Digital teknologi

Digital – To tilstander, enten av eller på (”0 eller 1”, ”true/false”, ”yes/no”)
Lettere lagring i digital i forhold til analog
Signaler sendes enklere med digital, robust signaloverføring

Binær tallsystem (0, 1)

- Et binært tall er representert ved symbolene 0 og 1
- Eks: $(101)_{bin} = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

Binær	oktal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Octale tallsystemer (8 som base)

- Et oktalt tall er representert ved symbolene 0, 1, 2, → 7
- Eks: $(252)_{okt} = 2 \bullet 8^2 + 5 \bullet 8^1 + 2 \bullet 8^0$

Heksadesimale tall (0-9 + a-e)

- Et heksadesimalt tall er representert ved symbolene 0, 1, 2, ... 8, 9, A, B, C, D, E, F
- Eks: $(2B9)_{heks} = 2 \bullet 16^2 + 11 \bullet 16^1 + 9 \bullet 16^0$
- Eksempel med desimal tall:
  $(1A5,1C)_{16} = 1 \bullet 16^2 + 10 \bullet 16^1 + 5 \bullet 16^0 + 1 \bullet 16^{-1} + 12 \bullet 16^{-2} = (421,109375)_{des}$

Konverteringseksempler:

Konverter tallet (41)des til binær

41/2	=	20 + 1/2	$a_0 = 1$ (LSB)
20/2	=	10 + 0/2	$a_1 = 0$
10/2	=	5 + 0/2	$a_2 = 0$
5/2	=	2 + 1/2	$a_3 = 1$
2/2	=	1 + 0/2	$a_4 = 0$
1/2	=	0 + 1/2	$a_5 = 1$

Dermed: $(41)_{des} = (101001)_{bin}$

Binær	Heksadesimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Binær logikk

Definerte basis operasjoner:

- AND “ • ”
- OR “ + ”
- NOT “ ´ ”

Sannhetstabell

AND

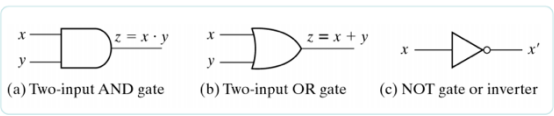
X Y	Z
0 0	0
0 1	0
1 0	0
1 1	1

OR

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

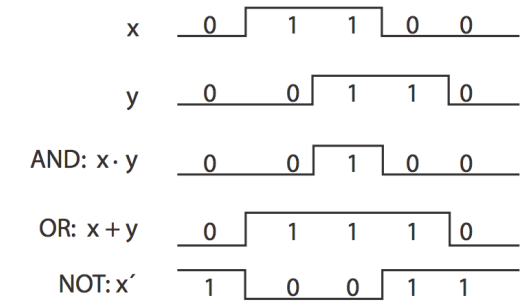
NOT

X	Y
0	1
1	0



(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

Input-output signaler for portene



Boolsk Algebra – uke 2

Algebra – Studie av algebraiske system

Algebraisk system:

- Mengde av elementer
- Et sett av operasjoner på elementene (funksjoner) som tilfredsstiller en liste av aksiomer/postulater
  - *Aksiom/postulat* – Regler som ikke skal bevises (utgangspunkt for algebraisk system)

"Toverdi" Boolsk algebra

- Mengde: {0, 1}
- 2 operasjoner på mengden over: OR "+" *og* AND "•"

OR

x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

AND

x	y	x•y
0	0	0
0	1	0
1	0	0
1	1	1

DeMorgans teorem

$(x \cdot y)' = x' + y'$ $\qquad$ $(x + y)' = x' \cdot y'$

På invertert form:

$x \cdot y = (x' + y')'$ $\qquad$ $x + y = (x' \cdot y')'$

Regneregler – oversikt

$x + 0 = x$	$x \cdot 1 = x$
$x + x' = 1$	$xx' = 0$
$x + y = y + x$	$xy = yx$
$x + (y+z) = (x+y) + z$	$x(yz) = (xy)z$
$x(y+z) = xy + xz$	$x + (yz) = (x+y)(x+z)$
$x + x = x$	$x \cdot x = x$
$x + 1 = 1$	$x \cdot 0 = 0$
$x + xy = x$	$x(x+y) = x$
$(x+y)' = x'y'$	$(xy)' = x' + y'$

Huntington postulater

(P0)	Mengden {0,1} er lukket under "+" og "•"		lukket
(P1)	$x + x = x$	$x \cdot x = x$	
(P2)	$x + 0 = x$	$x \cdot 1 = x$	ident.el.
(P2b)	$x + 1 = 1$	$x \cdot 0 = 0$	
(P5)	$x + x' = 1$	$x \cdot x' = 0$	komplem
(P6)	$0 \neq 1$		minst 2 el.
(P3)	$x + y = y + x$	$x \cdot y = y \cdot x$	kommutativ
(P4)	$x \cdot (y + z) = x \cdot y + x \cdot z$	$x+(y \cdot z) = (x+y) \cdot (x+z)$	distributiv
(P5)	$(x')' = x$		

Dualitet for postulatene:
Kan bytte "•" med + hvis man bytter "0" med "1"

Presedens:
Først utføres "0", så "'", så "•" og til slutt "+"

Sannhetstabell:

En bolsk funksjon kan visualiseres via en sannhetstabell. En gitt funksjon har *kun* en sannhetstabell, men en sannhetstabell har uendelig mange funksjonsuttrykk.

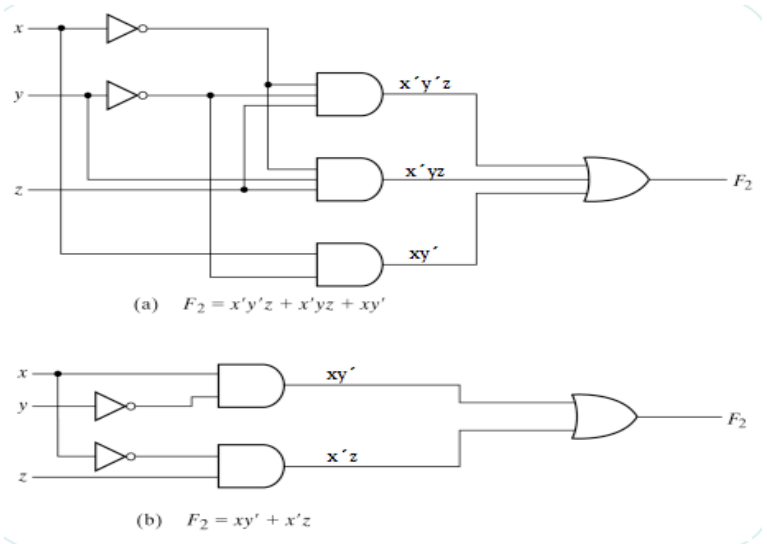
Eksempel:

$F = x + y'z$ →

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

A	B	C	D	X
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

	$C\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	1	0	0
$\bar{A}B$	0	0	1	1
AB	0	0	C	1
$A\bar{B}$	0	0	0	0



Forenkling av utrykk:
F = x´y´z + x´yz + xy´

Prosedyre:
F = x´z(y´+y) + xy
F = x´z • 1 + xy´
F = x´z + xy´

Flere eksempler:

Eks 1)
F = x (x' + y)
F = 0 + xy
F = xy

Eks 2)
F = x + x´y
F = (x + x´) (x+y)
F = 1 (x + y)
F = x + y

Eks 3)
F = (x + y) (x + y´)
F = x + (yy´)
F = x + 0
F = x

Eks 4)
F = xy + x´z + yz
F = xy + x´z ´yx (x + x´)
F = xy + x´z + xyz + x´yz
F = xy (1 + z) + x´z(1 + y)
F = xy + x'z

Eks 5)
F = (x + y)(x´ + z)(y + z)
F = (x + y)(x´+ z) **Dualitet**

Komplement av funksjon:

Eks 1)
F = x'yz' + x'y'z
F´ = (x'yz' + x'y'z)'
F´= (x'yz')'(x'y'z)'
F´ = (x + y' + z)(x + y + z')

Eks 2)
F = x(y'z'+yz)
F' = (x(y'z' + yz))'
F' = x' + (y'z' + yz)'
F' = x' + (y'z')'(yz)'
F' = x' + (y + z)(y' + z')

Minterm
I en funksjon kan en binær variabel x opptre som x eller x´
En funksjon kan være gitt på "**sum av produkt**" form. Mintermer har notasjon m_x

Eksempel:

F = xy + xy´ + x

Hvert "produktledd" som inneholder **alle** variablene kalles en **minterm**.
For to variable finnes det 4 forskjellige mintermer:

xy + xy´+ x´y + x´y´

For 3 variable finnes det 2^3 forskjellige mintermer.

Maksterm

En funksjon kan være gitt på "**produkt av sum**" form. Makstermer har notasjon M_x

Eksempel:

F = (x + y)(x + y´)y

Hvert "summeledd" som inneholder alle variablene kalles en **maksterm**
For to variable finnes det 4 forskjellige makstermer:

(x + y)(x + y´)(x´ + y)(x´ + y´)

For n variable finnes det 2^n forskjellige makstermer.

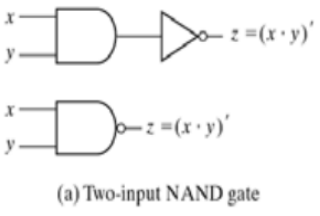
Sannhetstabell / mintermer
Hvis man generer en funksjon ut i fra sannhetstabellen får man en sum av mintermer.
Eksempel: F = x´y´z + xy´z´+ xyz´ →

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

En sannhetstabell kan sees på som en liste av mintermer.

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

NAND

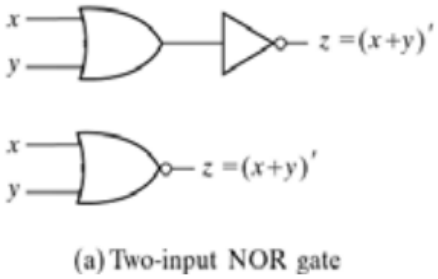


(a) Two-input NAND gate

NAND

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

NOR

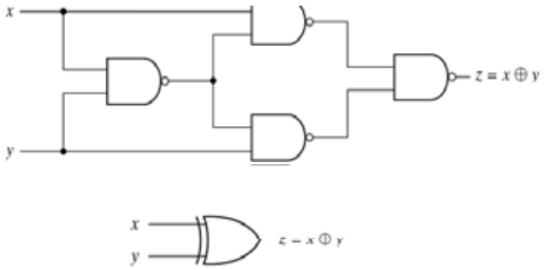


(a) Two-input NOR gate

NOR

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

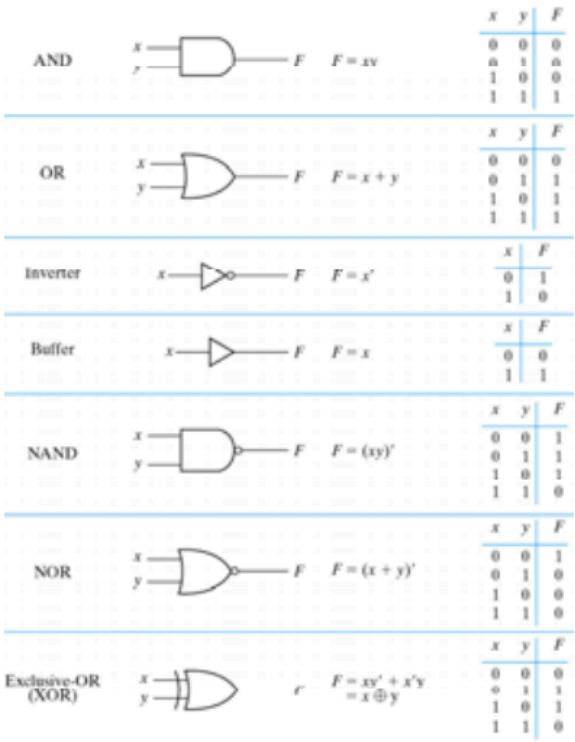
XOR



Exclusive-OR Implementations

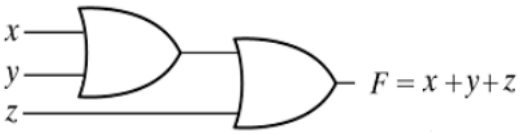
XOR

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



Flerinputs-porter

Flerinputs-implementasjon av AND/OR kan lages direkte:

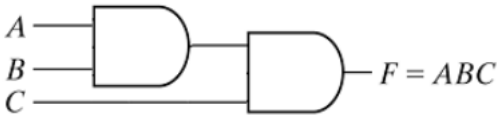


OR

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

AND

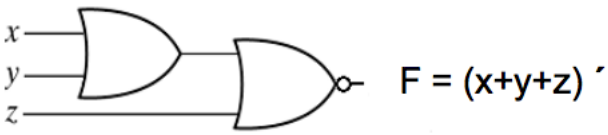
A B C	F
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1



Flerinputsimplementasjon av NAND / NOR kan ikke lages direkte

NOR

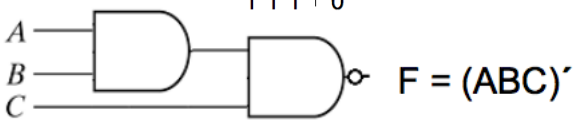
x y z	F
0 0 0	1
0 0 1	0
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	0



Mulig løsning

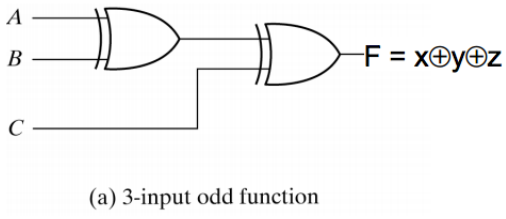
NAND

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



Mulig løsning

Flerinputs implementasjon av XOR (oddefunksjon) kan lages direkte



(a) 3-input odd function

XOR

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

Generell design prosedyre

1. Bestem hvilke signal som er innganger og utganger
2. Sett opp sannhetstabell for alle inngangskombinasjoner
3. Generer funksjonsuttrykket som sum av mintermer
4. Tilpass / forenkle funksjonsuttrykket mot aktuelle porter

Karnaughdiagram – uke 3

Grafisk metode for forenkling av Boolske uttrykk

- Uttrykket må være representert ved sum av mintermer (m_x). Disse leser vi direkte ut av sannhetstabellen
- Metoden egner seg for funksjoner med 2-4(5) variable

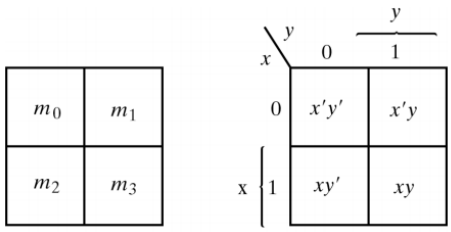
Eksempel, 2 variable:

$F = m_1 + m_3 = a´b + ab$

Eksempel, 4 variable:

$F = m_0 + m_1 + m_{15} = A´B´C´D´ + A´B´C´D + ABCD$

Oppsett av karnaugh med 2 variable

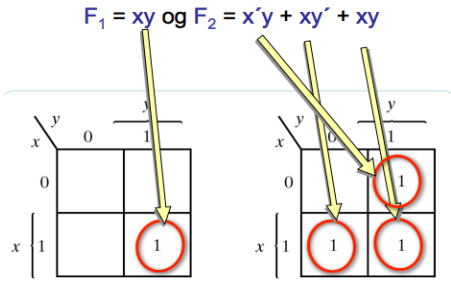


Setter inn mintermene i diagram

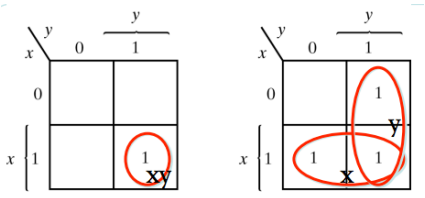
- Eksempel: generell funksjon – 2 variable
  - $F = m_0 + m_1 + m_2 + m_3$

Utlesing

- Grupperer naboruter som inneholder ”1” slik at vi får sammenhengende rektangler, Velg så store grupper som mulig. Antall element må være en potens av 2
- Representerer gruppene ved de variablene i gruppen som ikke varierer.



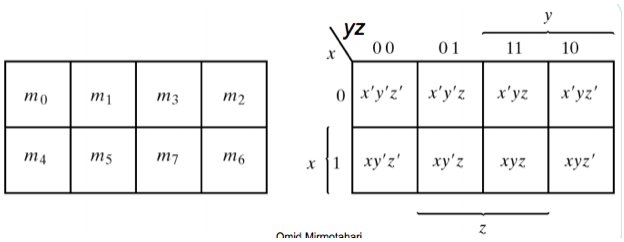
- Funksjonene som diagrammene beskriver er nå gitt av summen av uttrykkene som representerer hver gruppe.
  - $F_1 = xy$ $F_2 = x + y$



Karnaugh – 3 variable

Passering av mintermer for 3-variable funksjoner:

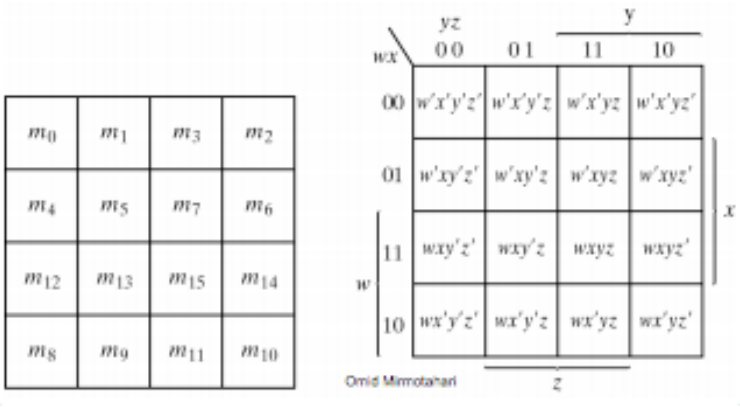
- Mintermene plasseres slik at *kun 1 variabel* varierer i mellom hver vannrette/loddrette naborute.



Karnaugh – 4 variable

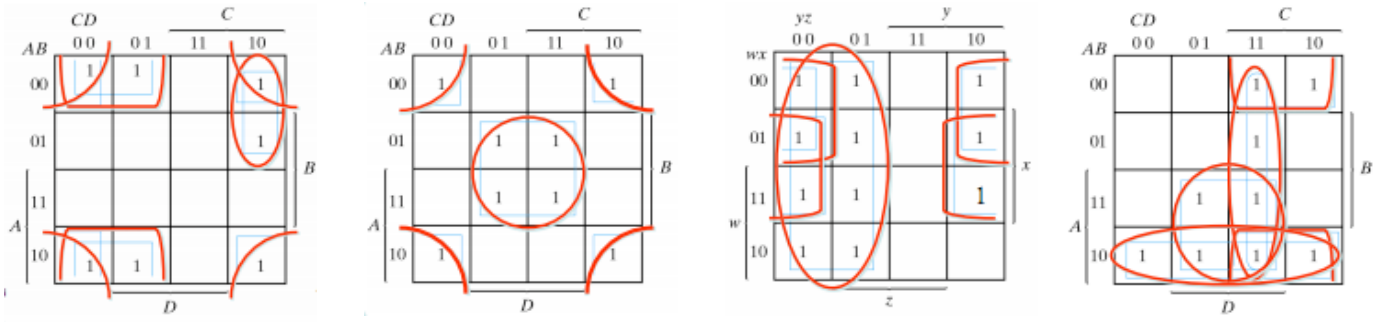
Plassering av mintermer for 4-variable funksjoner

- Mintermene plasseres slik at *kun 1 variabel* varierer i mellom hver vannrette/loddrette naborute.



Grupperingsregler for diagram med 2-4 variable

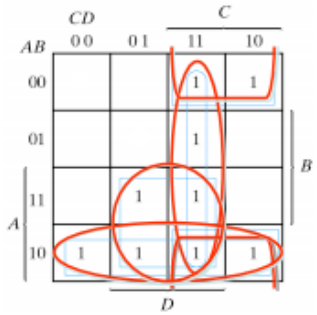
Grupperer naboruter som inneholder ”1” slik at vi får sammenhengende rektangler med 2^n form. Ytterkantene av diagrammet kan også være naboruter (→).



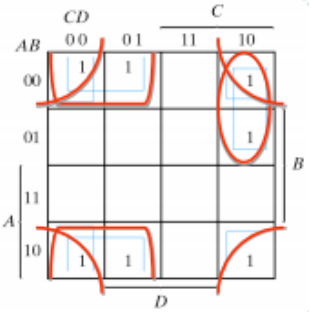
Utlesningssregler for diagram med 2-4 variable

Representerer hver gruppe ved de variablene i gruppen som ikke varierer. Diagrammets funksjon blir summen av hvert gruppeledd:

Eksempel – F =AD + CD + B´C + AB´

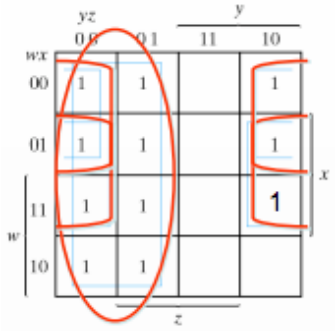


F = B´D´+ C´B´+ A´CD´

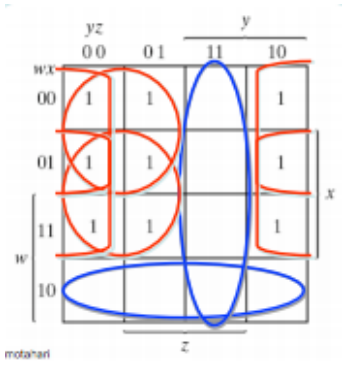


F = y´+ w´z´+ xz´

Merker oss at jo større ruter desto enklere uttrykk



Utlesning av ”0”ere
Ved å lese ut de tomme rutene (”0”erne) fra diagrammet får man F´
Dette kan noen ganger gi en enklere funksjon, eksempel:



F´= yz + wx´

F = (yz + wx´)´

Hadde vi lest ut ”1”erne ville vi fått_

F = xy´ + w´y´ + w´z´+ xz´

Designprosedyre:

1. Bestem innganger
2. Bestem utganger
3. Sett opp sannhetsverditabell
4. Finn mintermer
5. Sett inn i karnaughdiagram
6. Les ut av karnaughdiagram
7. Implementer uttrykkene

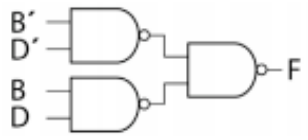
Algebraisk reduksjon til NAND/NOR

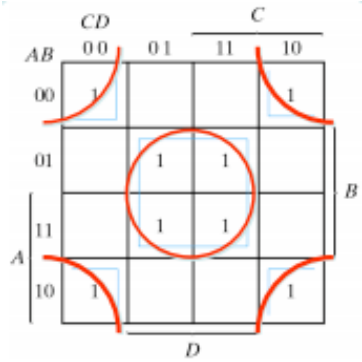
I noen tilfeller har man kun NAND og NOR kretser til rådighet
Eksempel:

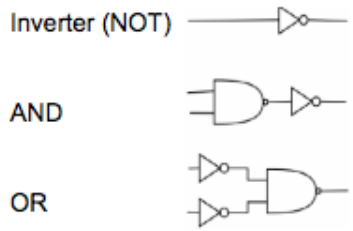
F = B´D´+ BD

DeMorgan:

F = ((B´D´)´ • (BD)´)´



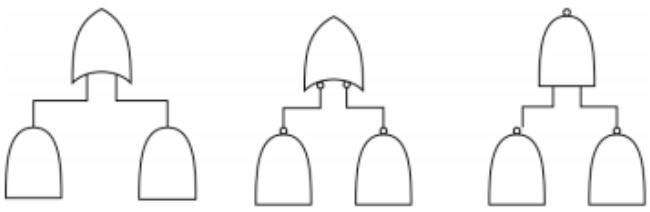




All kombinatorisk logikk kan implementeres ved hjelp av NAND

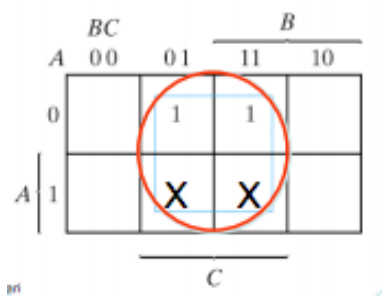
NAND reduksjon

- Tegn kretsen på sum av produkt form
- Bytt ut AND med NAND og OR med invert-OR
- Bytt ut invert-OR med NAND og sjekk alle inverteringer.



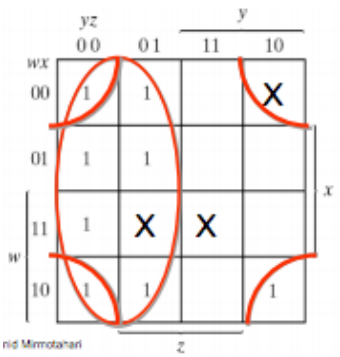
Don't care
I noen tilfeller har man inngangskombinasjoner som aldri dukker opp.
I andre tilfeller bryr man seg ikke om utfallet for visse inngangskombinasjoner.
Slike kombinasjoner kalls *don't care* kombinasjoner og markeres med "X"

"X"er kan man sette som man vil, enten "0" eller "1"
Eksempelet under velger de "X" til å være "1", og får en enklere funksjon, F = C



A	B	C	F	
0	0	0	0	
0	0	1	1	
0	1	0	0	
0	1	1	1	
1	0	0	0	
1	0	1	X	(1)
1	1	0	0	
1	1	1	X	(1)

Eks2)

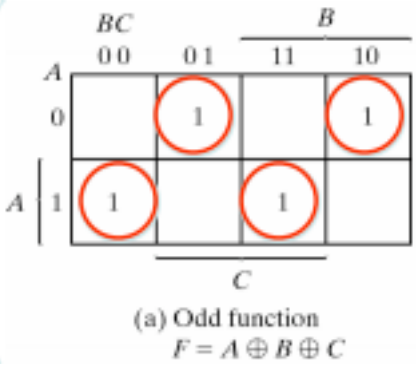


Velger to av "X"ene til "1" og en "X" til "0"

Får to grupper: F = y´+ x´z´

XOR
XOR funksjonen dekker maksimalt "uheldige" "1"er plasseringer i diagrammet

Har man XOR porter til rådighet bruker man disse
I dette eksemplet kan 1stk. 3-inputs XOR realisere F

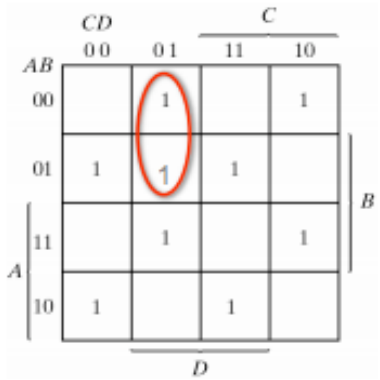


(a) Odd function
$F = A \oplus B \oplus C$

XOR funksjonen kan kombineres med andre ledd

Eksempel:

F = A⊕B⊕C⊕D + A´C´D



Uke 4 – Kombinatorisk Logikk

Binær addisjon:
Prosedyren for binær addisjon er identisk med prosedyren for desimal addisjon.

Eks: adder 5 og 13:

```
  1 1   1          0 5
  0 0 1 0 1       +1 3
+ 0 1 1 0 1       ----
= 1 0 0 1 0       =1 8
```

7	0 1 1 1
6	0 1 1 0
5	0 1 0 1
4	0 1 0 0
3	0 0 1 1
2	0 0 1 0
1	0 0 0 1
0	0 0 0 0
-1	1 1 1 1
-2	1 1 1 0
-3	1 1 0 1
-4	1 1 0 0
-5	1 0 1 1
-6	1 0 1 0
-7	1 0 0 1
-8	1 0 0 0

Negative binære tall:
Mest vanlig representasjon: 2'er komplement
Lar mest signifikante bit være 1 for negative tall
Dette må være "avtalt" på forhånd

Eks: 4 bit kan representere tallene -8 til +7

2'er komplement
Setter minus foran et binært tall ved å invertere alle bittene og plusse på 1

Eks:
finner –5:
invertert 5:
-5:

```
  1 0 1 0
+ 0 0 0 1
= 1 0 1 1
```

7	0 1 1 1
6	0 1 1 0
5	0 1 0 1
4	0 1 0 0
3	0 0 1 1
2	0 0 1 0
1	0 0 0 1
0	0 0 0 0
-1	1 1 1 1
-2	1 1 1 0
-3	1 1 0 1
-4	1 1 0 0
-5	1 0 1 1
-6	1 0 1 0
-7	1 0 0 1
-8	1 0 0 0

Binær subtraksjon
Fremgangsmåte for tall representert ved 2'er komplement:
Adder tallene på vanlig måte.

```
    0 0 1 1          3
+   1 0 0 0      + -8
=   1 0 1 1      = -5
```

Betyr negativt tall

Eksempel:

```
    1 1
    0 1 1 0          6
+   1 1 1 0      + -2
= (1) 0 1 0 0    =  4
```

Går ut

Betyr positivt tall

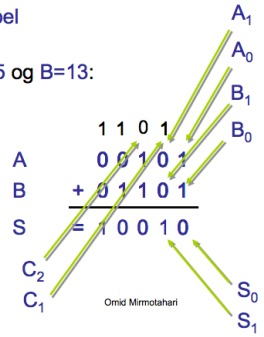
Binær adder
En av de mest brukte digitale kretser. Vanlige anvendelser: Mikroprosesser ALU / Xbox / miksebord / digitalt kommunikasjonsutstyr / AD-DA omformere osv.

- Basis for addisjon / subtraksjon / multiplikasjon / divisjon og mange andre matematiske operasjoner
- All form for filtrering / signalbehandling

Ønsker å designe en generell binær adder

Funksjonelt eksempel

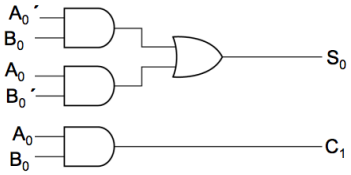
Adder to tall A=5 og B=13:



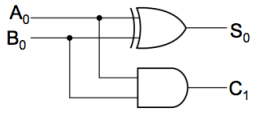
Halvadder (ingen mente inn)
Adderer sammen de to minst signifikante bittene A_0 og B_0.
Elementet har 2 innganger og 2 utganger.
$S_0 = A_0´B_0 + A_0B_0´ ? A_0 \oplus B_0$
$C_1 = A_0B_0$

Sannhetstabell

A_0	B_0	S_0	C_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$S_0 = A_0´B_0 + A_0B_0´$
$C_1 = A_0B_0$



$S_0 = A_0 \oplus B_0$
$C_1 = A_0B_0$

Fulladder (mente inn)
Adderer sammen bit A_n, B_n med eventuelt *mente* inn.

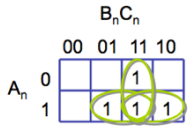
Sannhetstabell

A_n	B_n	C_n	S_nC_{n+1}
0	0	0	0 0
0	0	1	1 0
0	1	0	1 0
0	1	1	0 1
1	0	0	1 0
1	0	1	0 1
1	1	0	0 1
1	1	1	1 1

Elementet har 3 innganger og 2 utganger

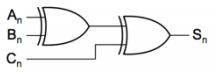
$S_n = A_n \oplus B_n \oplus C_n$ (oddefunksjon)
$C_{n+1} = A_n´B_nC_n + A_nB_n´C_n + A_nB_nC_n´ + A_nB_nC_n$

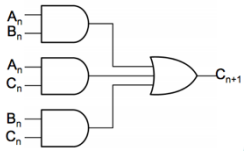
<u>Forenkling</u>
Forenkler C_{n+1} ved Karnaughdiagram.



$C_{n+1} = A_n´B_nC_n + A_nB_n´C_n + A_nB_nC_n´ + A_nB_nC_n$
$C_{n+1} = A_nB_n + A_nC_n + B_nC_n$

<u>Implementasjon I</u>
Rett fram implementasjon

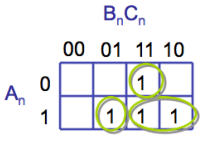




$S_n = A_n \oplus B_n \oplus C_n$
$C_{n+1} = A_nB_n + A_nC_n + B_nC_n$

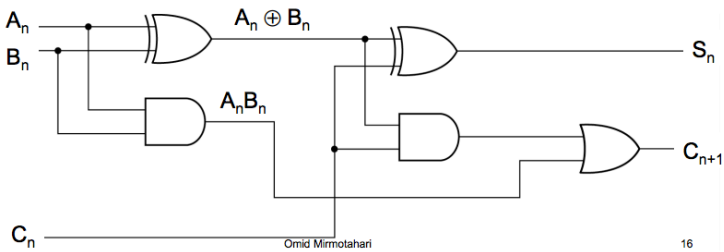
<u>Implementasjon II</u>
Forenklet implementasjon av C_{n+1} basert på gjenbruk av porter fra S_n
Leser ut C_{n+1} fra karnaughdiagram på nytt

$S_n = (A_n \oplus B_n) \oplus C_n$

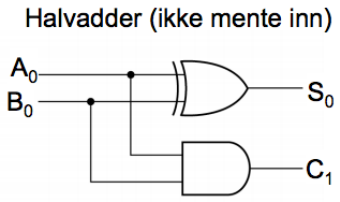


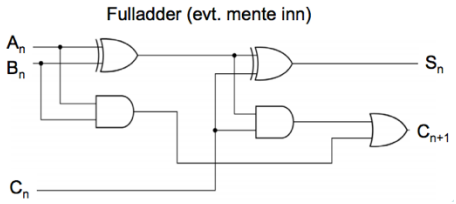
$C_{n+1} = A_nB_n + A_nB_n´C_n + A_n´B_nC_n$
$C_{n+1} = A_nB_n + (A_nB_n´ + A_n´B_n)\,C_n$
$C_{n+1} = A_nB_n + (A_n \oplus B_n)\,C_n$

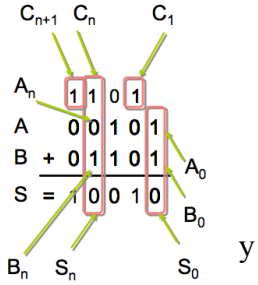
Vanlig implementasjon av en-bits fulladder
$S_n = (A_n \oplus B_n) \oplus C_n$
$C_{n+1} = A_nB_n + (A_n \oplus B_n)\,C_n$



Binær adder







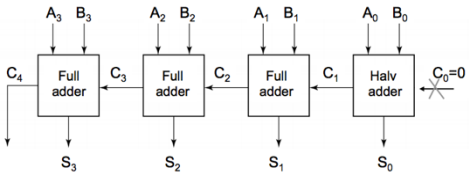
y

Et adder system

Systemelementer:

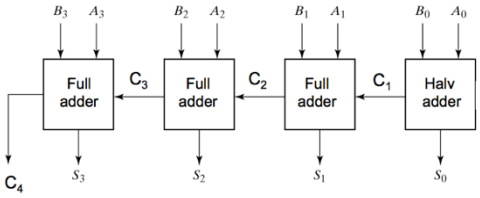
Halvadder: Tar ikke mente inn

Fulladder: Tar mente inn



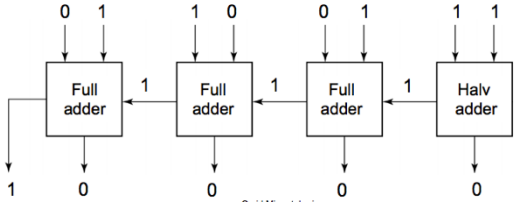
Menteforplantning

4-bits binær adder



Portforsinkelse gir menteforplantning (rippeladder)

Eksempel: Adderer 0101 og 1011

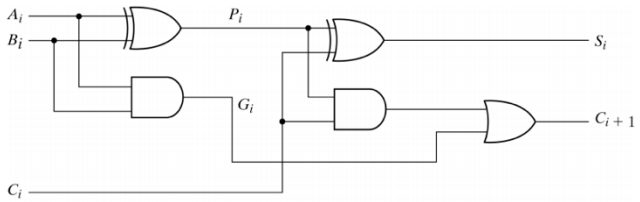


"Carry Lookahead"

Ønsker å unngå menteforplantning – gir økt hastighet

G_i – generate: brukes i menteforplantningen

P_i – propagate: påvirker menteforplantningen



$S_i = P_i \oplus C_i$

$C_{i+1} = G_i + P_i C_i$

For en 4-bits adder bestående av 4 fulladdertrinn har vi:

$S_i = P_i \oplus C_i$ $\quad\quad$ $C_{i+1} = G_i + P_i C_i$

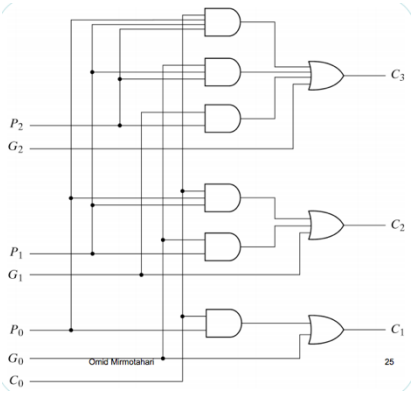
Uttrykker C_1, C_2 og C_3 rekursivt

$C_1 = G_0 + P_0 C_0$

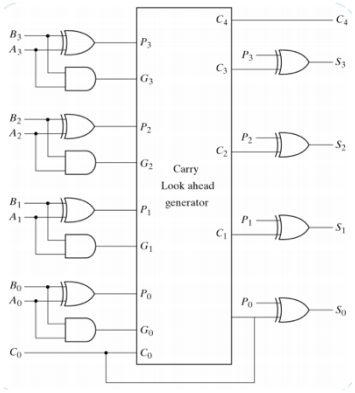
$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

Rett fram implementasjon av C_1, C_2, C_3

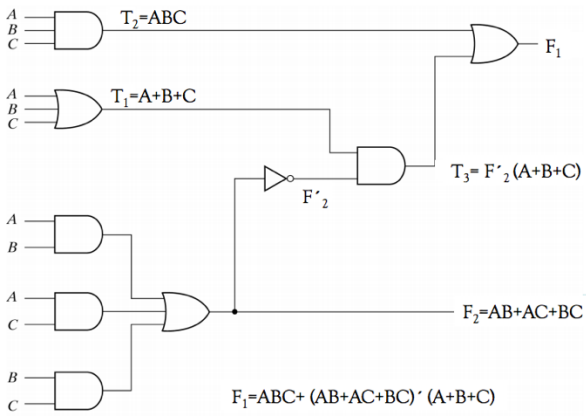


4-bits Carry Lookahead adder med input carry C_0



Generell analyseprosedyre for digitale kretser

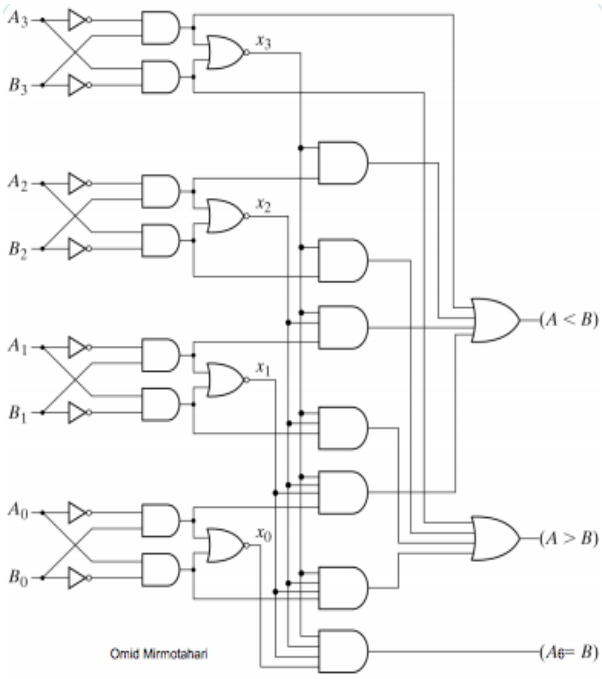
1) Sett funksjonsnavn på ledningene
2) Finn funksjonene
3) Kombiner funksjonsuttrykkene



Uke 5 – Kombinatorisk logikk (elementer)

Komparator – sammenligner to tall A og B

- 3 utganger: A=B, A>B og A<B



Eksempel: 4-bits komparator

Utgang A=B

Slår til hvis $A_0=B_0$ og $A_1=B_1$ og $A_2=B_2$ og $A_3=B_3$

Kan skrives: $(A_0 \oplus B_0)'(A_1 \oplus B_1)'(A_2 \oplus B_2)'(A_3 \oplus B_3)'$

Utgang A>B slår til hvis:

($A_3>B_3$) eller

($A_2>B_2$ og $A_3=B_3$) eller

($A_1>B_1$ og $A_2=B_2$ og $A_3=B_3$) eller

($A_0>B_0$ og $A_1=B_1$ og $A_2=B_2$ og $A_3=B_3$)

Kan skrives: $(A_3B_3') + (A_2B_2')(A_3 \oplus B_3)' + (A_1B_1')(A_2 \oplus B_2)'(A_3 \oplus B_3)' + (A_0B_0')(A_1 \oplus B_1)'(A_2 \oplus B_2)'(A_3 \oplus B_3)'$

Utgang A<B slår til hvis:

($A_3<B_3$) eller

($A_2<B_2$ og $A_3=B_3$) eller

($A_1<B_1$ og $A_2=B_2$ og $A_3=B_3$) eller

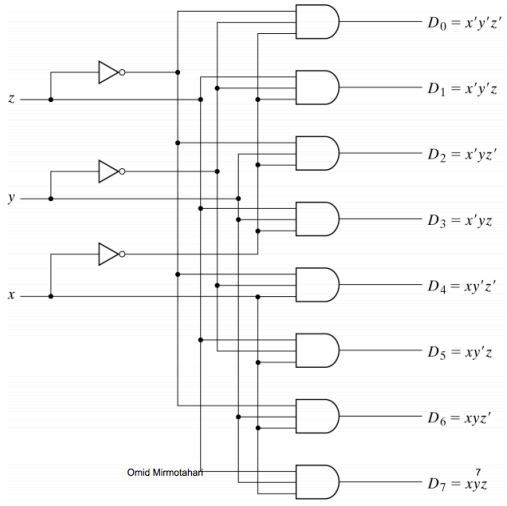
($A_0<B_0$ og $A_1=B_1$ og $A_2=B_2$ og $A_3=B_3$)

Kan skrives: $(A_3'B_3) + (A_2'B_2)(A_3 \oplus B_3)' + (A_1'B_1)(A_2 \oplus B_2)'(A_3 \oplus B_3)' + (A_0'B_0)(A_1 \oplus B_1)'(A_2 \oplus B_2)'(A_3 \oplus B_3)'$

Dekoder – tar inn et binært ord, gir ut alle mintermer

Eks: 3bit inn → 8 bit ut

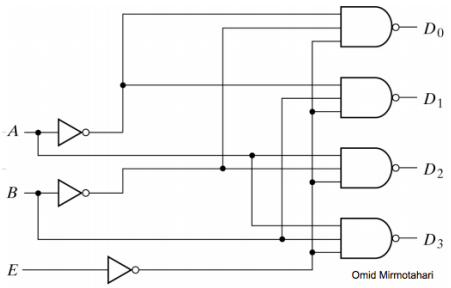
Innganger			Utganger							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Dekoder – **varianter**

Enable input: Enable aktiv - normal operasjon.
Enable inaktiv - alle utganger disablet

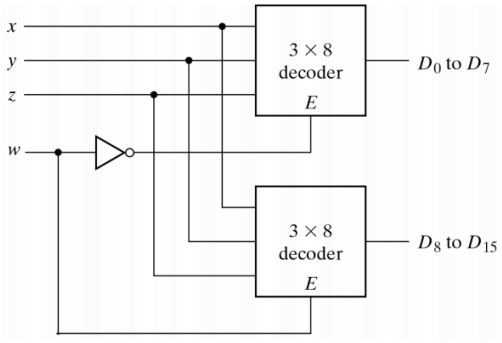
NAND logikk: Inverterte utganger



E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Eksempel

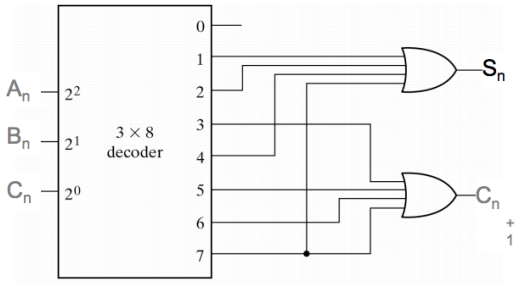
Aktiv "lav" enable inngang



Parallellkobling

Eksempel: Lager en 4x16 dekoder fra 2stk 3x8 dekodere med enable innganger

Eksempel:
Fulladder



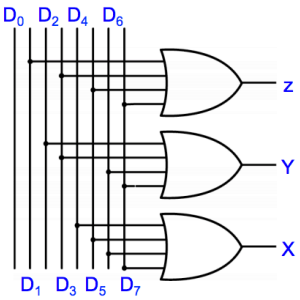
Generering av logiske funksjoner

Dekoder - elektrisk sannhetstabell.
Kan generere generelle logiske funksjoner direkte fra mintermene på utgangen

Enkoder – motsatt av dekoder

Eksempel

$x = D_4 + D_5 + D_6 + D_7$
$y = D_2 + D_3 + D_6 + D_7$
$z = D_1 + D_3 + D_5 + D_7$



Eksempel: 8x3 enkoder

Innganger								Utganger		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$x = D_4 + D_5 + D_6 + D_7$
$y = D_2 + D_3 + D_6 + D_7$
$z = D_1 + D_3 + D_5 + D_7$

Antar at det ikke eksisterer andre inngangskombinasjoner

Prioritets-enkoder

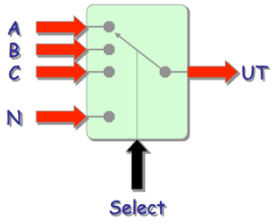
Problem i enkodere: Hva hvis man får flere "1"ere inn samtidig?
Løsning: Prioritets-enkoder

Hvis flere "1"ere inn - ser kun på inngang med høyst indeks (prioritet)
Eks: 8x3 prioritets-enkoder

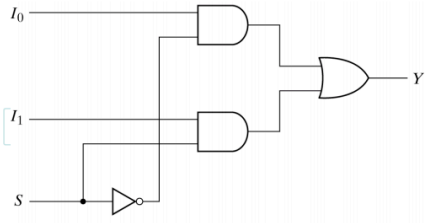
Innganger								Utganger		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
x	1	0	0	0	0	0	0	0	0	1
x	x	1	0	0	0	0	0	0	1	0
x	x	x	1	0	0	0	0	0	1	1
x	x	x	x	1	0	0	0	1	0	0
x	x	x	x	x	1	0	0	1	0	1
x	x	x	x	x	x	1	0	1	1	0
x	x	x	x	x	x	x	1	1	1	1

Multiplekser

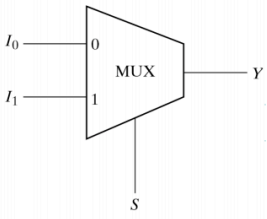
Multiplekser (MUX) – velger hvilke innganger som slippes ut
Hver inngang kan bestå av ett eller flere bit



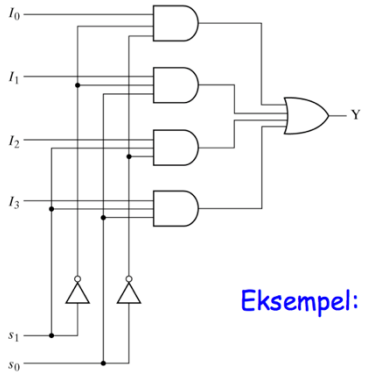
Eksempel: 2-1 MUX



Implementasjon



Symbol

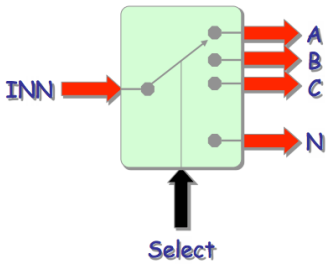


s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

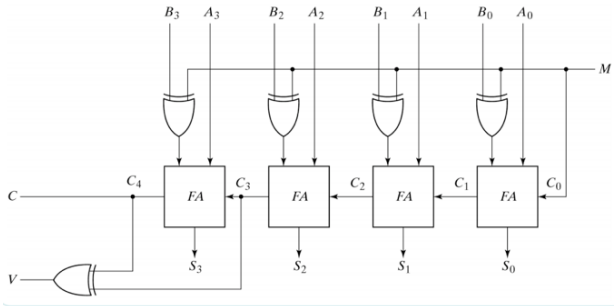
(b) Function table

Eksempel: 4-1 MUX

Demultiplekser – motsatt av multiplekser



Kombinert adder/subtraktor

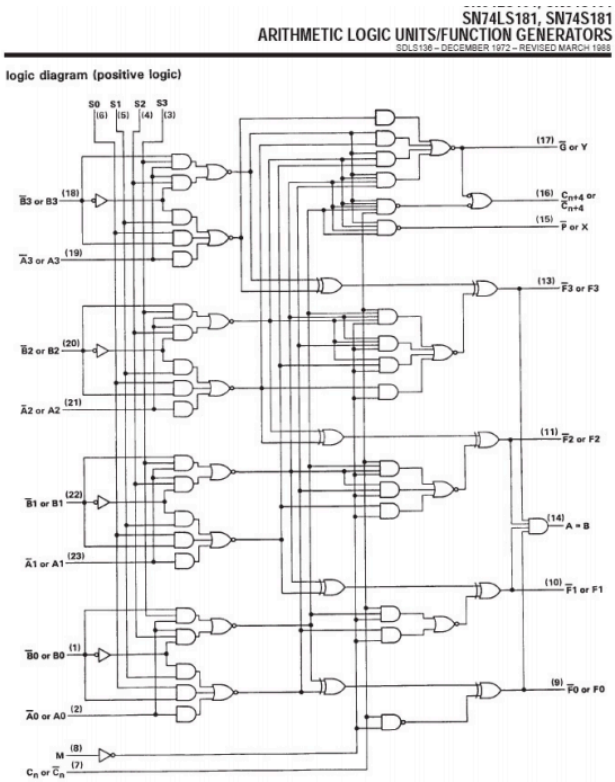


M=0: adder / M=1: subtraktor / V: overflow bit

ALU – Arithmetic Logic Unit
Generell regneenhet

Eksempel: SN74LS181, 4bit utbyggbar ALU 30 forskjellige operasjoner

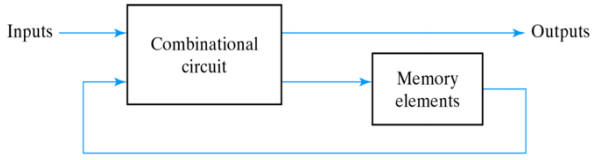
SELECTION				ACTIVE-HIGH DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	$\overline{C}_n$ = H (no carry)	$\overline{C}_n$ = L (with carry)
L	L	L	L	$F = \overline{A}$	F = A	F = A PLUS 1
L	L	L	H	$F = \overline{A+B}$	F = A + B	F = (A + B) PLUS 1
L	L	H	L	$F = \overline{A}B$	$F = A + \overline{B}$	$F = (A + \overline{B})$ PLUS 1
L	L	H	H	F = 0	F = MINUS 1 (2's COMPL)	F = ZERO
L	H	L	L	$F = \overline{AB}$	F = A PLUS AB	$F = A$ PLUS $A\overline{B}$ PLUS 1
L	H	L	H	$F = \overline{B}$	F = (A + B) PLUS AB	$F = (A + B)$ PLUS $A\overline{B}$ PLUS 1
L	H	H	L	$F = A \oplus B$	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	$F = A\overline{B}$	$F = A\overline{B}$ MINUS 1	$F = A\overline{B}$
H	L	L	L	$F = \overline{A} + B$	F = A PLUS AB	F = A PLUS AB PLUS 1
H	L	L	H	$F = \overline{A \oplus B}$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	$F = (A + \overline{B})$ PLUS AB	$F = (A + \overline{B})$ PLUS AB PLUS 1
H	L	H	H	F = AB	F = AB MINUS 1	F = AB
H	H	L	L	F = 1	F = A PLUS A†	F = A PLUS A PLUS 1
H	H	L	H	$F = A + \overline{B}$	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
H	H	H	L	F = A + B	$F = (A + \overline{B})$ PLUS A	$F = (A + \overline{B})$ PLUS A PLUS 1
H	H	H	H	F = A	F = A MINUS 1	F = A

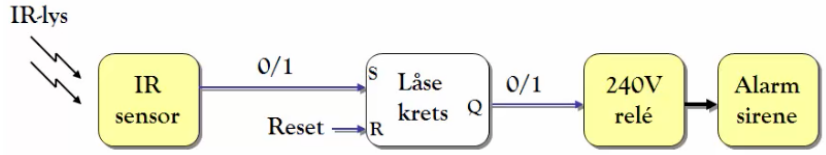


Uke 7 – Sekvensiell logikk (del 1)

Kombinatorisk logikk gitt ved at utgangsverdiene er entydig gitt av nåværende kombinasjon av inngangssignaler.

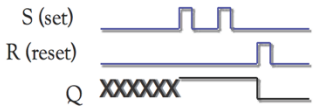
Sekvensiell logikk, inneholder hukommelse (låsekretser). Utgangsverdiene er gitt av nåværende inngangssignaler, samt sekvensen av tidligere inngangssignaler/utgangsverdier.

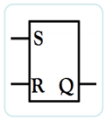




SR-latch – funksjonell beskrivelse

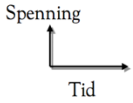
1) Kretsen skal sette Q til ”1” hvis den får ”1” på inngang S. Når inngang S går tilbake til ”0” skal Q forbli på ”1”





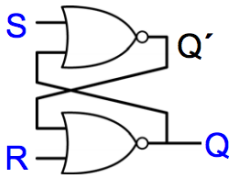
SR		Q
0	0	låst
0	1	0
1	0	1
1	1	0

2) Kretsen skal resette Q til ”0” når den får ”1” på inngang R. Når inngang R går tilbake til ”0” skal Q forbli på ”0”



3) Tilstanden ”1” på både S og R brukes normalt ikke

SR-latch portimplementasjon NOR



Øvre NOR

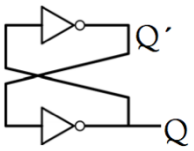
S	Q	Q´
0	0	1
0	1	0
1	0	0
1	1	0

Nedre NOR

Q´	R	Q
0	0	1
0	1	0
1	0	0
1	1	0

*Signalet Q´ er ikke invertert av Q for tilstand S=1, R=1

Tilstand S=0, R=0: En NOR port med fast ”0” inn på en av inngangene er ekvivalent med NOT



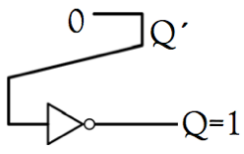
Øvre NOR

S	Q	Q´
0	0	1
0	1	0
1	0	0
1	1	0

Nedre NOR

Q´	R	Q
0	0	1
0	1	0
1	0	0
1	1	0

Tilstand S=1, R=0: En NOR port med fast ”1” inn på en av inngangene gir alltid ut ”0”



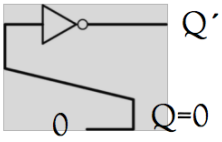
Øvre NOR

S	Q	Q´
0	0	1
0	1	0
1	0	0
1	1	0

Nedre NOR

Q´	R	Q
0	0	1
0	1	0
1	0	0
1	1	0

Tilstand S=0, R=1: En NOR port med fast ”1” inn på en av inngangene gir alltid ut ”0”



Øvre NOR

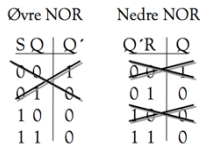
S	Q	Q´
0	0	1
0	1	0
1	0	0
1	1	0

Nedre NOR

Q´	R	Q
0	0	1
0	1	0
1	0	0
1	1	0

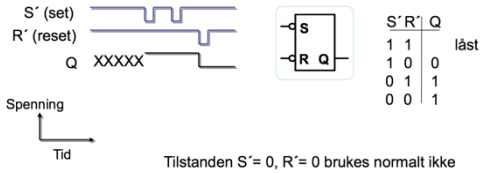
Tilstand S=1, R=1: En NOR port med fast ”1” inn på en av inngangene gir alltid ut ”0”

I denne tilstanden er utgang Q´ ikke den inverterte av Q. Denne tilstanden brukes normalt ikke.



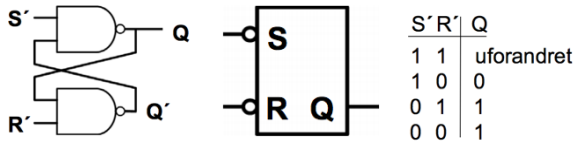
Tilstand S=0, R=0 gir ? (**låst**)
Tilstand S=0, R=1 gir 0 (**0**)
Tilstand S=1, R=0 gir 1 (**1**)
Tilstand S=1, R=1 gir 0 (**0**)

S´R´ latch – lik funksjon som SR latch, men reagerer på “0” inn (negativ logikk)



Tilstanden S´= 0, R´= 0 brukes normalt ikke

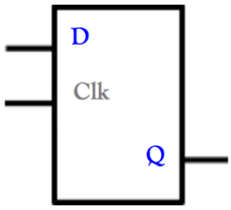
S´R´Latch – **Portimplementasjon NAND**
Kretsen kan analyserer på samme måte som for SR latch



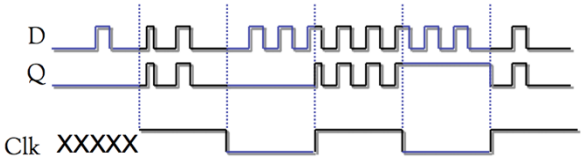
D-latch
Dataflyten gjennom en D-latch kontrolleres av et klokkesignal

1) Slipper gjennom et digital signal så lenge klokkeinngangen er ”1” (transparent)

2) I det øyeblikket klokkeinngangen går fra ”1” til ”0” låser utgangen seg på sin nåværende verdi. Forandringer på inngangen vil ikke påvirke utgangsverdien så lenge klokkesignalet er ”0”.

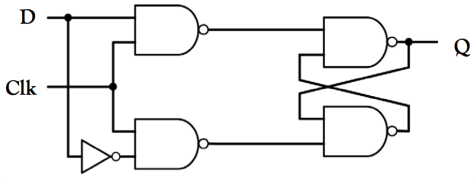


Clk = 1 : kretsen slipper gjennom signalet
Clk = 0 : kretsen holder (låser) utgangssignalet

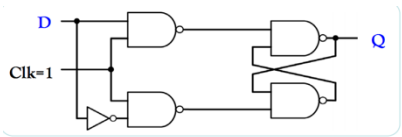


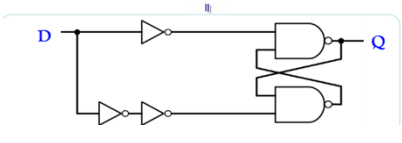
Logisk verdi på D i det øyeblikk Clk går i fra ”1” til ”0” bestemmer verdien som holdes på Q

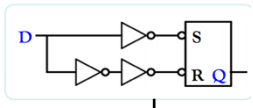
D-latch – implementasjon



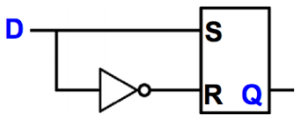
D-latch – analyse:







Tilstand 1: Clk = 1
NAND med fast “1” på en inngang er ekvivalent med NOT

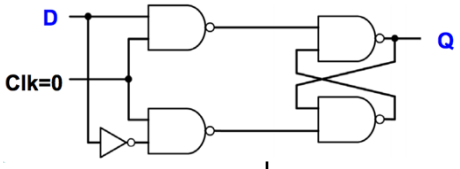


Dette er en S´R´ latch med inverterte innganger
Tilstand S´= R´ oppstår aldri, og kretsen er ekvivalent med en S´´R´´ eller SR latch.

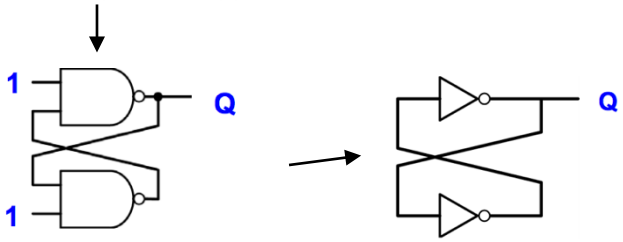
S og R vil alltid være forskjellige
Fra sannhetstabellen ser vi at Q=S=D, kretsen slipper signal D rett gjennom (transparent).

SR	Q	
~~0 0~~	~~Q~~	uforandret
0 1	0	
1 0	1	
~~1 1~~	~~0~~	

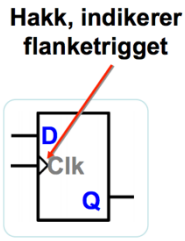
Tilstand 2: Clk = 0
NAND med ”0” på en inngang gir alltid ut ”1”.



NAND med ”1” på en inngang er ekvivalent med NOT. Får stabil låsekrets. Utgang holdes.

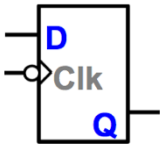


Flip-Flop



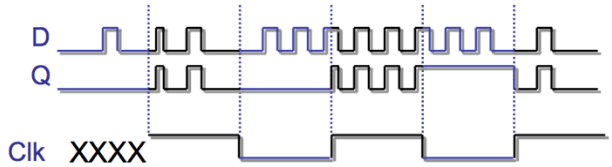
Flip-Flop’er kommer i to varianter:
- Positiv flanketrigget
- Negativ flanketrigget

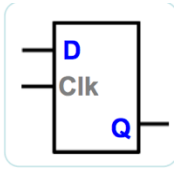
På en positiv flanketrigget Flip-Flop kan utgangen kun skifte verdi i det øyeblikk klokkesignalet går fra ”0” til ”1”.



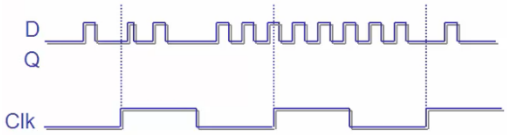
På en negativ flanketrigget Flip-Flop kan utgangen kun skifte verdi i det øyeblikk klokkesignalet går fra ”1” til ”0”.

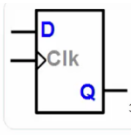
D-Flip-Flop
En D latch er transparent for Clk=1



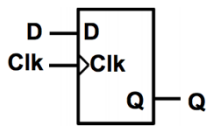


En positiv flanketrigget D flip-flop sampler verdien på D i det øyeblikk Clk går fra ”0” til ”1” (positiv flanke). Denne verdien holdes fast på utgangen helt til neste positive flanke.





Karakteristisk tabell/ligning
For flip-flop’er kan man generelt beskrive neste utgangsverdi Q(t+1) som funksjon av nåværende inngangsverdi(er), og nåværende utgangsverdi Q(t)



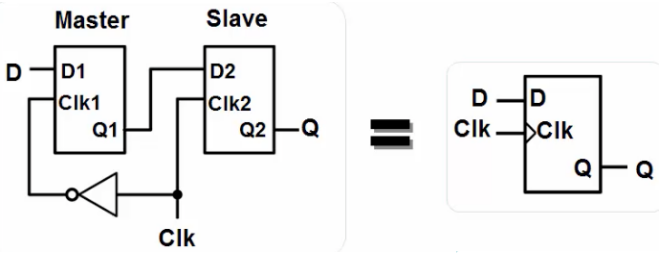
Karakteristisk tabell for D flip-flop

D	Q(t+1)
0	0
1	1

Karakteristisk ligning for D flip-flop

$$Q(t+1) = D$$

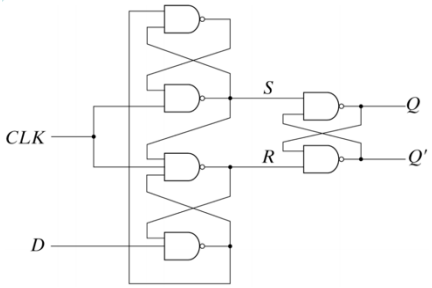
En positiv flanketrigget D flip-flop kan velges av to D-latcher (Master-Slave)



Under Clk=0 er første D latch (master) transparent

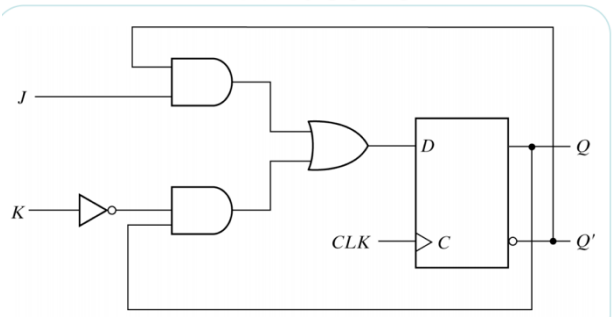
Under Clk=1 er siste D latch (slave) transparent

D-Flip-Flop – kompakt versjon

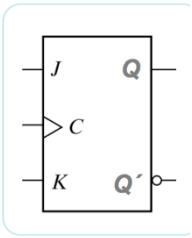


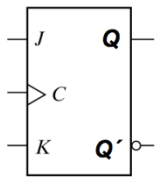
JK Flip-Flop

Kretsoppbygging



Grafisk symbol





En JK flip-flop har følgende egenskaper:

J=0, K=0: Utgang låst

J=0, K=0: Resetter utgang til ”0”

J=1, K=0: Setter utgang til ”1”

J=1, K=1: Inverterer utgang Q→Q´

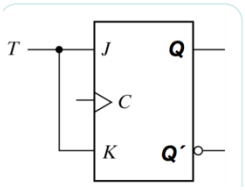
J K	Q(t+1)
0 0	Q(t)
0 1	0
1 0	1
1 1	Q´(t)

$$Q(t+1) = JQ'(t) + K'Q(t)$$

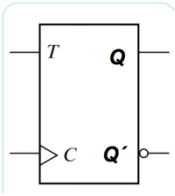
Utgangen kan kun forandre verdi på stigende klokkeflanke. En JK flip-flop er den mest generelle flip-floppen vi har.

T Flip-Flop

Kretsoppbygging

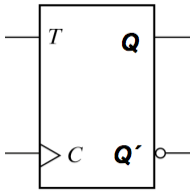


Grafisk symbol



T	Q(t+1)
0	Q(t)
1	Q´(t)

$Q(t+1) = T \oplus Q(t)$



En T flip-flop har følgende egenskaper

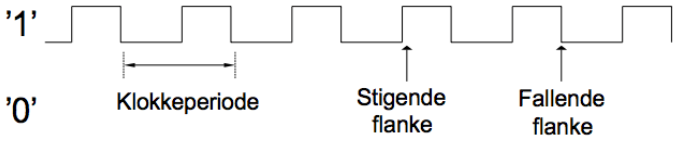
T=0, Utgang låst

T=1, Inverterer utgang Q→Q´

Utgangen kan kun forandre verdi på stigende klokkeflanke. Det er lett å lage tellere av T flip-flop'er.

Uke 8 – Sekvensiell logikk (del 2)

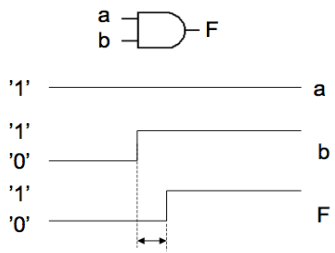
- I synkrone sekvensielle kretser skjer endringen(e) i output samtidig med endringen i et klokkesignal.
- I asynkrone sekvensielle kretser skjer endringen(e) i output uten noe klokkesignal.
- Nesten alle kretser er synkrone.



- Et klokkesignal er et digitalt signal som veksler mellom ´0´ og ´1´ med fast takt.
- Den omvendte av klokkeperioden kalles (klokke)frekvensen, altså
- Ønsker så høy klokkefrekvens som mulig, fordi hver enkelt operasjon da bruker så kort tid som mulig.
- Maksimal klokkefrekvens bestemmes av flere faktorer, blant annet:
  - Lengde på signalveiene
  - Last
  - Forsinkelse gjennom porter (delay)
  - Teknologi
- NB: Hastighet er ikke direkte proporsjonal med klokkefrekvens.

$$frekvens = \frac{1}{klokkeperioden}$$

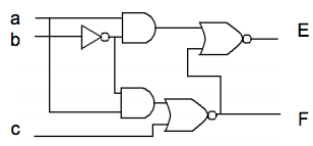
Portforsinkelse / tidsforsinkelse



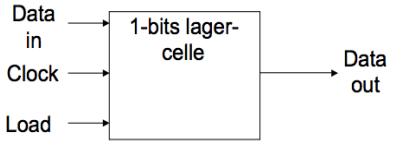
Logisk dybde

Logisk dybde: Antall porter et signal passerer fra inngang til utgang.
Ved å redusere logisk dybde reduseres forsinkelsen gjennom kretsen.

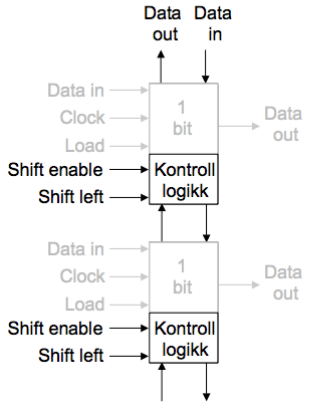
Eksempel:



Internt i en CPU trengs en fleksibel type lagercelle som kan lagre et bit.
Som regel trenger man å lagre hele byte, halvord eller ord, og gjøre samme operasjon på alle bitene.
Load-signalet bestemmer om ny verdi skal lastes inn eller ikke.
Ved å sette sammen 1-bits celler i parallell, får man et register.
Hvis man i tillegg kan laste data over i nabocellen, kalles det et skiftregister.



Shiftregister - enkel

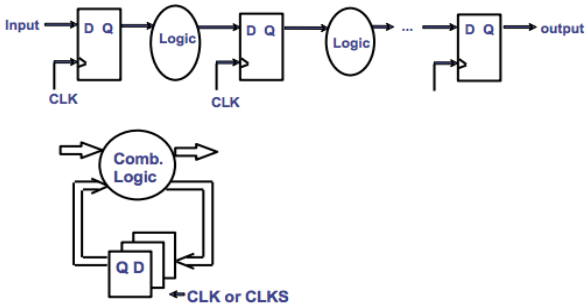


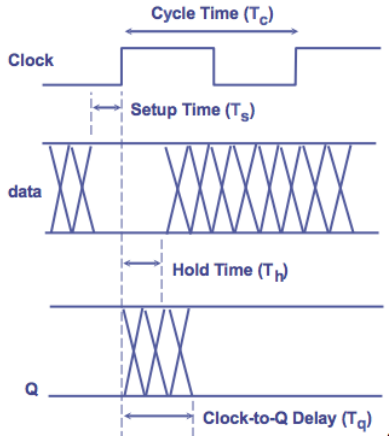
Load	Shift Enable	Shift Left	
'0'	'X'	'X'	Ingen endring
'1'	'0'	'X'	Data out := Data in
'1'	'1'	'0'	Shift right
'1'	'1'	'1'	Shift left

Omid Mirmotahari 8

System (klokket)

De fleste VLSI systemer er en kombinasjon av:
(a) Pipeline
(b) tilstandsmaskiner (FSM)





1-fase klokking

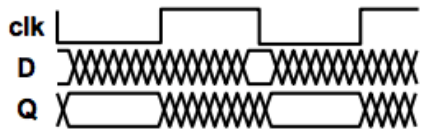
Tc = klokkeperioden

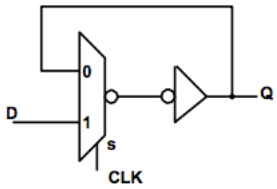
Ts = tiden før klokkeflanken hvor inngangen må være stabil og tilgjengelig

Th = tiden etter klokkeflanken hvor inngangssignalet må fortsatt være stabil

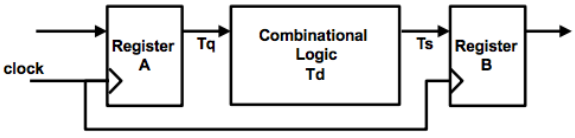
Tq = tiden det tar fra klokkeflanken til utgangen er klar

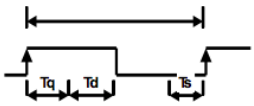
1-fase klokking *eksempel*





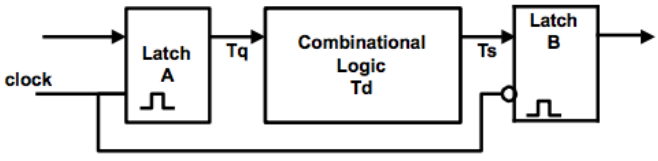
System timing

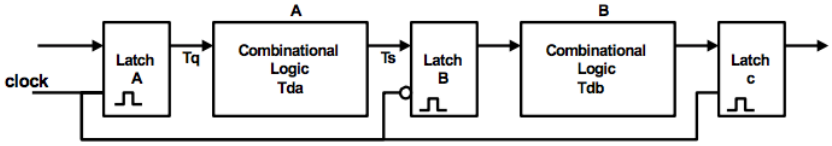


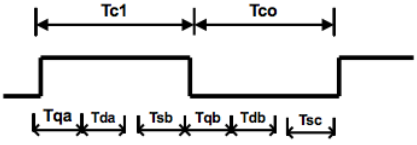


Tc >Tq + Td + Ts

Alternativt, kan man bruke latcher som lagrings element for å spare plass

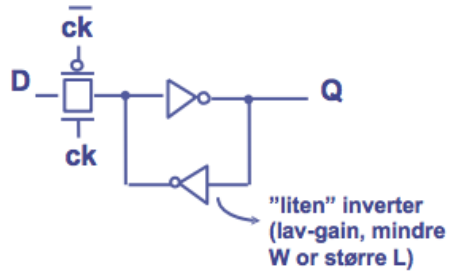


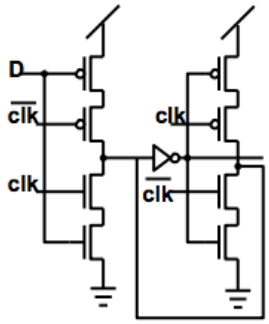




Tc1>Tqa+Tda+Tsb Tco>Tqb+Tdb+Tsc
If Tc=Tc1+Tco and Tc1=Tco, Tqa=Tqb, Tsb=Tsc
=> The limit is Tc = Tda + Tdb + 2(Tq+Ts)

Klokket latch i CMOS





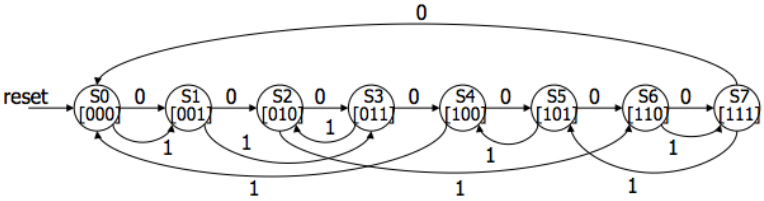
Tilstandsmaskin - Teller

- Synkron 3-bits teller har en modus signal M
  - M = 0, teller oppover i binær sekvens
  - M = 1, teller oppover i Gray Code sekvens

Binært: 000, 001, 010, 011, 100, 101, 110, 111
Gray: 000, 001, 011, 010, 110, 111, 101, 100

Et utvalg inputkombinasjoner

Input M	Nåværende tilstand	Neste tilstand
0	000	001
0	001	010
1	010	110
1	110	111
1	111	101
0	101	110
0	110	111



Uke 9 – Tilstandsmaskin

Engelsk: Finite State Machine

Tilstandsmaskiner er en metode å beskrive systemer med logisk og dynamisk (tidsmessig) oppførsel.

Brukes mye innen:

- Logiske/digitale styresystemer
- Sanntidssystemer
- Telekommunikasjon
- Kompilatorteknikk
- Digitalteknikk

Modellen av tilstandsmaskin består av:

- En rekke tilstander
- Hendelser som endrer systemet fra en tilstand til en annen
- Aksjoner som er et resultat av hendelser

En **tilstandsmaskin** er et **sekvensielt system** som gjennomløper et et sett med tilstander styrt av verdiene på inngangssignalene.

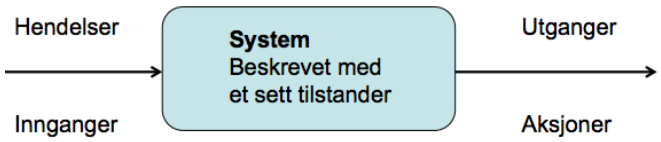
Tilstanden systemet befinner seg i, pluss eventuelt **inngangsverdier** bestemmer **utgangsverdiene**.

Tilstandsmaskinens-konseptet gir en **enkel** og **oversiktlig** måte å designe **avanserte systemer** på.

Tilstand:
- er et begrep som benyttes til å beskrive systemets status / tilstand.
- er et verdisett / attributter som beskriver systemets egenskaper.

Hendelser:
- er et begrep som benyttes om innganger / påvirkninger på systemet.
- kan beskrives som en plutselig og kortvarig påvirkning av systemet.

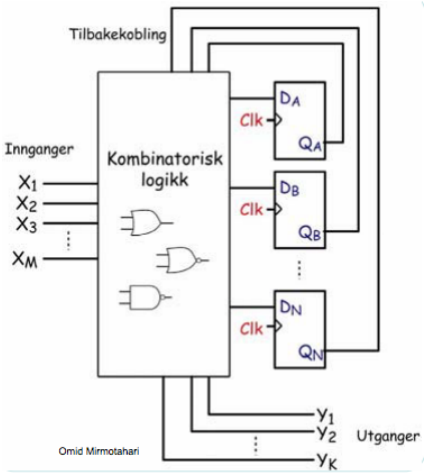
Aksjoner:
- er det som kommer ut av systemet. Det vil si resultatet.
- er en respons på en hendelse.



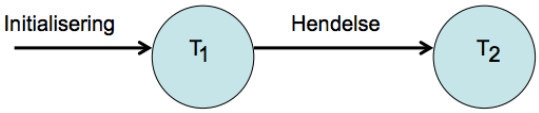
Generell tilstandsmaskin basert på D flip-flops.

N-stk flip-flops gir 2^N forskjellige tilstander.

Utgangssignalene er en funksjon av **nåværende tilstand** pluss eventuelle inngangsverdier.



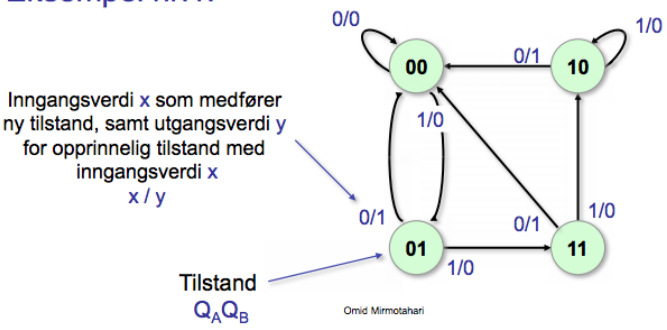
For å visualisere oppførselen til systemer brukes gjerne tilstandsdiagrammer
- sirkler angir tilstander
- piler angir tilstandsendring
- Hendelse og aksjoner settes over piler som angir tilstandsendringen

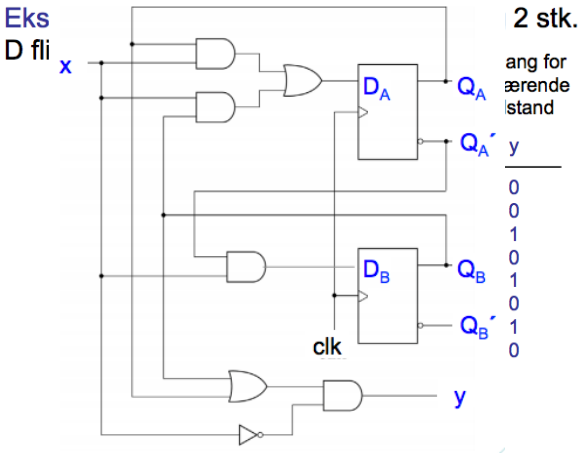


Implementasjon og kretsdesign

Tilstandsdiagram = grafisk illustrasjon av egenskapene til en tilstandsmaskin
Tilstandstabell = sannhetstabell for tilstandsmaskin

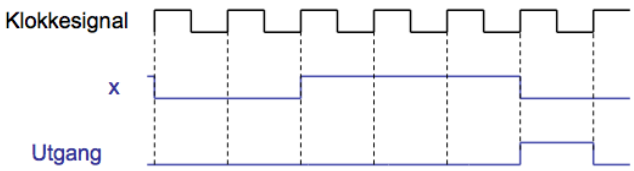
Eksempel nr.1:





Tilstandsmaskin der utgang **y** er en funksjon av **tilstanden** gitt av verdiene til $\mathbf{Q_A}$ og $\mathbf{Q_B}$, samt inngangen **x**.

Design av sekvensdetektor
Ønsker å lage en krets som finner ut om det har forekommet tre eller flere **"1"**ere etter hverandre i en klokket bit-sekvens x
Klokket bit-sekvens: Binært signal som kun kan skifte verdi synkront med et klokkesignal



Tilstandsdiagram
Velger å ha **4** tilstander. Lar hver tilstand symbolisere antall **"1"**ere som ligger etter hverandre i bit-sekvensen.
Inngang: bit-sekvens x
Utgang: gitt av tilstanden, **"0"** for tilstand **0-2**, "**1**" for tilstand **3**

Bruker D flip-flops
D_A og D_B settes til de verdiene man ønsker at Q_A og Q_B skal ha i neste tilstand

$$D_A = Q_A'Q_Bx + Q_AQ_B'x + Q_AQ_Bx$$

$$D_B = Q_A'Q_B'x + Q_AQ_B'x + Q_AQ_Bx$$

$$y = Q_AQ_B$$

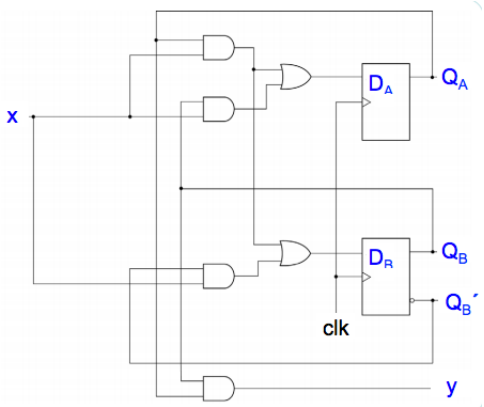
Nåværende tilstand		Inngang	Neste tilstand		Utgang for nåværende tilstand
Q_A	Q_B	x	Q_A	Q_B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

Forenkler uttrykkene med Karnaugh-diagram

$$D_A = Q_Ax + Q_Bx$$

$$D_B = Q_Ax + Q_B'x$$

$$y = Q_AQ_B$$



<u>Reduksjon av tilstander</u>
En tilstandsmaskin gir oss en eller flere **utgangssignal** som **funksjon** av en eller flere **inngangssignal**.

Hvordan dette implementeres internt i maskinen er uinteressant sett utenifra

I noen tilfeller kan man fjerne tilstander (forenkle designet) uten å påvirke inngangs/utgangs-funksjonene.

Hvis **to tilstander** har samme utgangssignal, samt leder til de **samme nye tilstandene** gitt like inngangsverdier, er de to opprinnelige tilstandene **like**. En tilstand som er lik en annen tilstand, kan **fjernes**.

Eksempel:

Tilstand G er lik tilstand E

Nåværende tilstand	Inngang	Neste tilstand	Utgang
A	0	B	0
A	1	B	0
B	0	C	0
B	1	D	0
C	0	A	0
C	1	D	0
D	0	E	0
D	1	F	1
E	0	A	0
E	1	F	1
F	0	G	0
F	1	F	1
G	0	A	0
G	1	F	1

Eksempel:

Fjerner tilstand G. Erstatter hopp til G med hopp til E

Nåværende tilstand	Inngang	Neste tilstand	Utgang
A	0	B	0
A	1	B	0
B	0	C	0
B	1	D	0
C	0	A	0
C	1	D	0
D	0	E	0
D	1	F	1
E	0	A	0
E	1	F	1
F	0	E	0
F	1	F	1

Eksempel:

Nå er tilstand F lik tilstand D

Fjerner tilstand F

Nåværende tilstand	Inngang	Neste tilstand	Utgang
A	0	B	0
A	1	B	0
B	0	C	0
B	1	D	0
C	0	A	0
C	1	D	0
D	0	E	0
D	1	F	1
E	0	A	0
E	1	F	1
F	0	E	0
F	1	F	1

Eksempel:

Har fjernet tilstand F

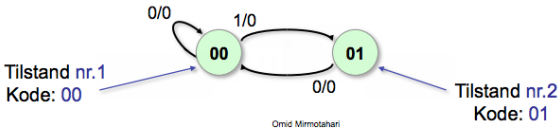
Nåværende tilstand	Inngang	Neste tilstand	Utgang
A	0	B	0
A	1	B	0
B	0	C	0
B	1	D	0
C	0	A	0
C	1	D	0
D	0	E	0
D	1	D	1
E	0	A	0
E	1	D	1

Tilordning av tilstandskoder

I en tilstandsmaskin med **M** tilstander må hver tilstand tilordnes en kode basert på minimum **N** bit der $2^N \geq M$.

Kompleksiteten til den kombinatoriske delen avhenger av valg av tilstandskode.

Anbefalt strategi for valg av kode: prøv-og-feil i tilstandsdiagrammet

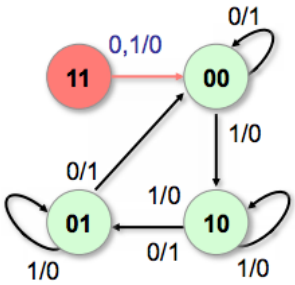


Ubrukte tilstander

I en tilstandsmaskin med **N** flip-flopper vil det alltid finnes 2^N tilstander. Designer man for **M** tilstander der $M < 2^N$ vil det finnes ubrukte tilstander.

Problem: Under oppstart (power up) har man ikke full kontroll på hvilken tilstand man havner i først. Havner man i en ubrukt tilstand som ikke leder videre til de ønskede tilstandene vil systemet bli låst.

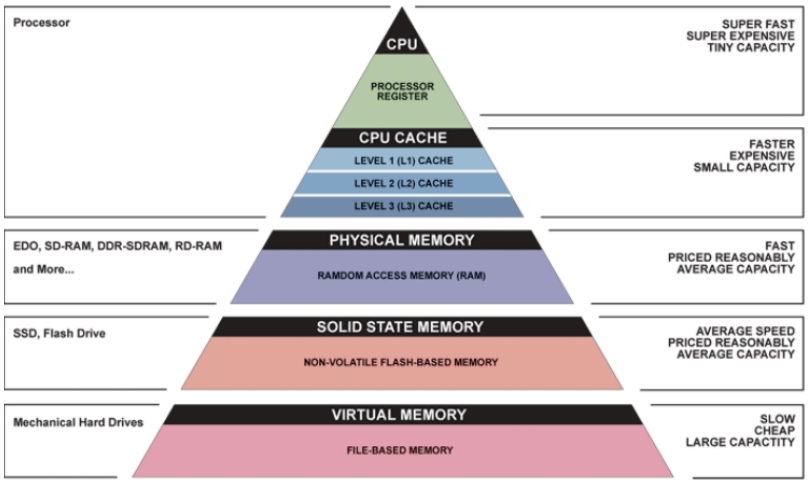
Løsning: Design systemet slik at alle ubrukte tilstander leder videre til en ønsket tilstand.



Generell designprosedyre basert på D flip-flops

1) Definer tilstandene, inngangene og utgangene
2) Velg tilstandskoder, og tegn tilstandsdiagram
3) Tegn tilstandstabell
4) Reduser antall tilstander hvis nødvendig
5) Bytt tilstandskoder hvis nødvendig for å forenkle
6) Finn de kombinatoriske funksjonene
7) Sjekk at ubrukte tilstander leder til ønskede tilstander
8) Tegn opp kretsen

Uke 10 – Datamaskinarkitektur



Pipelining

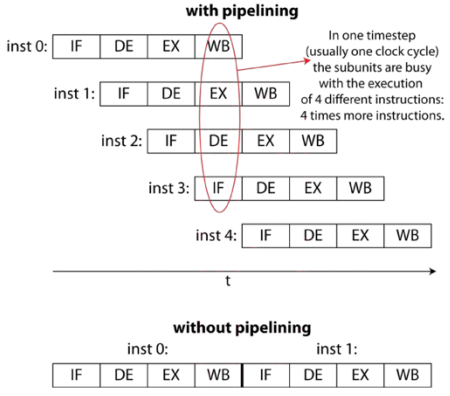
- Innfører samlebåndsprinsipp for eksekvering av instruksjoner
- Hver instruksjon må splittes opp i uavhengige deler (subinstruksjoner) som utføres etter hverandre
- Hver subinstruksjon kan utføres uavhengig av de andre subinstruksjonene
- Neste instruksjon settes igang før forrige instruksjon er helt ferdig.
- NB!! Hver instruksjon tar like lang tid å utføre, men prosessoren utfører flere instruksjoneri et gitt tidsrom!

Pipelining - Analogi

- Vasking av klær
    1. Vaske i vaskemaskin
    2. Trøke i tørketrommel
    3. Brette sammen
    4. Sette dem i skapet

Pipelining instruksjonssett

- Eksempelvis kan vi ha følgende subinstruksjoneri en pipeline
    - IF: Instruction fetch (get the instruction)
    - DE: decode and load (from a register)
    - EX: Execute
    - WB: Write back (write the result to a register)
- **PS!** Read og write fra register/MEM kan gjøres i hver sin halvdel av en sykel (1/2 klokkeperiode)



Pipeline med flere trinn

- Denne 4-trinns pipeline er den korteste som finnes for en CPU.
- Moderne CPU bruker vesentlig flere trinn
- Pentium III har 16 trinn, Pentrium 4 har 31 trinn

Speed-up

- Det å ha en 4 trinns pipeline betyr ikke at man får 4 ganger raskere prosessering. Det går alltid noe tid bort til administrering av instruksjone

Komplikasjoner ved pipelining

- Til enhver tid kan det være subinstruksjoner fra opptil 4 instruksjoner i en pipeline
- Noen ganger er ikke alle subinstruksjonene gyldige
- Neste instruksjon kan ikke eksekveres rett etter hvis hopp-betingelsen slår til
- En slik situasjon kalles HAZARD. Vi har tre typer:
    1. Resource hazard
    2. Data hazard
    3. Control Hazard