOT
A3	$0 \bullet 0 = 0$	AND/OR
A4	$1 \bullet 1 = 1$	AND/OR
A5	$0 \bullet 1 = 1 \bullet 0 = 0$	AND/OR

Aksiomer - Dualitet

- Både aksiomer og teoremer tilfredstiller *dualitet*.
- *Dualitet*: Hvis vi bytter 0 ↔ 1 og OR ↔ AND, vil uttrykket fortsatt være gyldig.
  - Den duale til A1 skriver vi som A1'

Number	Axiom	Dual	Name
A1	B = 0 if B ≠ 1	B = 1 if B ≠ 0	Binary Field
A2	$\overline{0} = 1$	$\overline{1} = 0$	NOT
A3	0 • 0 = 0	1 + 1 = 1	AND/OR
A4	1 • 1 = 1	0 + 0 = 0	AND/OR
A5	0 • 1 = 1 • 0 = 0	1 + 0 = 0 + 1 = 1	AND/OR

Teoremer med en variabel

- Teoremene T1 til T5 beskriver hvordan vi kan forenkle uttrykk med en variabel.
- Utledes direkte fra Aksiomene.

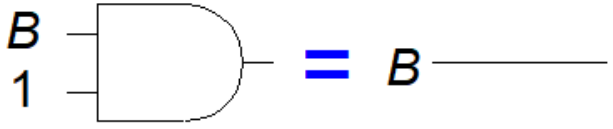
Number	Theorem	Dual	Name
T1	B • 1 = B	B + 0 = B	Identity
T2	B • 0 = 0	B + 1 = 1	Null Element
T3	B • B = B	B + B = B	Idempotency
T4	(B')' = B		Involution
T5	B • B' = 0	B + B' = 1	Complements

AND

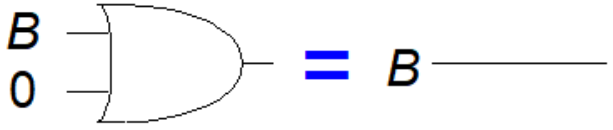
X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

T1 – Identitet

- $B \cdot 1 = B$



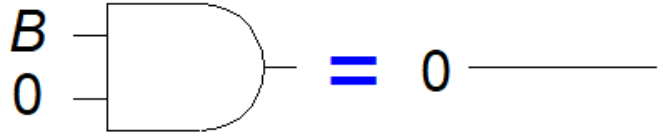
- $B + 0 = B$



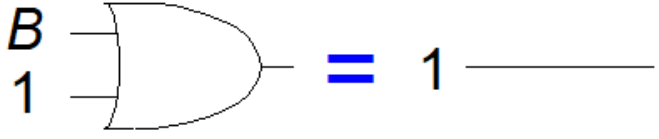
Vi kan erstatte porten med en direktekobling (ledning).

T2 - Null Element (dominerende)

- $B \cdot 0 = 0$

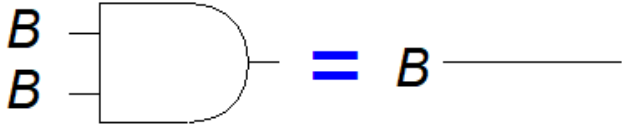


- $B + 1 = 1$

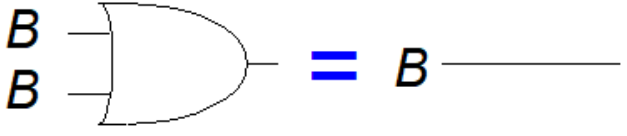


T3 - Idempotency

- $B \cdot B = B$



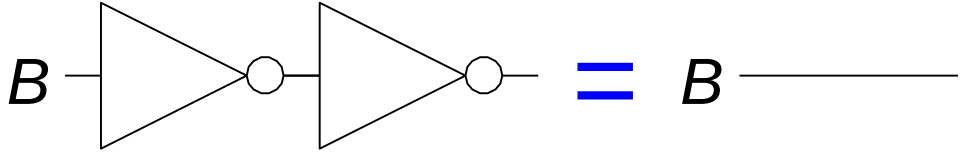
- $B + B = B$



Å kombinere et signal med seg selv endrer ingenting.

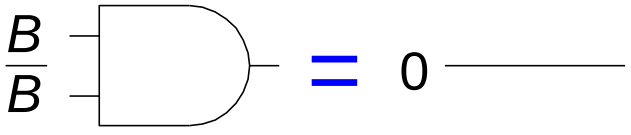
T4 - Involution

- $\bar{\bar{B}} = B$

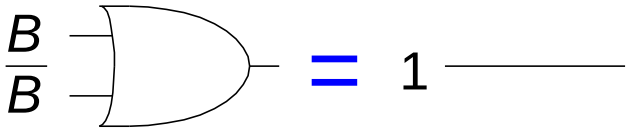


T5 - Complements

- $B \cdot \bar{B} = 0$



- $B + \bar{B} = 1$



Teoremer med flere variable

Number	Theorem	Name
T6	$B \bullet C = C \bullet B$	Commutativity
T7	$(B \bullet C) \bullet D = B \bullet (C \bullet D)$	Associativity
T8	$B \bullet (C + D) = (B \bullet C) + (B \bullet D)$	Distributivity
T9	$B \bullet (B + C) = B$	Covering
T10	$(B \bullet C) + (B \bullet \overline{C}) = B$	Combining
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D) = (B \bullet C) + (\overline{B} \bullet D)$	Consensus

- Hvordan vet vi dette?

Metoder for å bevise

- Metode 1: Perfect induction
  - Teste alle mulige kombinasjoner
- Metode 2: Via algebra
  - Bruke andre teoremer og aksiomer for å bevise sammenhengen.
    - Dvs. “Få en side av ligningen til å bli identisk med den andre siden.”

Bevis via Perfect Induction

- Sjekke alle mulige inngangskombinasjoner, dvs. sannhetstabellen.
- Hvis begge utrykkene gir de samme utgangene for alle mulige innganger, er utrykkene like.

Bevis med *Perfect Induction*

Number	Theorem	Name
T6	B•C = C•B	Commutativity

B	*C*	*BC*	*CB*
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

- Sannhetstabbelen viser at BC er identisk med CB.

Bevis via Algebra

Number	Theorem	Name
T9	B• (B+C) = B	Covering

Metode 2: Bevis ved å benytte aksiomer og teoremer.

B•(B+C)	= B•B + B•C	T8: Distributivity
	= **B** + B•C	T3: Idempotency
	= B•(1 + C)	T8: Distributivity
	= B•(**1**)	T2: Null element
	= B	T1: Identity

Teoremer med flere variabler - dualitet

#	Theorem	Dual	Name
T6	B•C = C•B	B+C = C+B	Commutativity
T7	(B•C) • D = B • (C•D)	(B + C) + D = B + (C + D)	Associativity
T8	B • (C + D) = (B•C) + (B•D)	B + (C•D) = (B+C) (B+D)	Distributivity
T9	B • (B+C) = B	B + (B•C) = B	Covering
T10	(B•C) + (B•$\overline{C}$) = B	(B+C) • (B+$\overline{C}$) = B	Combining
T11	(B•C) + ($\overline{B}$•D) + (C•D) = (B•C) + ($\overline{B}$•D)	(B+C) • ($\overline{B}$+D) • (C+D) = (B+C) • ($\overline{B}$+D)	Consensus

- **Merk:** T8’ skiller seg ut fra vanlig algebra.

Forenkling av uttrykk

Mål: Forenkle uttrykk til færrest mulig *implicants*, med færrest mulig *literals*

- **Implicant**: produkter of literals

  ABC, *AC*, *BC*

- **Literal**: variabler og deres komplementer

  A, *A'*, *B*, *B'*, *C*, *C'*

Forenklingsmetoder

- **Distributivity (T8, T8')** B (C+D) = BC + BD
  B + CD = (B+ C)(B+D)
- **Covering (T9')** A + AP = A
- **Combining (T10)** PA' + PA = P
- **Expansion (T10, T9')** P = PA' + PA (utvidelse)
  A = A + AP
- **Duplication** A = A + A
- **"Simplification" theorem** PA' + A = P + A
  PA + A' = P + A'

Bevis “Simplification” Theorem – Metode 1

Skal vise at PA + A’ = P + A’

- **Variasjon 1**:

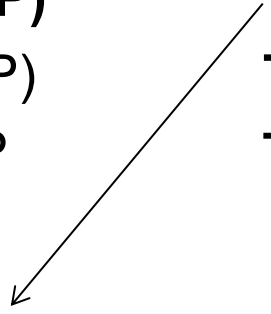
PA+A’	= PA + **(A’ + A’P)**	**T9’**
PA+A’	= PA + **A’P + A’**	**T6**
PA+A’	= P(**A+A’**) + A’	**T8**
PA+A’	= P(**1**) + A’	**T5’**
PA+A’	= P + A’	**T1**

Bevis “Simplification” Theorem – Metode 2

Skal vise at PA + A’ = P + A’

- **Variasjon 2**:

PA+A’	= **(A’+A)(A’+P)**	**T8’ (uvant)**
PA+A’	= **(1)**(A’+P)	**T5’**
PA+A’	= A’+P	**T1**

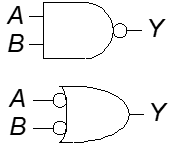


T8’: *B + (C•D) = (B+C) (B+D)*
CD + B = (B+C) (B+D)

T12 - De Morgan teoremet

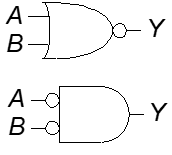
#	Theorem	Dual	Name
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \ldots} = \overline{B_0} + \overline{B_1} + \overline{B_2} \ldots$	$\overline{B_0 + B_1 + B_2 \ldots} = \overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2} \ldots$	DeMorgan's Theorem

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



Komplementet av et produkt er summen av komplementene

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



Komplementet av en sum er produktet av komplementene

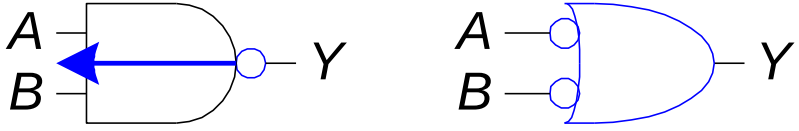
T12 - De Morgan eksempel

$$
\begin{aligned}
Y &= \overline{((A + \overline{BD}) \cdot \bar{C})} \\
&= \overline{(A + \overline{BD})} + \bar{\bar{C}} && \text{T12} \\
&= \left(\bar{A} \cdot \overline{\overline{BD}}\right) + C && \text{T12 + T4} \\
&= (\bar{A} \cdot BD) + C && \text{T4} \\
&= \bar{A}BD + C
\end{aligned}
$$

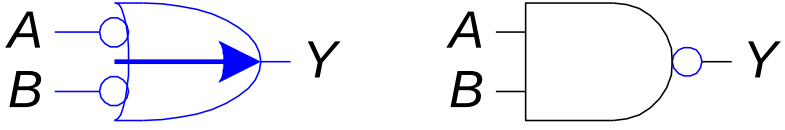
“Bubble Pushing” (De Morgan, T12)

Praktisk metode for å finne funksjonsuttrykket når det er brukt mye NOR og NAND porter (CMOS).

- **Baklengs:**
  - Kroppen forandres
  - Inngangene får bobler.



- **Forlengs:**
  - Kroppen forandres
  - Utgangen får boble.

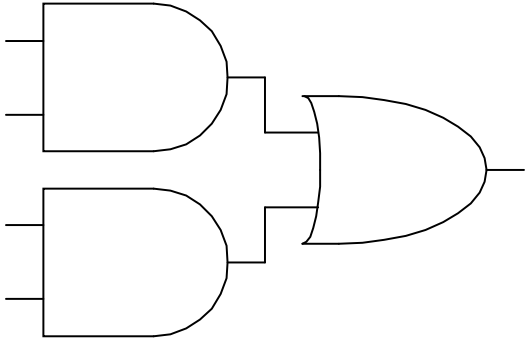


Noen ord om forenkling

- Forenkling ved Boolsk Algebra er litt «prøve seg fram», ikke metodisk .
  - Men forståelse av Boolsk Algebra er viktig, selve fundamentet til digital teknologi.
- Karnaugh Maps er en grafisk «metode». Fungerer kun for 4 (5) variabler.
  - Brukes ikke i praksis.
  - Ikke pensum IN2060.
- I dag brukes diverse *heuristiske algoritmer* som finner gode løsninger på relativt kort tid.
  - Espresso heuristic logic minimizer (IBM - 1982)

Noen logiske krets konvensjoner

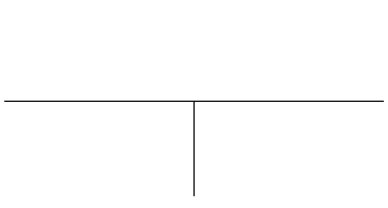
- Signalflyt fra “venstre til høyre”
- Innganger starter fra venstre eller toppen
- Utganger er til høyre eller nederst



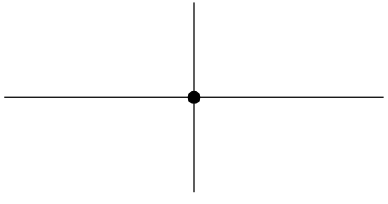
Logiske krets konvensjoner (forts.)

- Vi har alltid kobling ved T-koplinger.
- En “dot” der linjer krysser indikerer kobling.
- Linjer som krysser uten “dot” har ingen kobling.

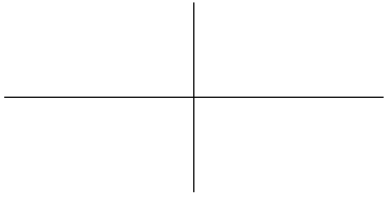
wires connect at a T junction



wires connect at a dot



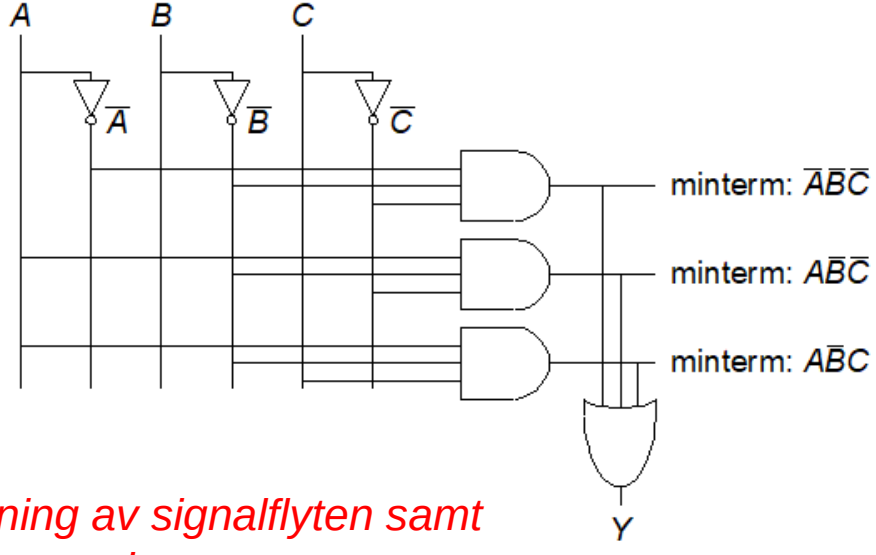
wires crossing without a dot do not connect



Eksempel - logisk krets

Enkel «sum-of-products» implementasjon.

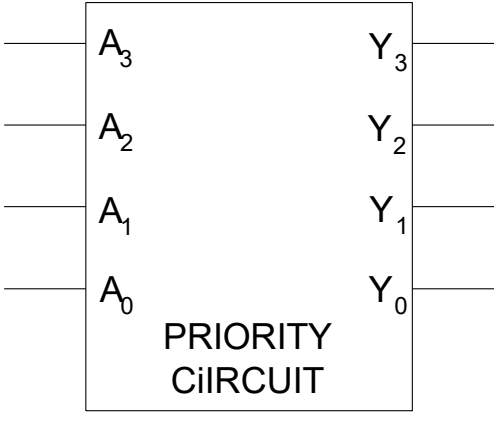
Y= A'B'C' + AB'C' + AB'C



Merk retning av signalflyten samt T-koblinger og kryss.

Kretser med flere utganger

- Eksempel: Prioritetskrets
  - Utgangen skal genspeile det mest signifikante bitet som er høy.

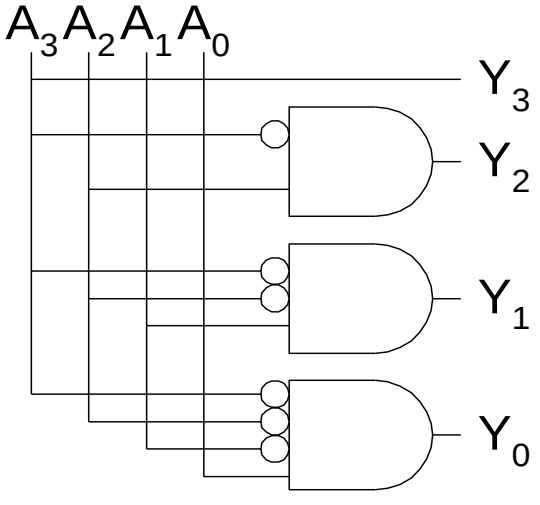


A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

Kretser med flere utganger (forts.)

Kan “lese” løsningen direkte ut av sannhetstabellen.

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0



Don’t cares

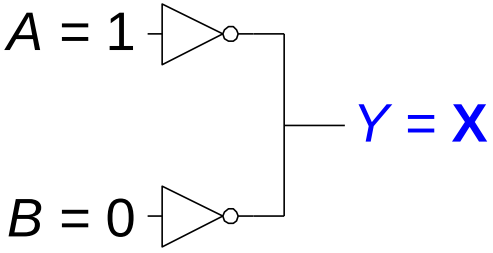
Når andre innganger ikke har noen betydning for resultatet. Markeres med x i sannhetstabellen.

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	x	0	0	1	0
0	1	x	x	0	1	0	0
1	x	x	x	1	0	0	0

Contention - X

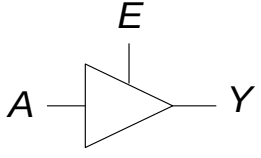
- **Contention:** Krets som prøver å drive en utgang til 0 og 1 samtidig.
  - Faktiske utgangsverdi blir noe midt i mellom. Ofte ustabil.
  - Kan føre til stort strømforbruk (kortslutning).



- Merk:
  - Contention indikerer ofte en feil (bug).
  - x brukes også for “don’t care”. Sjekk konteksten for å skille dem.

Flytende utgang: Z

- Flytende utgang kan være 0, 1 eller noe midt i mellom.
  - Eksempel: **Tri-state buffer**



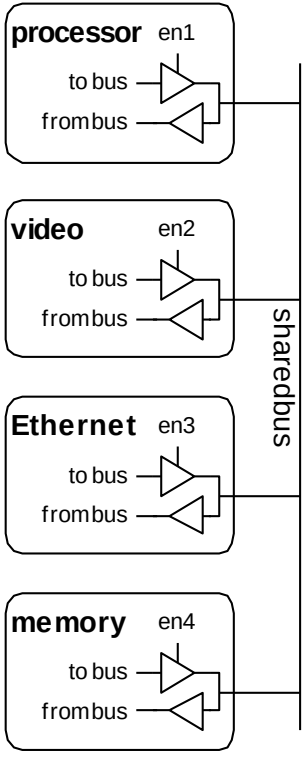
E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

Merk: Et voltmeter vil ikke kunne måle om en node er flytende.

Tristate buss

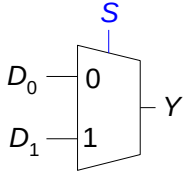
Bruker flytende noder

- Flere moduler kan dele samme “bus” (ledinger).
- Kun en modul er aktiv av gangen.
- Verdiene *en1* til *en4* bestemmer hvilken modul som er aktiv på bussen.



Multiplekser

- Selekterer en av N innganger til å bli satt til å være utgangen.
- Eksempel: 2:1 MUX.
  - Merk: For N innganger trenger vi $\log_2 N$-bit select styresignaler

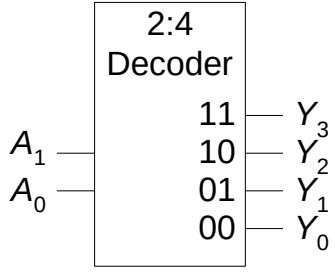


S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S	Y
0	D_0
1	D_1

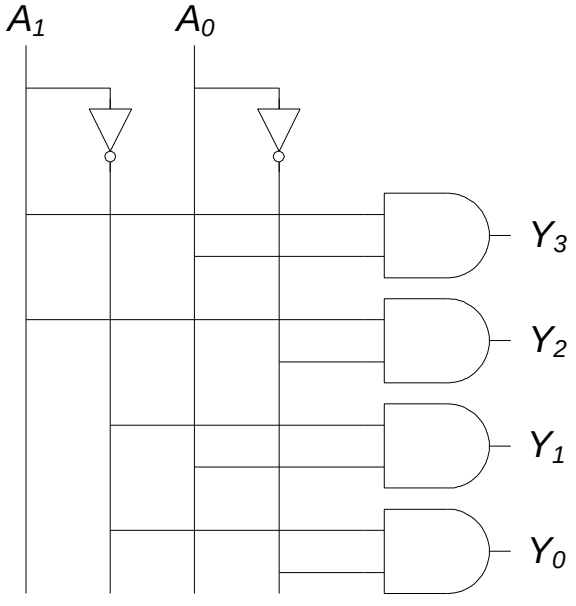
Dekoder

- *N* innganger, 2^N utganger
- Inngangene bestemmer hvilken utgang som skal være høy (gjenspeiler mintermen til inngangen).



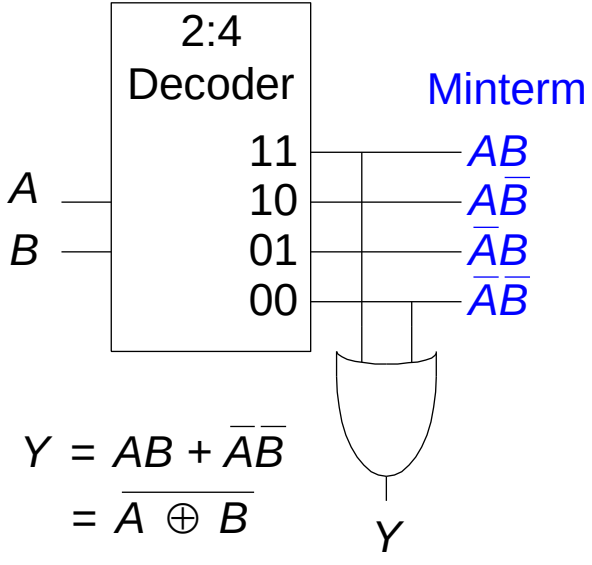
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Dekoder implementasjon



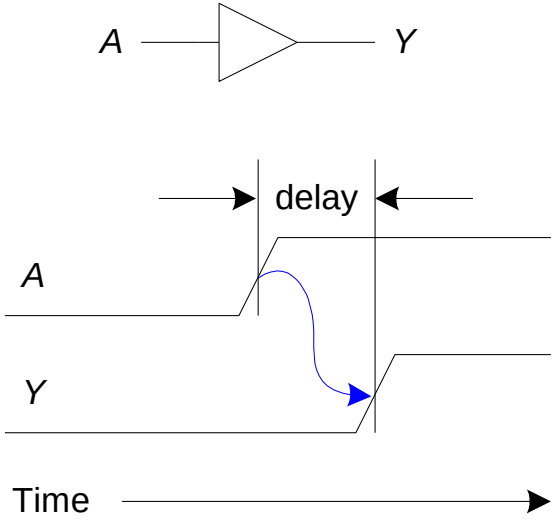
Logikk som bruker dekodere

- Dekoder kombinert med en OR port kan brukes til å implementere logiske funksjoner.

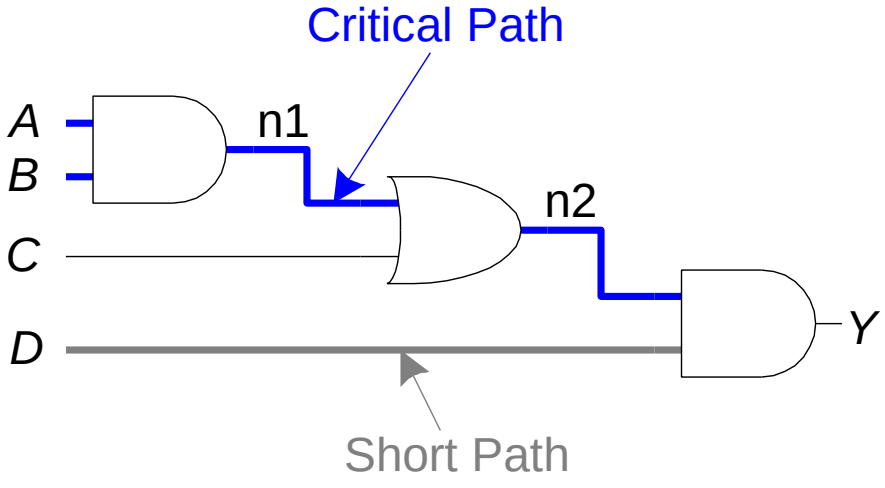


Portforsinkelse

- Logiske porter må implementeres fysisk.
- Det tar litt tid fra inngangen forandrer seg til utgangen reagerer.
  - Kapasitans og resistans i kretsen.
  - Generell begrensning av fart (<lysfarten)
- Alle digitale kretser har tidsbegrensinger.



Critical Path



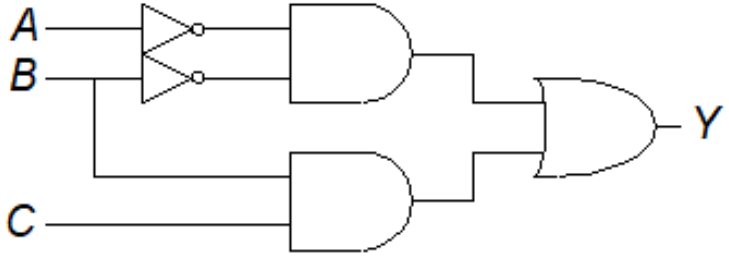
- Ser at *Critical Path* er lengst, dvs. signalet må gjennom flere porter.
  - Signal flyten gjennom *Critical Path* vil ta lengre tid.

Glitches

- Når et enkel forandring på inngangen forårsaker utgangen til å forandre seg flere ganger.
  - Kan ta litt tid før utgangen «bestemmer» seg.
  - Vil stabilisere seg når Critical Path er ferdig.
  - Litt av grunnen til at av vi trenger synkron logikk Kap. 3.

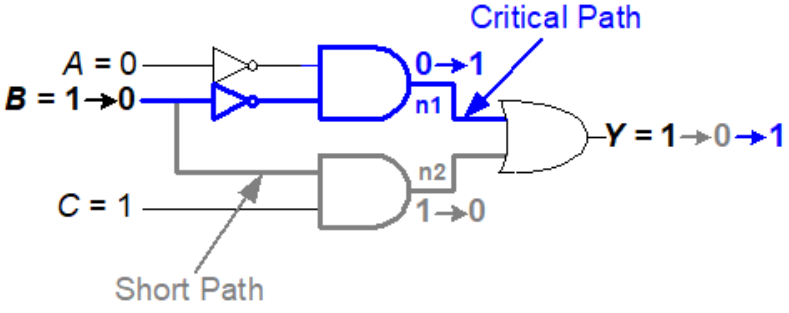
Glitch eksempel

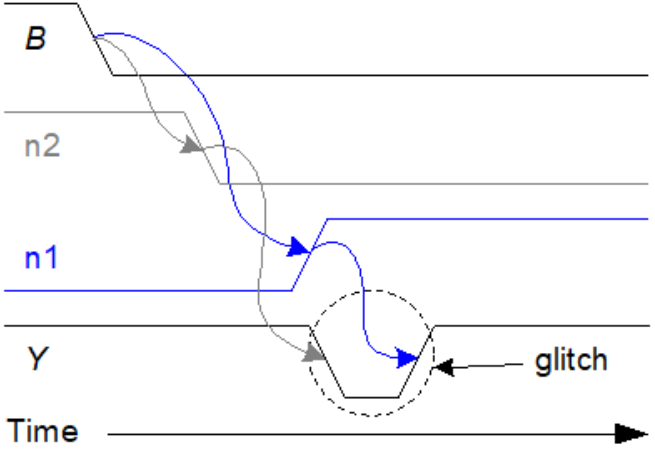
- Hva skjer når:
  - A=0
  - C=1
  - B går fra 1 til 0?



  - Y = A'B'+BC
    - Fra Y= 0'1' + 1·1 = 1 (B=1)
    - Til Y= 0'0' + 0·1 = 1 (B=0)

Glitch eksempel (forts.)





Inngangene til OR porten er ikke ferdig på samme tid. Dette gir oss i dette tilfelle en Glitch.

Hvorfor forstå Glitches?

- Glitches lager normalt ikke problemer pga. synkron logikk (Kap. 3).
- Men kan gi utfordringer for noen spesielle tilfeller (gray-kode, display). Karnaugh map kan benyttes.
- Det kan være viktig å finne dem for å unngå mulige problemer (via simuleringer / oscilloskop).
- Egentlig umulig å unngå.
  - Simultane forandringer på flere innganger kan også skape Glitches.

Oppsummering

- Kombinatorisk logikk
  - Boolske funksjoner
  - Sannhetstabell
  - Minterm og maxtermer
  - Boolsk Algebra
    - Aksiomer / teoremer
    - Bevis
    - Forenkle
  - Hvordan tegne kretser
  - Noen større kretser som *multiplexere* og *dekodere*
  - Portforsinkelse

Til neste gang

- Sekvensiell og synkron logikk + tilstandsmaskiner.
- Viktig kunnskap til neste gang
  - Kontroll på logiske porter og forstå de litt større kombinatoriske kretsene.
  - Forstå koblingen mellom sannhetstabeller og logiske utrykk og porter.
  - Grunnleggende forståelse av boolsk algebra.

- Ikke sett på Karnaugh Map (frivillig, ikke pensum)

Ukeoppgaver – Uke 2

Exercies fra boka

- 2.1, 2.3, 2.5, 2.7, 2.13, 2.15, 2.17, 2.22, 2.24, 2.25, 2.26, 2.33, 2.39 og 2.45.
- **Ekstra**: 2.11, 2.36, 2.40 og 2.48.