UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

INF1400

Digital teknologi

Joakim Myrvoll

2014

Innhold

1 Forenkling av funksjonsuttrykk

1.1 Huntingtons postulater

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

- Dualitet for postulatene:
  Kan bytte "·"med + hvis man bytter "0"med "1"
- Presedens:
  Først utføres "()", så "'", så "·"og til slutt "+"

1.2 DeMorgans

- (x + y)' = x'y'
- (xy)' = x' + y'

1.3 Karnaughdiagram

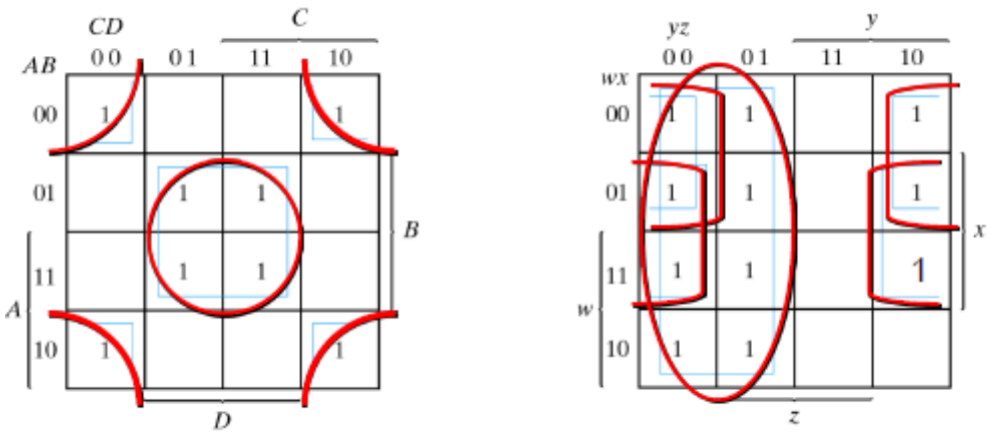
- Grafisk metode for forenkling av Boolske uttrykk
  - Uttrykket må være representert ved sum av mintermer.

1.3.1 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 kalles don't care kombinasjoner og markers med "X"

1.3.2 Grupperingsregler

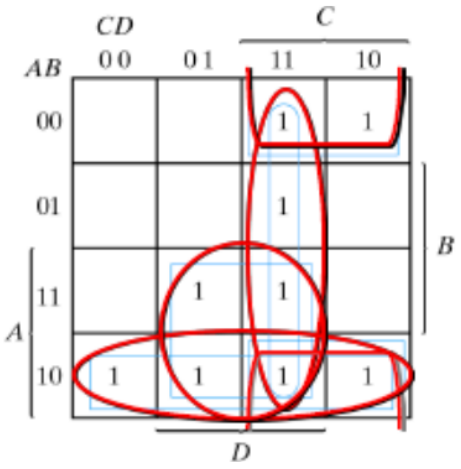
- Velg så store grupper av elementer som mulig.
- Antall elementer må være potens av 2.
- Gruppere 1'erne $\rightarrow$ F
- Gruppere 0'erne $\rightarrow$ F'
- Don't carekan grupperes etter eget ønske



Figur 1: Eksempel på grupperinger

1.3.3 Utlesningsregler

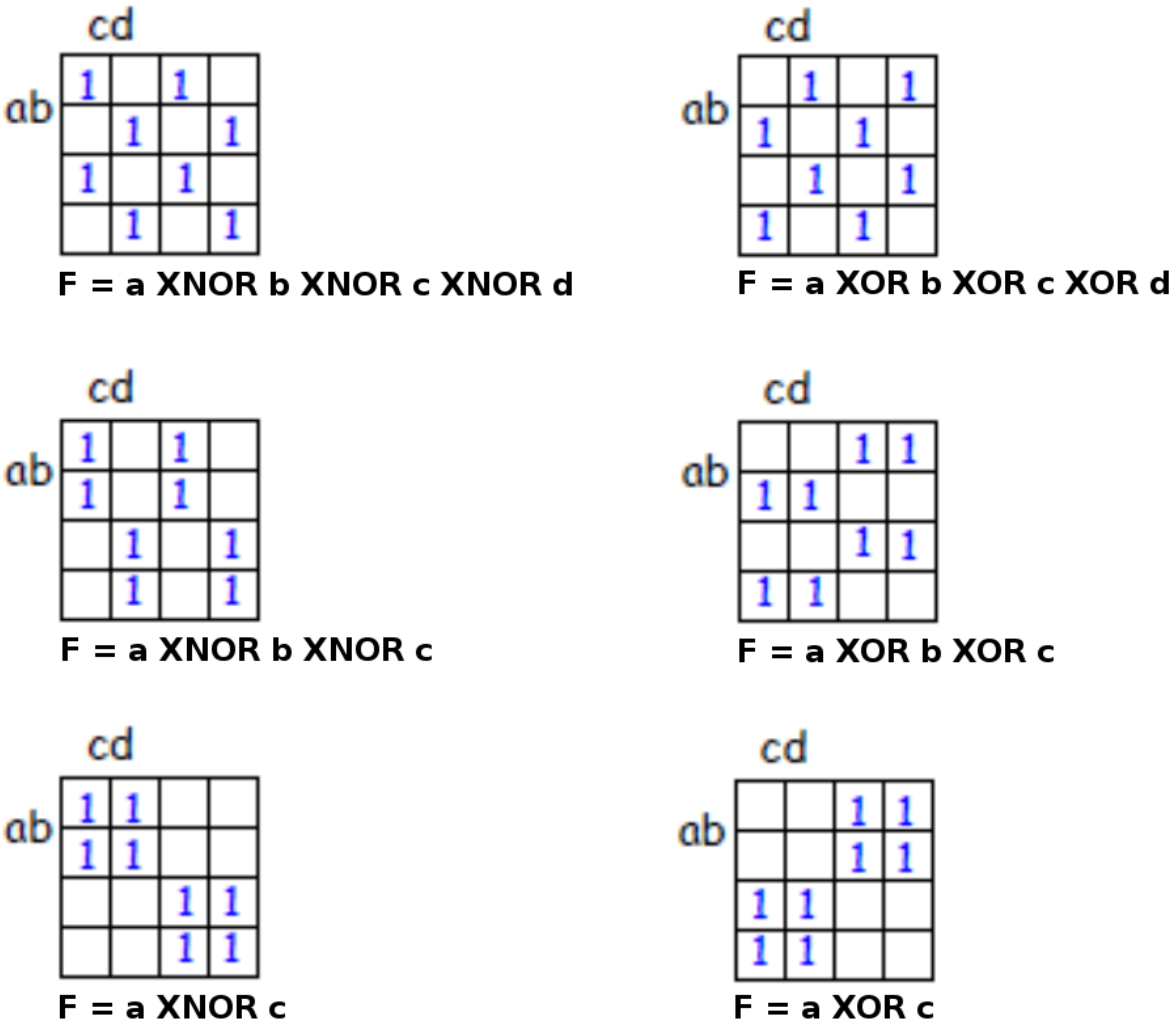
- Les av hver gruppe slik at den mintermen som ikke varierer gjelder.
- Diagrammets funksjon blir da summen av hvert gruppeledd.
- Jo større ruter/grupper desto enklere uttrykk



Figur 2: F = AD + AB' + CD + B'C

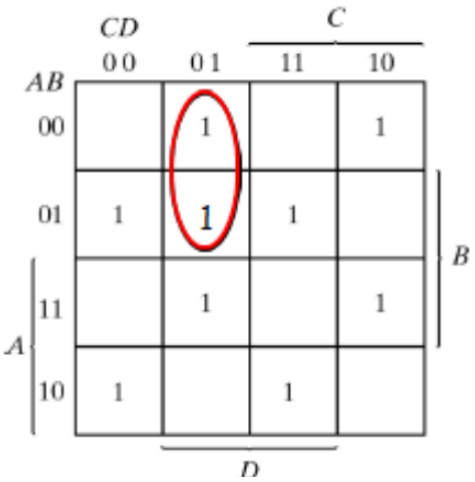
1.3.4 Spesialteknikker

- XOR/XNOR funksjonen dekker maksimalt "uheldige"1'er plasseringer i diagrammet



Figur 3:

- XOR/XNOR funksjonen kan kombineres med andre ledd.



Figur 4: F = A $\oplus$ B $\oplus$ C $\oplus$ D + A'C'D

2 Maks- og Mintermer

2.1 Minterm

I en funksjon kan en binær variabel x opptre som x eller x'. En funksjon kan være gitt på sum av produktform.

Hvert produktleddsom inneholder alle variablene, kalles en minterm.

For n variable fins det 2^n forskjellige mintermer.

Eksempel:
For to variable fins det 2^2 forskjellige mintermer:
xy + xy' + x'y + x'y'

2.2 Maksterm

En funksjon kan være gitt på produkt av sumform.

Hvert summeleddsom inneholder alle variablene kalles en maksterm.

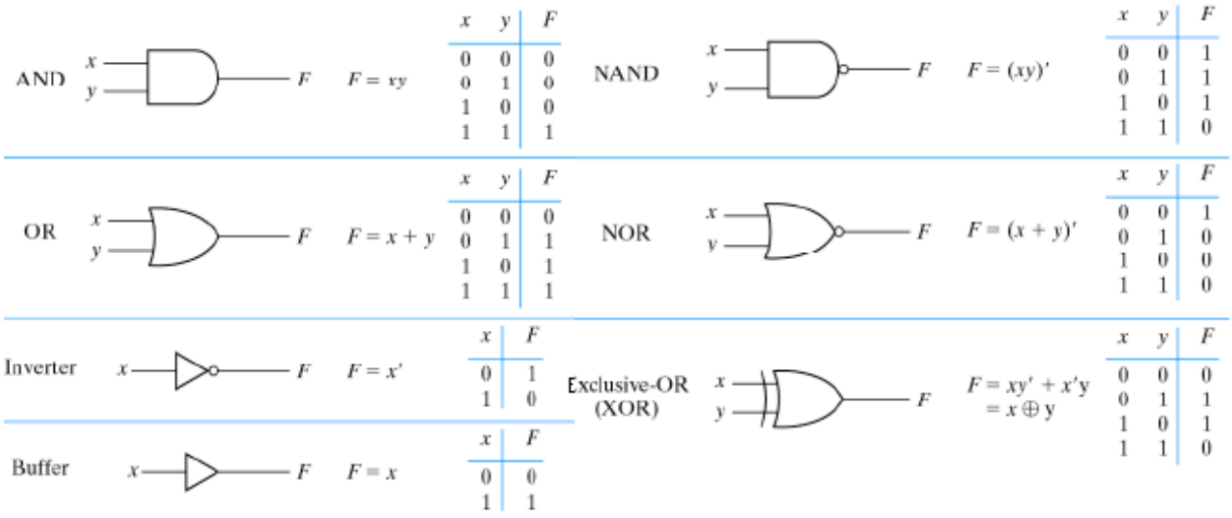
For n variable fins det 2^n forskjellige maksterm er.

Eksempel:
For to variable fins det 2^2 forskjellige maksterm er:
(x+y)(x+y')(x'+y)(x'+y')

2.3 Notasjon

Minterm: F(x,y,z) = $\sum(m_3, m_6)$ = $\sum$(3,6) = x'yz + xyz'
Maksterm: F(x,y,z) = $\Pi(M_3, M_6)$ = Π(3,6) = (x+y'+z')+(x'+y'+z)

3 Logiske Porter



Figur 5: Logiske porter

4 Konvertering av tall

4.1 Heksadesimalt til binært

Hvert heksadesimale tall består av ett fire-bits binært tall.
Eksempel:

$(123)_{16}$
$1 \rightarrow 0001$
$2 \rightarrow 0010$
$3 \rightarrow 0011$
$\rightarrow (0001\ 0010\ 0011)_2$

4.2 Desimalt til binært

Del på to og se på resten.
Eksempel:

$(123{,}625)_{10}$

123	: 2 → 1	LSB
61	: 2 → 1	
30	: 2 → 0	
15	: 2 → 1	
7	: 2 → 1	
3	: 2 → 1	
1	: 2 → 1	

,

0,6252 = 1,25 → 1 LSB
0,25· 2 = 0,50 → 0

0,50· 2 = 1,00 $\rightarrow$ 1
$\rightarrow$ (0111 1011,101)$_2$

4.3 Oktadesimalt til binært

Hvert oktadesimale tall består av ett tre-bits binært tall.
Eksempel:

(123)$_8$
1 $\rightarrow$ 001
2 $\rightarrow$ 010
3 $\rightarrow$ 011
$\rightarrow$ (001 010 011)$_2$

4.4 Heksadesimalt til desimalt

Eksempel:

(123,123)$_{16}$

$$1 \rightarrow 1 \cdot 16^2 \quad \rightarrow 256$$
$$2 \rightarrow 2 \cdot 16^1 \quad \rightarrow 32$$
$$3 \rightarrow 3 \cdot 16^0 \quad \rightarrow 3$$
,
$$1 \rightarrow 1 \cdot 16^{-1} \rightarrow \frac{1}{16^1} \quad \rightarrow 0.0625$$
$$2 \rightarrow 2 \cdot 16^{-2} \rightarrow \frac{2}{16^2} \quad \rightarrow 0.0078125$$
$$3 \rightarrow 2 \cdot 16^{-3} \rightarrow \frac{3}{16^3} \quad \rightarrow 0.000732421875$$

$\rightarrow 256 + 32 + 3 + 0.0625 + 0.0078125 + 0.000732421875$
$\rightarrow$ (291.0710449)$_{10}$

4.5 Generelt

$$(\ldots a_2 a_1 a_0, a_{-1} a_{-2} \ldots)_r = \ldots + a_2 \cdot r^2 + a_1 \cdot r^1 + a_0 \cdot r^0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + \ldots$$

5 2'er komplement

For å konvertere til negative tall brukes ofte 2'er komplement. Man må da ta den inverse av det binære tallet og legge til en.

Eksemepl:

-5,5
5,5 = 0101,1

	1010,0	(1'er komplement)
+	0000,1	(legg til en)
=	1010,1	(-5,5)

Hvis man vil ha et fler-bits svar, legg til enere istedenfor nuller.

5.1 Binært til desimalt

For å konverte fra 2'er komplement til desimalt bruker man samme fremgangsmåte;

11011

```
   00100    (invers)
+  00001    (legg til en)
-----------
=  00101    (· -1)
```

$\rightarrow$ -$(1 \cdot 2^2 + 1 \cdot 2^0)$
$\rightarrow$ -5

6 Boolsk Algebra

6.1 Binær addisjon

```
    1001    (9)
+   1000    (8)
------------
=  10001    (17)
```

6.2 Binær subtraksjon

```
    1010,101    (10,625)
+   0101,001    (-5,5)
--------------------
= ~~1~~0101,001    (5,125)
```

6.3 Binær multiplikasjon

```
      1101    (13)
x     0111    (7)
-----------------------
      0111    (0111 x 1)
     0000     (0111 x 0, shift)
    0111      (0111 x 1, shift)
   0111       (0111 x 1, shift)
-----------------------
=  1011011    (91)
```

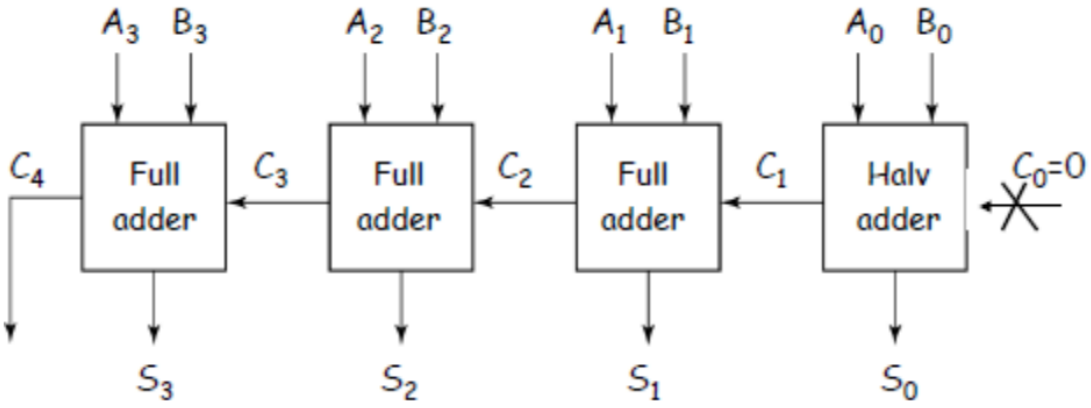
6.4 Binær divisjon

91 ÷ 7 = 13
1011011 ÷ 0111 = 01101

```
  111      (0)
-
---------
  1011
- 0111     (1)
---------
  01000
- 00111    (1)
---------
  000011
- 000111   (0)
---------
  0000111
- 0000111  (1)
---------
  0000000
```

7 Kombinatorisk Logikk

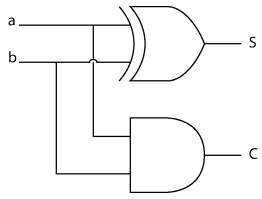
7.1 Binær adder



Figur 6: Adder system

7.1.1 Halvadder

En *halvadder* kan summere to 1-bit binære numre, og gi ut sum (S) og mente (C)



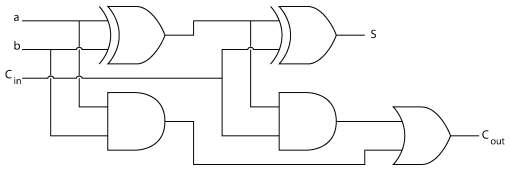
Figur 7: Implementasjon av halvadder

a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Figur 8: Sannhetstabell, halvadder

7.1.2 Fulladder

En *fulladder* kan summere to 1-bit binære numre og mente inn, og gi ut sum (S) og mente (C)

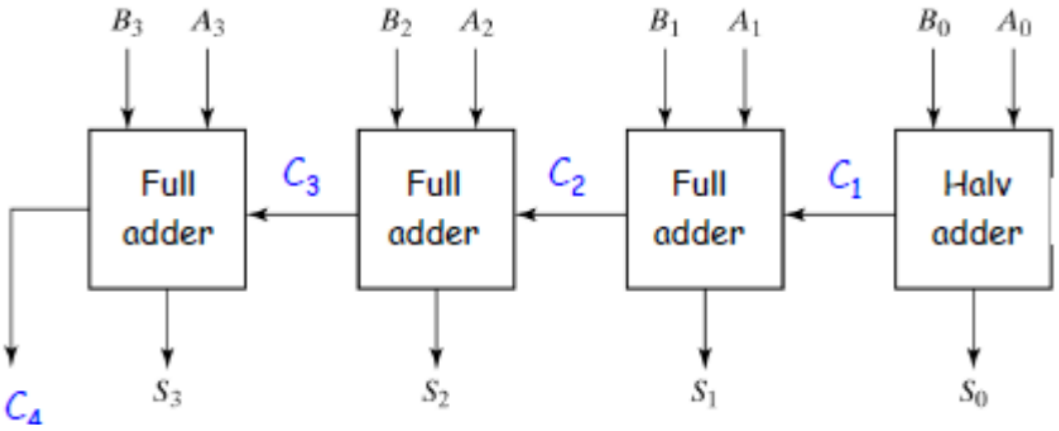


Figur 9: Implementasjon av fulladder

C_{in}	a	b	S	C_{out}
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

Figur 10: Sannhetstabell, fulladder

7.1.3 Menteforplantning



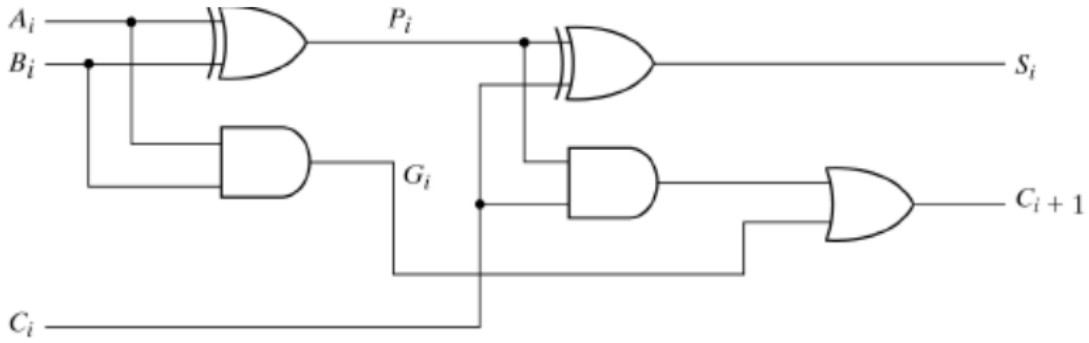
Figur 11: 4-bits binær adder

Portforsinkelse gir menteforplantning (rippeladder)

7.1.4 Carry look-ahead

Ønsker å unngå menteforplantning - gir økt hastighet.

- G_i – generate: brukes i menteforplantningen
- P_i – progagate: påvirker ikke menteforplantningen

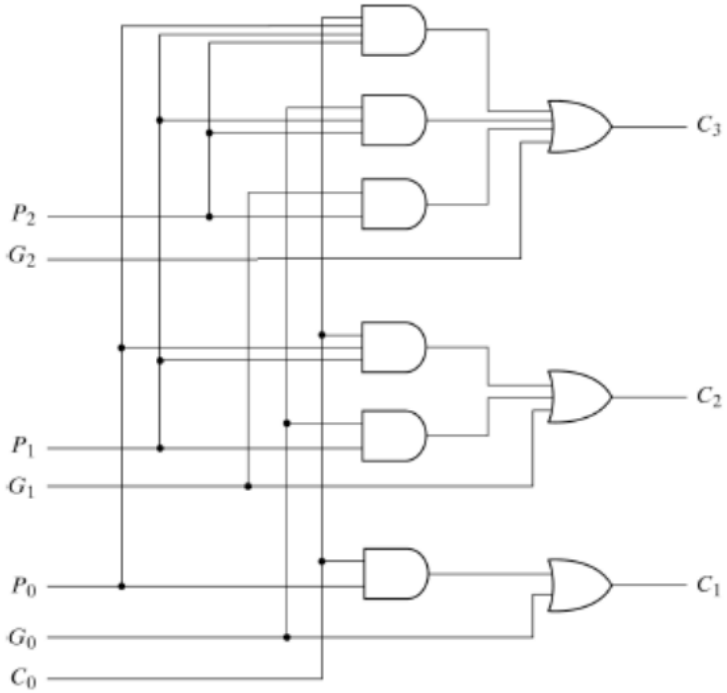


Figur 12:

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

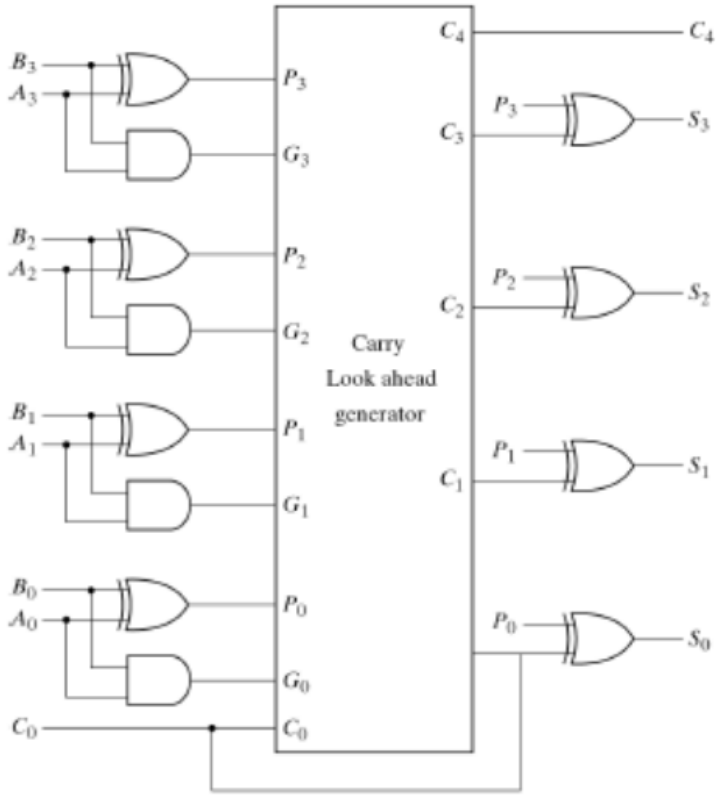
For en 4-bits adder bestående av 4 fulladdertrinn har vi:
Uttrykker C1, C2 og C3 rekursivt;

- C_1 = G_0 + P_0C_0
- C_2 = G_1+ P_1C_1 = G_0 + $P_1(G_0 + P_0C_0)$ = G_1 + P_1G_0 + $P_1P_0C_0$
- C_3 = G_2+ P_2C_2 = G_2 + P_2G_1 + $P_2P_1G_0$ + $P_2P_1P_0C_0$



Figur 13: Implementasjon av C_1, C_2, C_3

7.1.5 Carry look-ahead adder



Figur 14:

7.2 Komparator

Komparator – sammenligner to tall A og B

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

Eksempel: 4-bits komparator

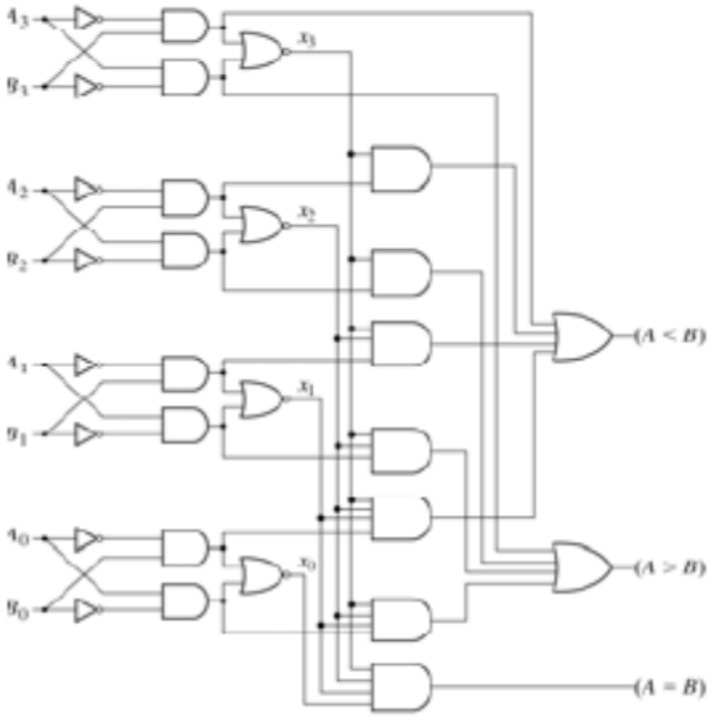
- Utgang A=B
  - Slår til hvis $A_0 = B_0$ og $A_1 = B_1$ og $A_2 = B_2$ og $A_3 = B_3$
  - Kan skrives: $(A0B \oplus 0)'(A1B \oplus 1)'(A2B \oplus 2)'(A3B \oplus 3)'$
- Utgang A>B slår til hvis:
  (A_3>B_3) eller
  (A_2>B_2 og A_3=B_3) eller
  (A_1>B_1 og A_2=B_2 og A_3=B_3) eller
  (A_0>B_0 og A_1=B_1 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'_3B_3) + (A'_2B_2) $(A_3 \oplus B_3)$'+ (A'_1B_1) $(A_2 \oplus B_2)$' $(A_3 \oplus B_3)$'+ $(A'_0B_0)(A_1 \oplus B_1)'(A_2 \oplus B_2)'(A_3 \oplus B_3)$'

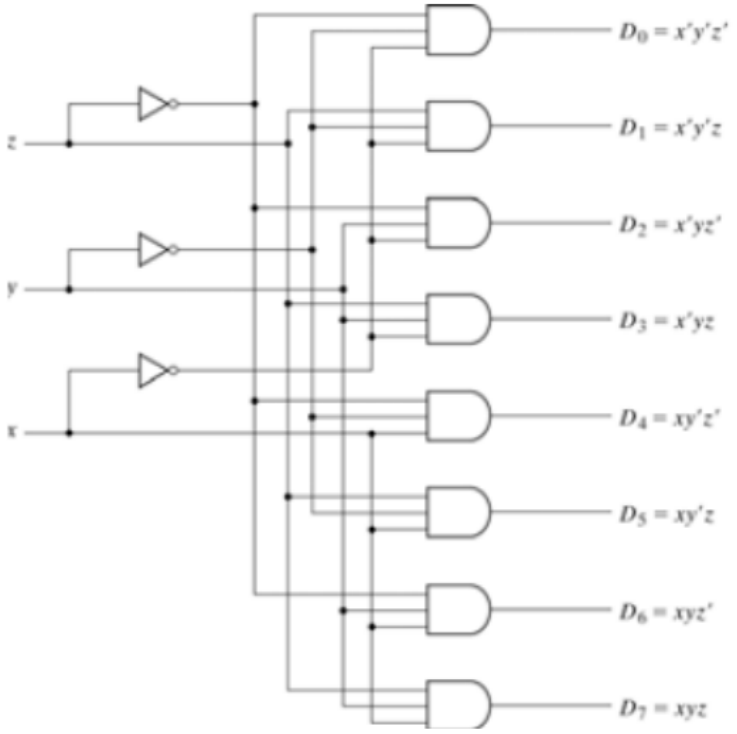
Eksempel



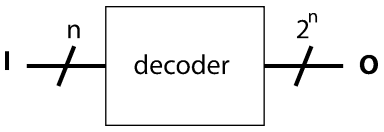
Figur 15: Komparator

7.3 Dekoder

Dekoder n inngangssignaler til 2^n utgangssignaler.
Tar inn et binært ord, gir ut alle mintermer.



Figur 16: Eksempel: 3 bit inn / 8 bit ut



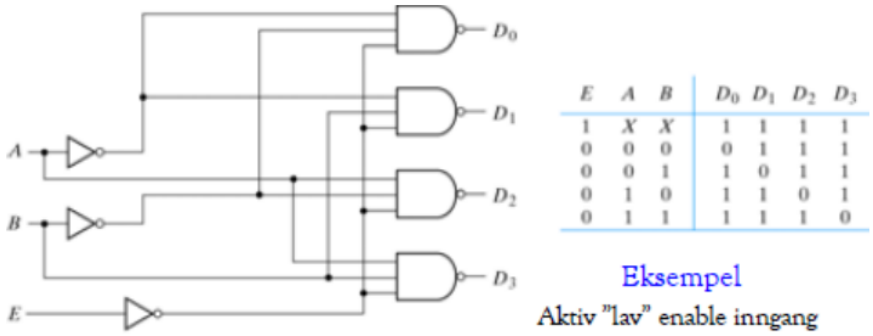
Figur 17: Dekoder-symbol

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

Figur 18: Sannhetstabell for 3-bit dekoder

7.3.1 Varianter

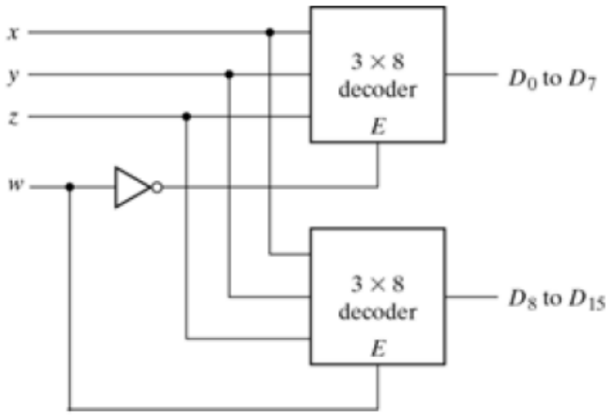
- Enable input:
  - Enable aktiv → normal operasjon
  - Enable inaktiv → alle utganger disablet



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

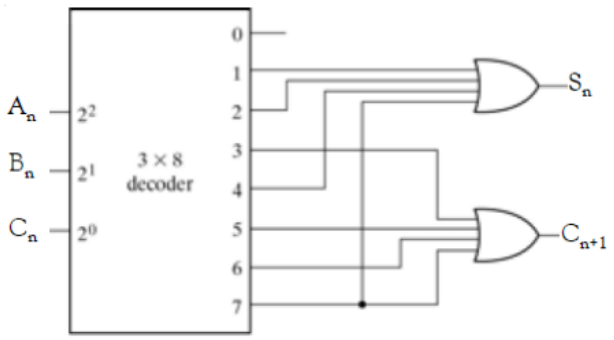
Figur 19: NAND logikk: Inverterte utganger

- Parallellkobling



Figur 20: Lager en 4x16 dekoder fra 2 stk 3x8 dekodere med enable innganger

- Generering av logiske funksjoner:
  Kan generere generelle logiske funksjoner direkte fra mintermene på utgangen



Figur 21: Fulladder

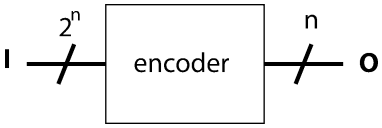
7.4 Enkoder

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

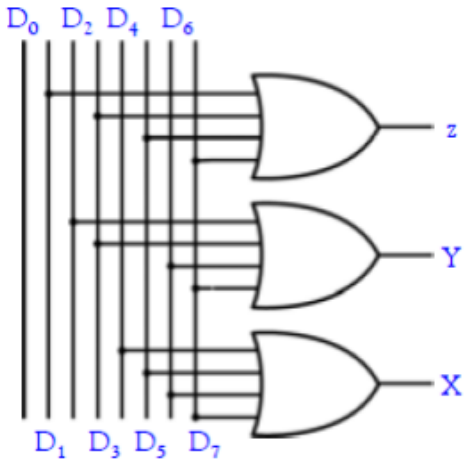
$x = D_4 + D_5 + D_6 + D_7$
$y = D_2 + D_3 + D_6 + D_7$
$z = D_1 + D_3 + D_5 + D_7$

Antar at det ikke eksisterer andre inngangskombinasjoner

Figur 22: 8x3 enkoder



Figur 23: Enkoder-symbol



Figur 24: 8x3 enkoder implementasjon

7.4.1 Prioritets-enkoder

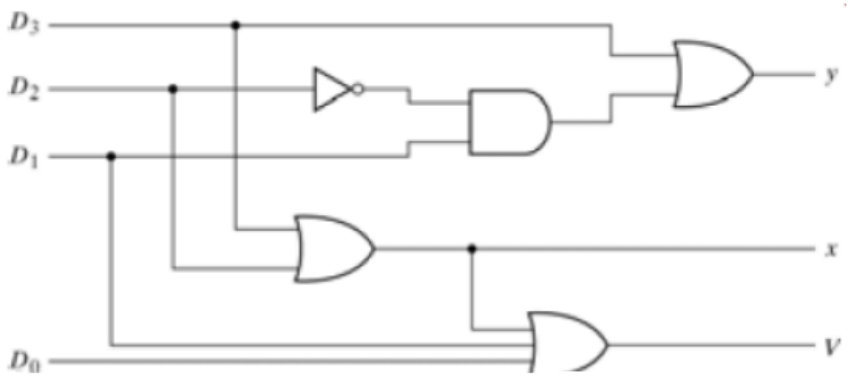
Problem i enkodere: Hva hvis man får flere 1'ere inn samtidig?

- Løsning: Prioritets-enkoder
  Hvis flere 1'ere inn ser kun på inngang med høyst indeks (prioritet)

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

Figur 25: 8x3 prioritets-enkoder

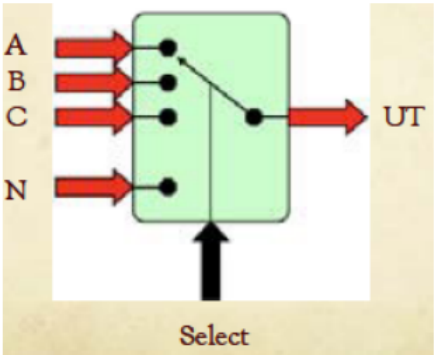
- Eksempel: 4x2 prioritets-enkoder med "valid"utgang



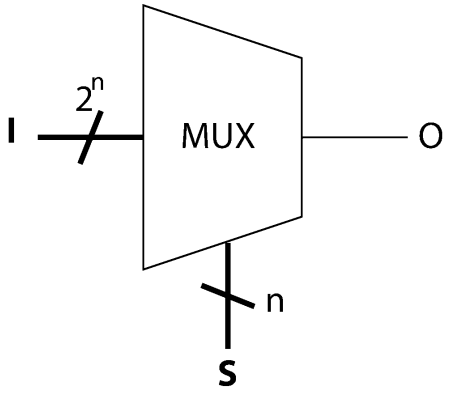
Figur 26: 'V' signaliserer at minst en inngang er '1'

7.5 Multiplexer (MUX)

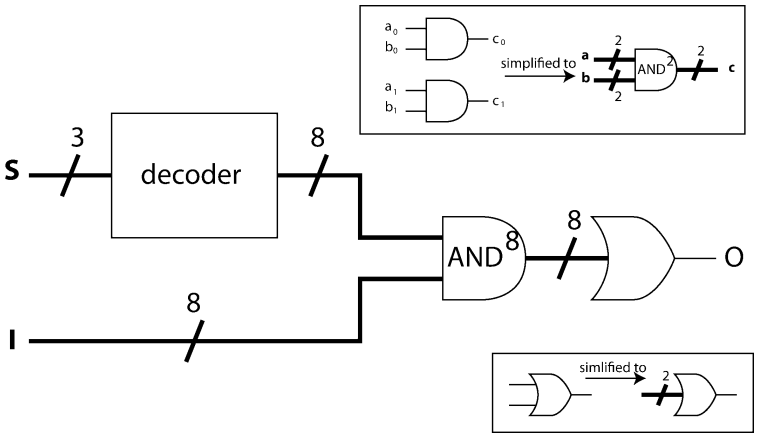
Velger hvilke innganger som slippes ut Hver inngang kan bestå av ett eller flere bit.



Figur 27: MUX



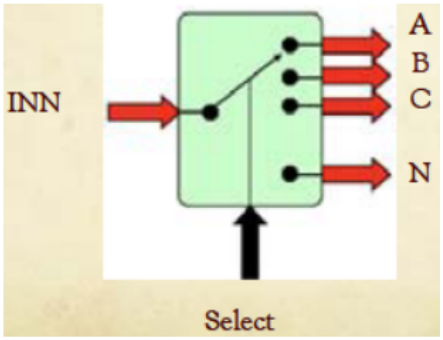
Figur 28: Multiplexer-symbol



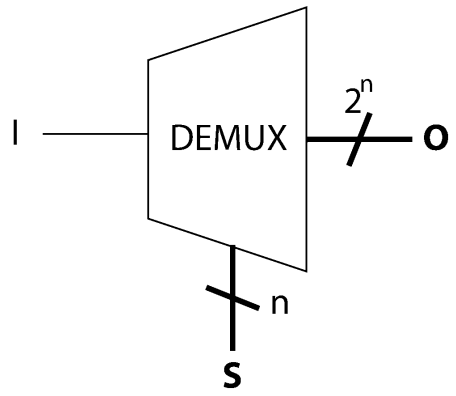
Figur 29: Mulig multiplexer implementasjon

7.6 Demultiplexer (DEMUX)

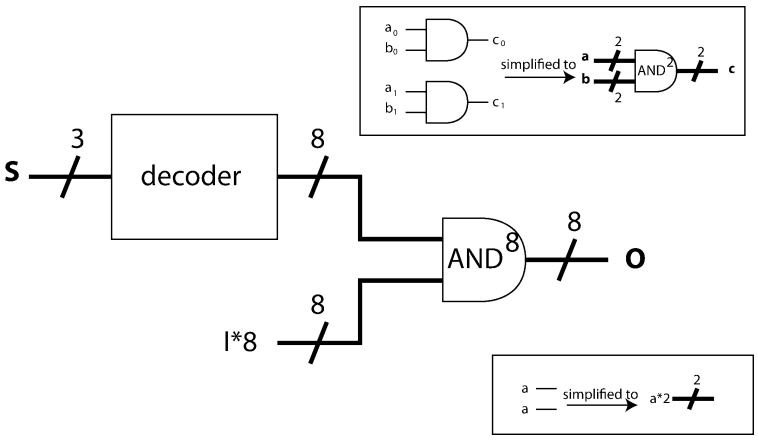
Motsatt av MUX



Figur 30: DEMUX



Figur 31: Demultiplexer-symbol

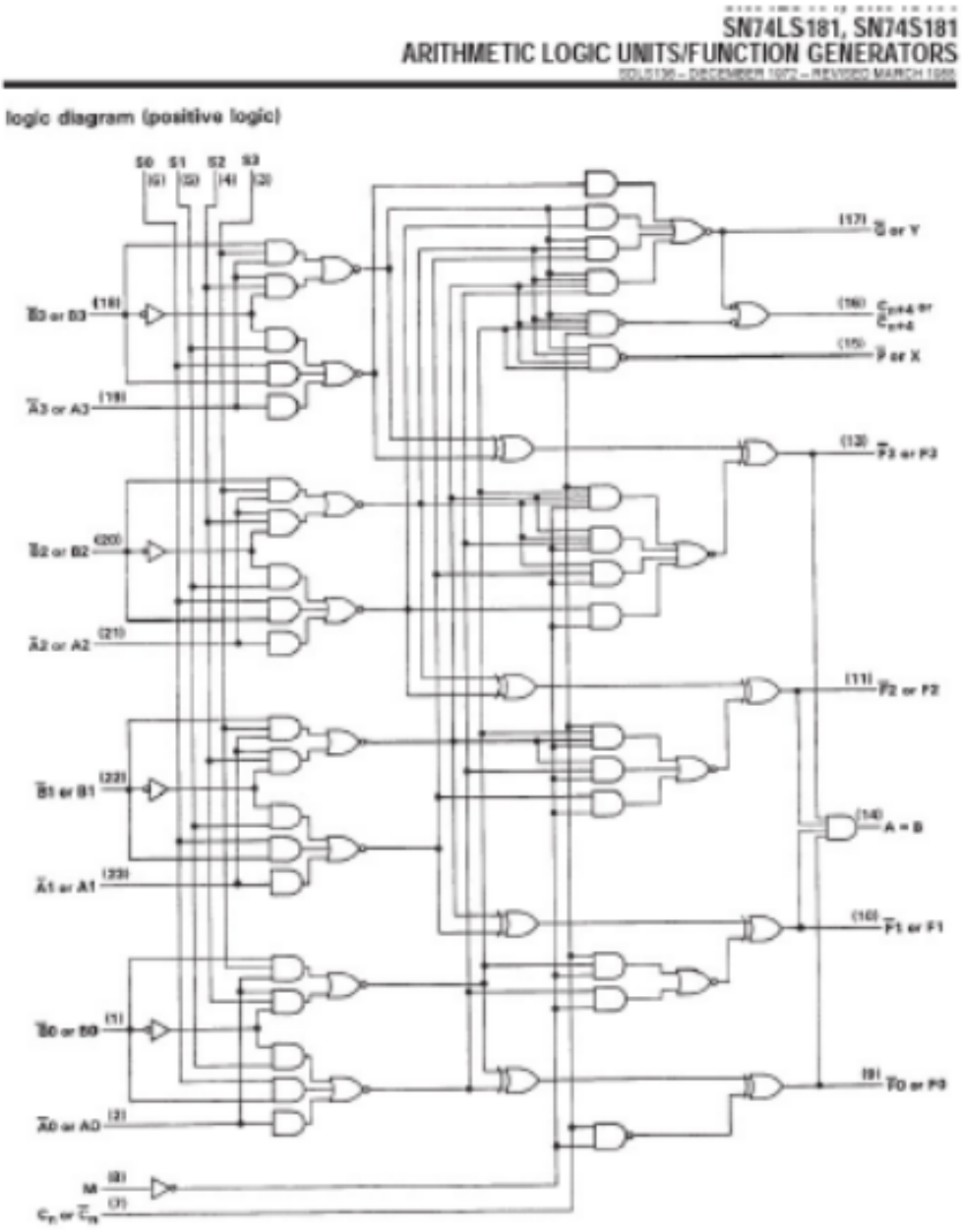


Figur 32: Mulig demultiplexer implementasjon

7.7 ALU (Aritmetic Logic Unit)

Generell regneenhet.

Eksempel
SN74LS181
4 bit utbyggbar
ALU
30 forskjellige operasjoner



Figur 33: SN74LS181

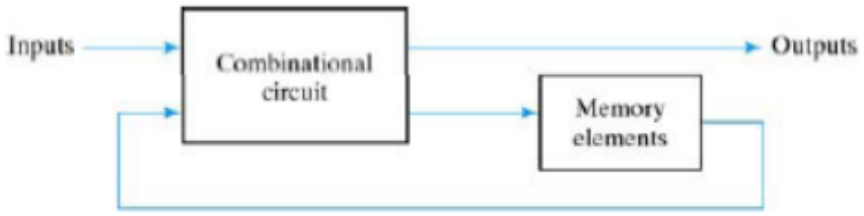
SELECTION				ACTIVE-HIGH DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	$\overline{C_n}$ = H (no carry)	$\overline{C_n}$ = L (with carry)
L	L	L	L	$F = \overline{A}$	F = A	F = A PLUS 1
L	L	L	H	$F = \overline{A + B}$	F = A + B	F = (A + B) PLUS 1
L	L	H	L	$F = \overline{A}B$	$F = A + \overline{B}$	$F = (A + \overline{B})$ 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 $A\overline{B}$	$F = A$ PLUS $A\overline{B}$ PLUS 1
L	H	L	H	$F = \overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$	$F = (A + B)$ 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

Figur 34: SN74LS181

8 Sekvensiell logikk

8.1 Definisjoner

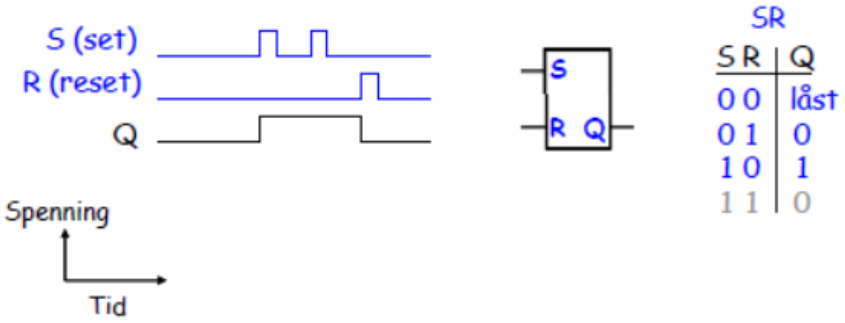
- Kombinatorisk logikk
  Utgangsverdiene er entydig gitt av nåværende kombinasjon av inngangsverdier
- Sekvensiell logikk
  Inneholder hukommelse (låsekretser)
  Utgangsverdiene er gitt av nåværende kombinasjon av inngangsverdier, samt sekvensen (tidligere inngangs-/utgangsverdier)



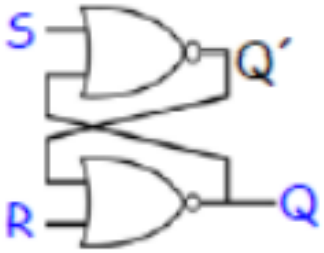
Figur 35: Sekvensiell logikk - generell oversikt

8.2 Latcher

8.2.1 SR Latch

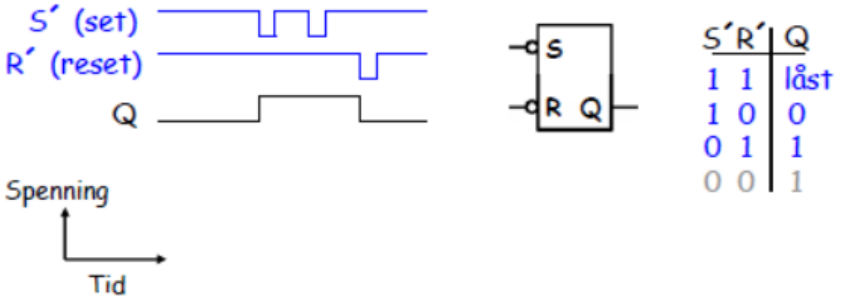


Figur 36: SR Latch

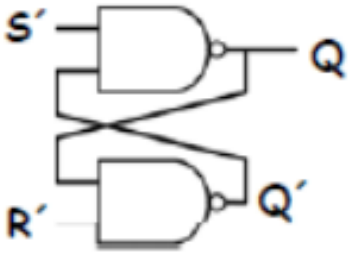


Figur 37: SR Latch implementert med NOR-porter

8.2.2 S'R' Latch



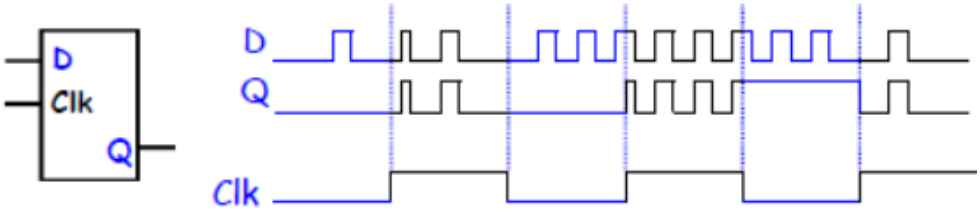
Figur 38: SR Latch



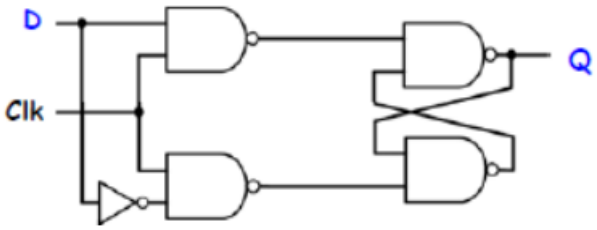
Figur 39: SR Latch implementert med NAND-porter

8.2.3 D Latch

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



Figur 40: D Latch



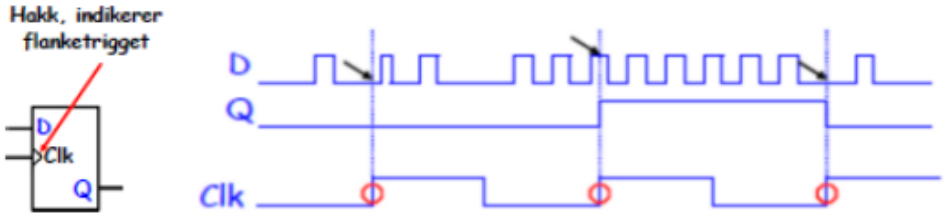
Figur 41: D Latch implementert med NAND-porter

8.3 Flip-flop'er

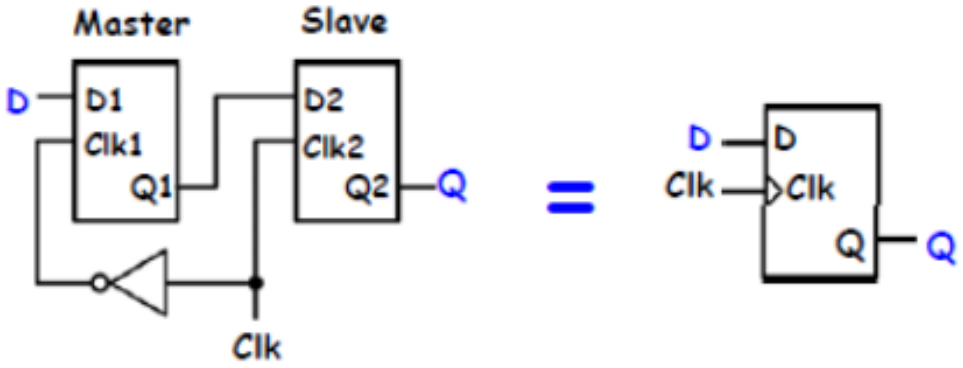
Flip-Flop'er kommer i to varianter:

- Positiv flanketrigget
  Utgangen vil skifte verdi i det øyeblikket klokkesignalet går fra "0til "1"
- Negativ flanketrigget
  Utgangen vil skifte verdi i det øyeblikket klokkesignalet går fra "1til "0"

8.3.1 D Flip-flop

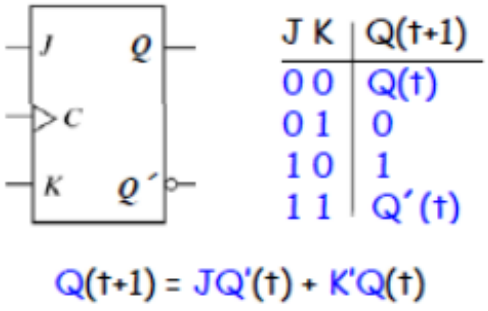


Figur 42: D Flip-Flop

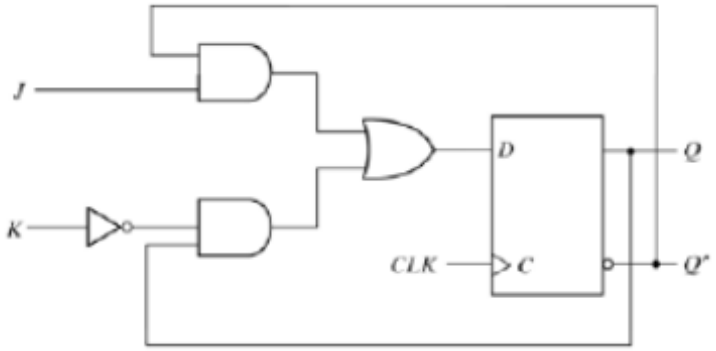


Figur 43: D Flip-Flop Master-Slave

8.3.2 JK Flip-flop

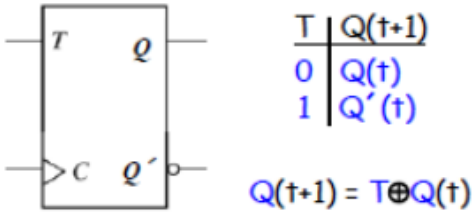


Figur 44: JK Flip-Flop

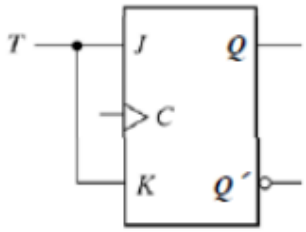


Figur 45: JK Flip-Flop implementasjon

8.3.3 T Flip-flop



Figur 46: T Flip-Flop

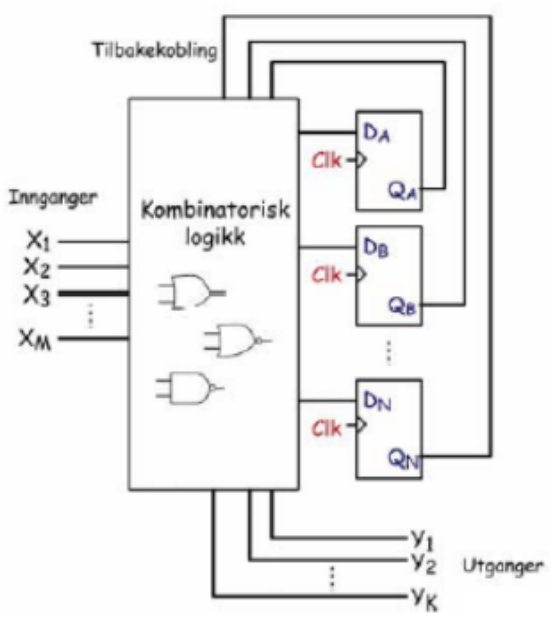


Figur 47: T Flip-Flop implementasjon

9 Tilstandsmaskin (FSM)

En tilstandsmaskin er et sekvensielt system som gjennomløper et sett med tilstander styrt av verdiene på inngangssignalene Tilstanden systemet befinner seg i, pluss evt. inngangsverdier bestemmer utgangsverdiene Tilstandsmaskins-konseptet gir en enkel og oversiktlig måte å designe avanserte systemer på.

Generell tilstandsmaskin basert på D flip-flops $\rightarrow$ N-stk flip-flops gir 2n forskjellige tilstander Utgangssignalene er en funksjon av nåværende tilstand pluss evt. inngangsverdier

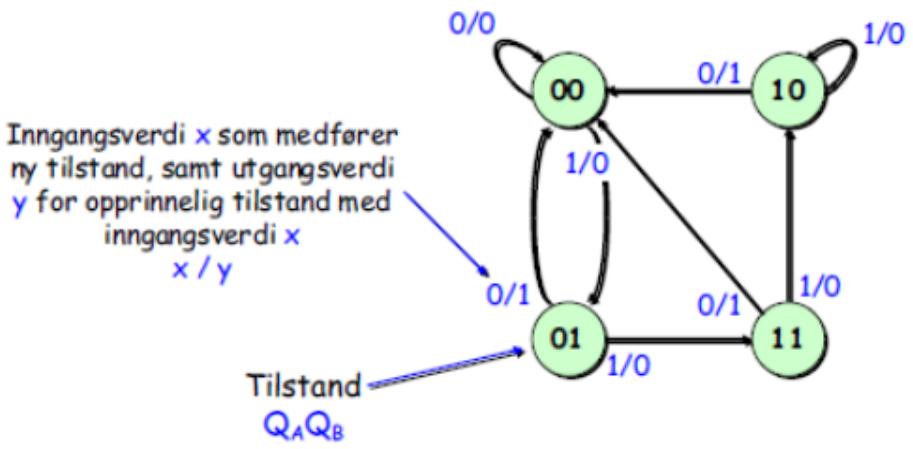


Figur 48: Tilstandsmaskin

9.1 Tilstandstabell

Nåværende tilstand $Q_A\ Q_B$	Inngang x	Neste tilstand $Q_{A+1}\ Q_{B+1}$	Utgang for nåværende tilstand y
00	0	00	0
00	1	01	0
01	0	00	1
01	1	11	0
10	0	00	1
10	1	10	0
11	0	00	1
11	1	10	0

9.2 Tilstandsdiagram



Figur 49: Tilstandsdiagram

9.3 Reduksjon av tilstander

En tilstandsmaskin gir oss en eller flere utgangssignal som funksjon av en eller flere inngangssignal.

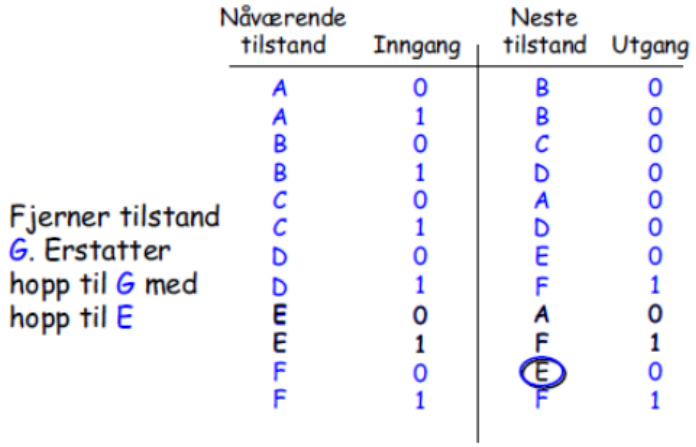
Hvordan dette implementeres internt i maskinen er uinteressant sett utenfra. I noen tilfeller kan man fjerne tilstander (forenkle designet) uten å påvirke inngangs/utgangsfunksjonene Hvis to tilstander har samme utgangssignal, samt leder til de samme nye tilstandene gitt like inngangsverdier, er de to opprinnelige tilstandene like. En tilstand som er lik en annen tilstand kan fjernes.

Eksempel:

Tilstand G er lik tilstand E

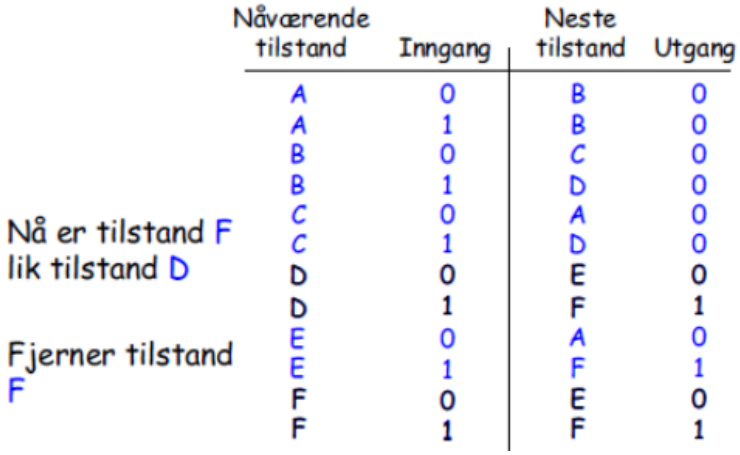
Nåværende tilstand	Inngang	Neste tilstand	Utgang
A	0	B	0
A	1	B	0
B	0	C	0
B	1	D	0
C	0	A	0
C	1	D	0
D	0	E	0
D	1	F	1
E	0	A	0
E	1	F	1
F	0	G	0
F	1	F	1
G	0	A	0
G	1	F	1

Figur 50: Reduksjon



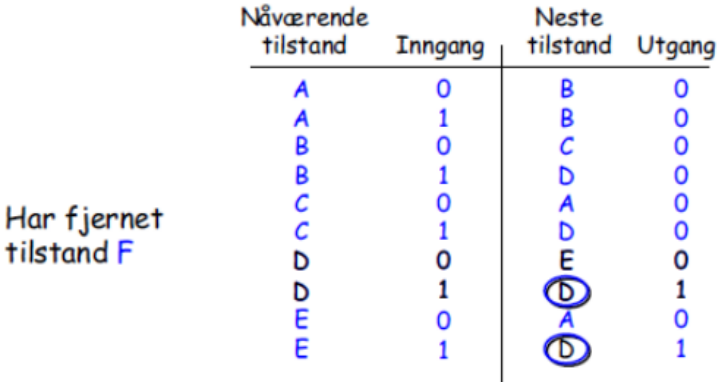
Nåværende tilstand	Inngang	Neste tilstand	Utgang
A	0	B	0
A	1	B	0
B	0	C	0
B	1	D	0
C	0	A	0
C	1	D	0
D	0	E	0
D	1	F	1
E	0	A	0
E	1	F	1
F	0	E	0
F	1	F	1

Figur 51: Reduksjon



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

Figur 52: Reduksjon



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

Figur 53: Reduksjon

9.4 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.

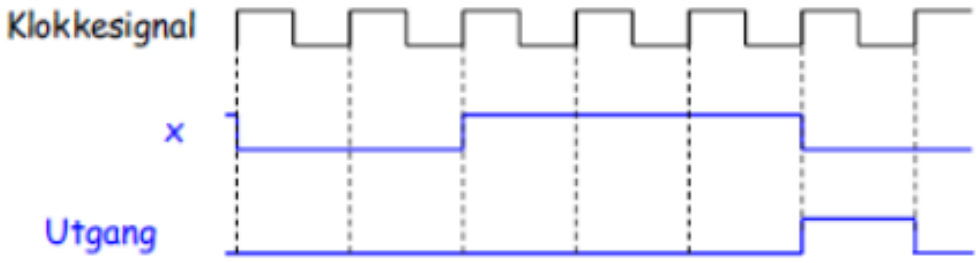
9.5 Generell designprosedyre basert på D flip-flops

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. Skjekk at ubrukte tilstander leder til ønskede tilstander
8. Tegn opp kretsen

9.6 Eksempel

Design av sekvensdetektor:
Ønsker å lage en krets som finner ut om det har forekommet tre eller flere "1ere etter hverandre i en klokket bit-sekvens x.

Klokket bit-sekvens: Binært signal som kun kan skifte verdi synkront med et klokkesignal.

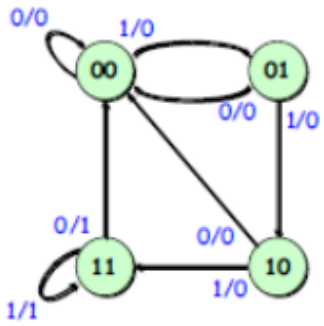


Figur 54:

9.6.1 Tilstandsdiagram

Velger å ha 4 tilstander. Lar hver tilstand symbolisere antall "1ere som ligger etter hverandre i bit-sekvensen.

Inngang: bit-sekvens x
Utgang: gitt av tilstanden, "0for tilstand 0-2, "1for tilstand 3



Figur 55: Tilstandsdiagram

9.6.2 Tilstandstabell

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.

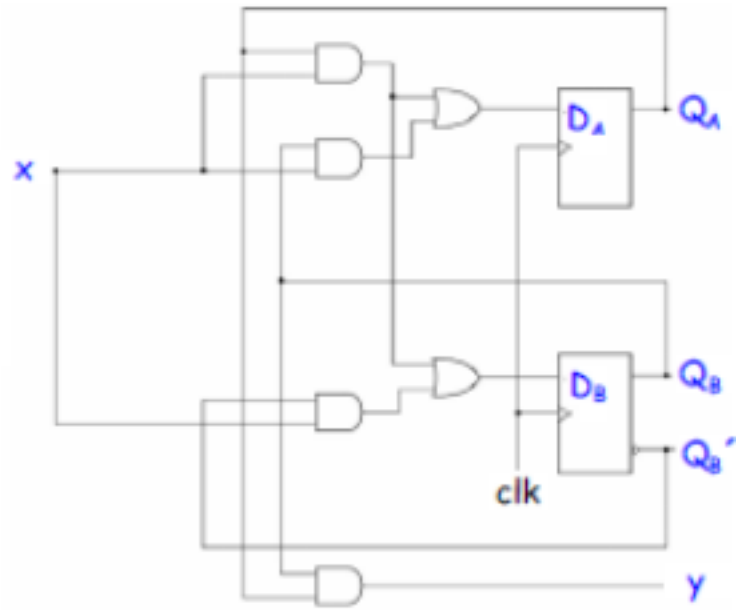
Nåværende tilstand Q_A Q_B	Inngang x	Neste tilstand D_A D_B	Utgang for nåværende tilstand y
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	11	0
11	0	00	1
11	1	11	1

D_A = Q_A'Q_Bx + $Q_A$$Q_B$'x + $Q_A$$Q_B$x
D_B = Q_A'Q_B'x + $Q_A$$Q_B$'x + $Q_A$$Q_B$x
y = $Q_A$$Q_B$

9.6.3 Forenkling

Forenkler uttrykkene med Karnaugh-diagram:

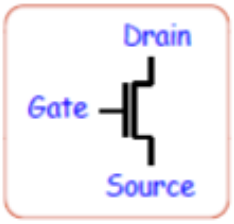
D_A = Q_Ax + Q_Bx
D_B = Q_Ax + Q_B'x
y = $Q_A$$Q_B$



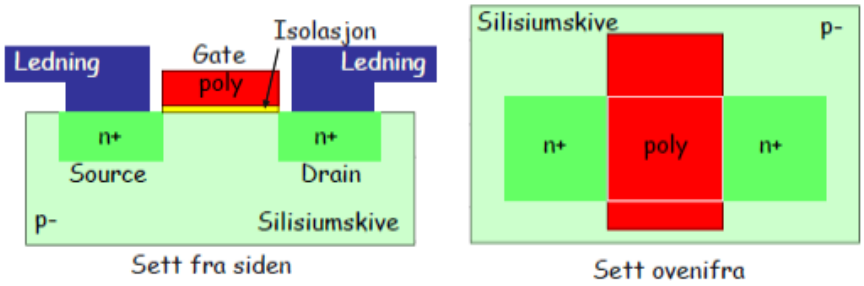
Figur 56: Tilstandsmaskin implementasjon

10 CMOS teknologi

10.1 NMOS (Negative doped Metal Oxide Silicon) transistoren



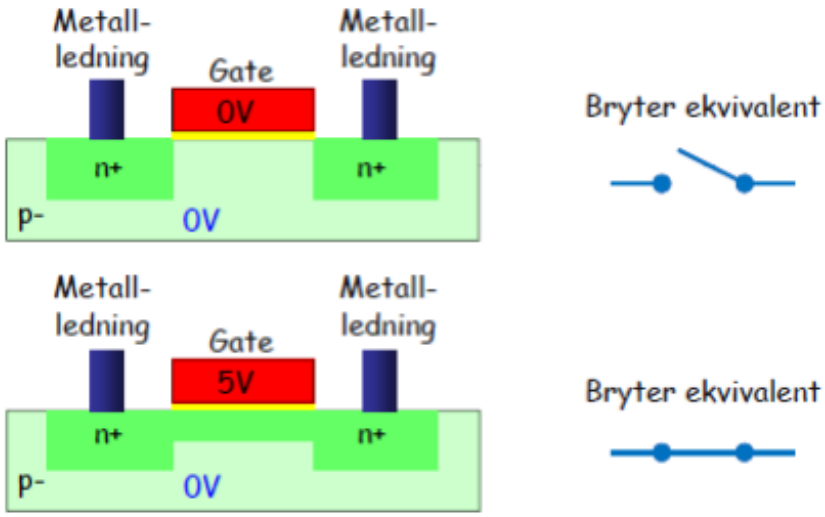
Figur 57: NMOS symbol



Figur 58: Spenningen på gate bestemmer om transistoren leder strøm mellom drain og source terminalene

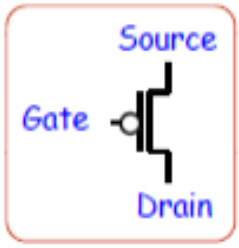
p- : Svakt positivt dopet silisium
n+ : Sterkt dopet silisium (ledende)
poly : Polykrystalinskt silisium (ledende)

10.1.1 NMOS brukt som styrt bryter (digital anvendelse)

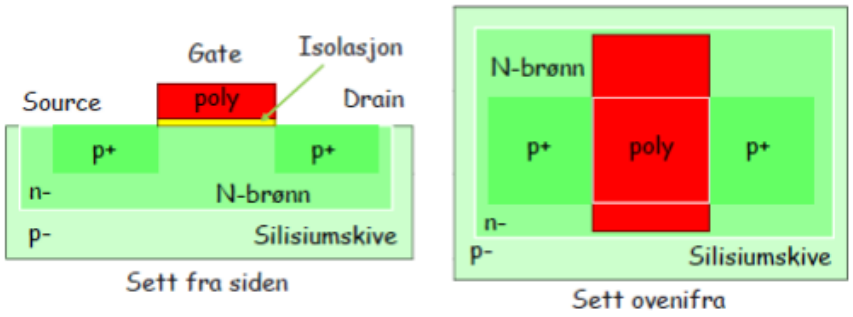


Figur 59: NMOS

10.2 PMOS (Positive doped Metal Oxide Silicon) transistoren



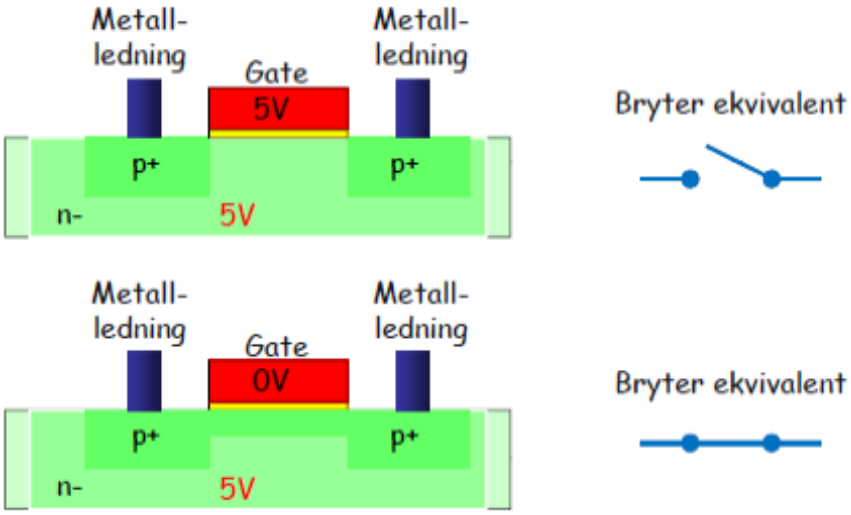
Figur 60: PMOS symbol



Figur 61: Spenningen på gate bestemmer om transistoren leder strøm i mellom drain og source terminalene

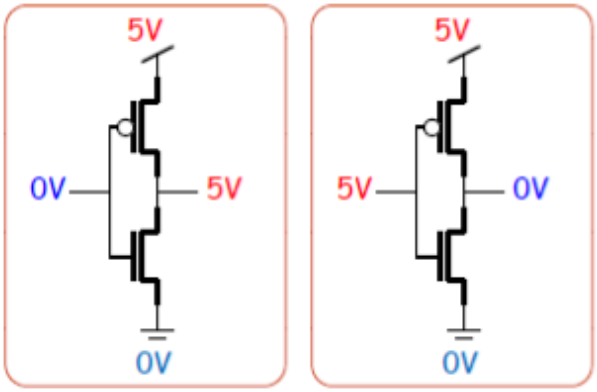
n- : Svakt negativt dopet silisium
p+ : Sterkt positivt dopet silisium (ledende)
poly : Polykrystalinskt silisium (ledende)

10.2.1 PMOS brukt som styrt bryter (digital anvendelse)



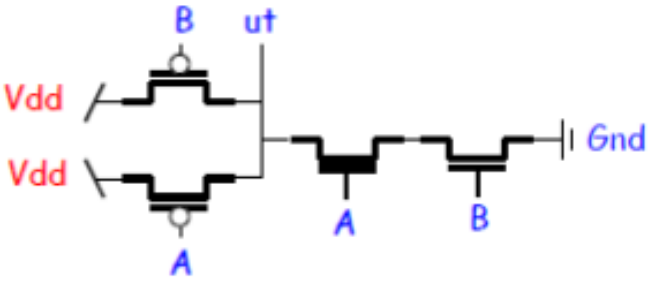
Figur 62: PMOS

10.3 CMOS-kretser



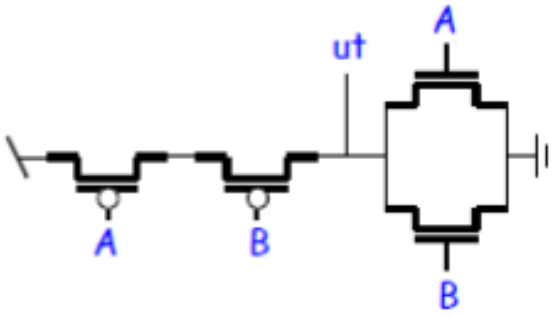
Figur 63: Inverter

AB	ut
00	1
01	1
10	1
11	0



Figur 64: NAND

AB	ut
00	1
01	0
10	0
11	0

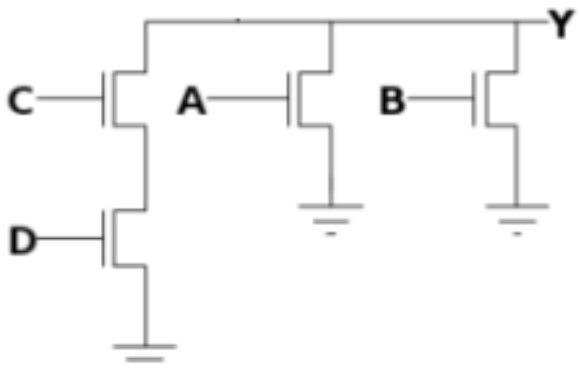


Figur 65: NOR

10.3.1 Fra funksjon til krets

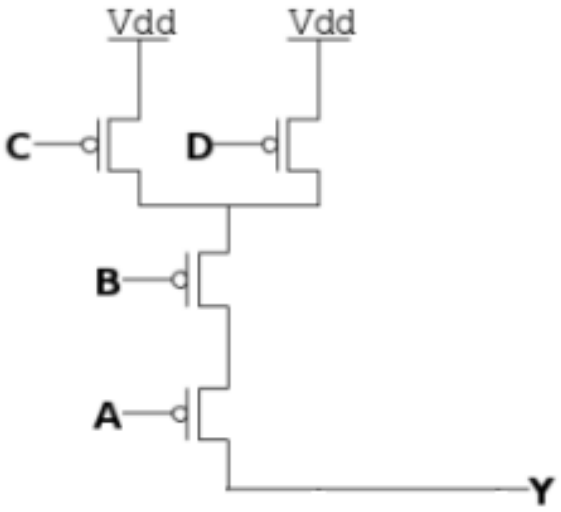
$$Y = \overline{((A+B)+(C*D))}$$

1. Først starter man med å lage nedtrekket (delen av kretsen som er koblet til GND). Man ønsker å ha mest logikk nærmest kildene.



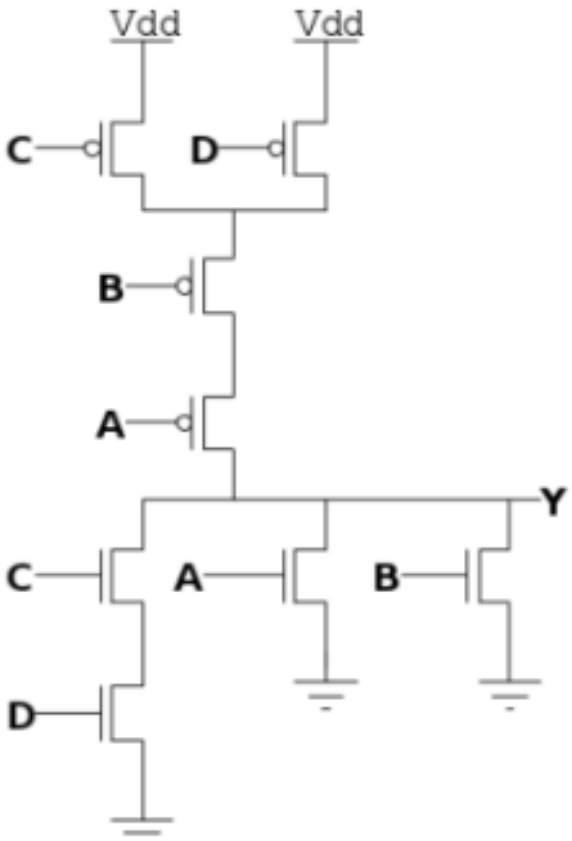
Figur 66: Nedtrekk

2. Ta komplementet av nedtrekket.
   - Parallell $\rightarrow$ seriell
   - Seriell $\rightarrow$ parallell



Figur 67: Opptrekk

3. Ferdig krets.

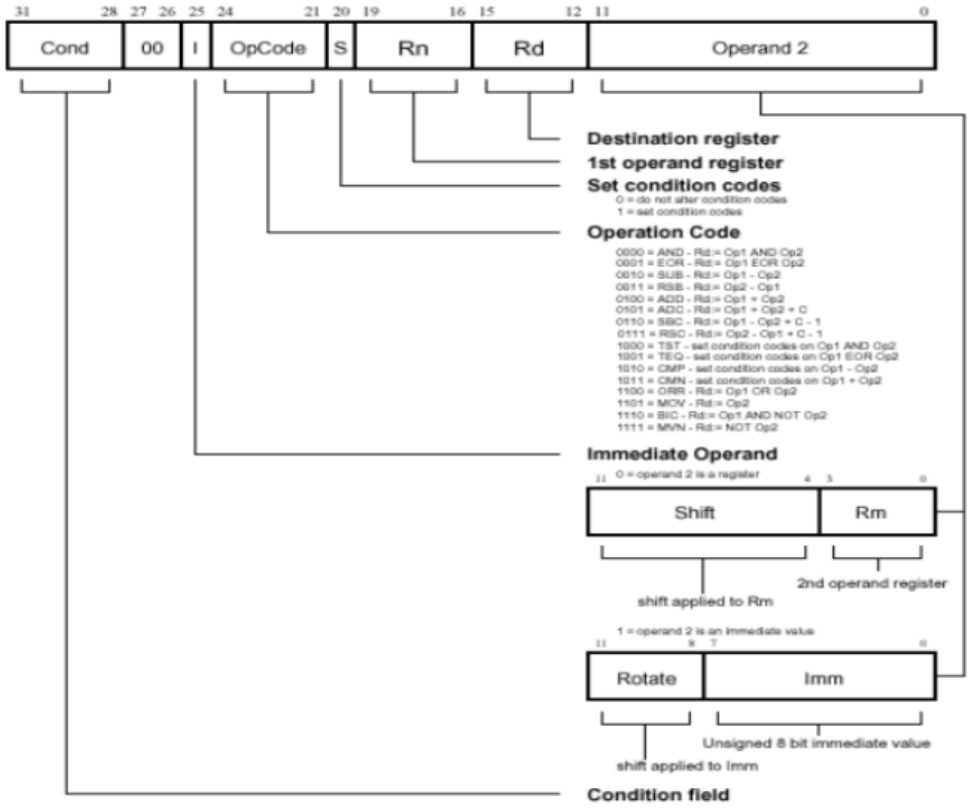


Figur 68: Y = $\overline{((A+B)+(C*D))}$

11 Datamaskinarkitektur

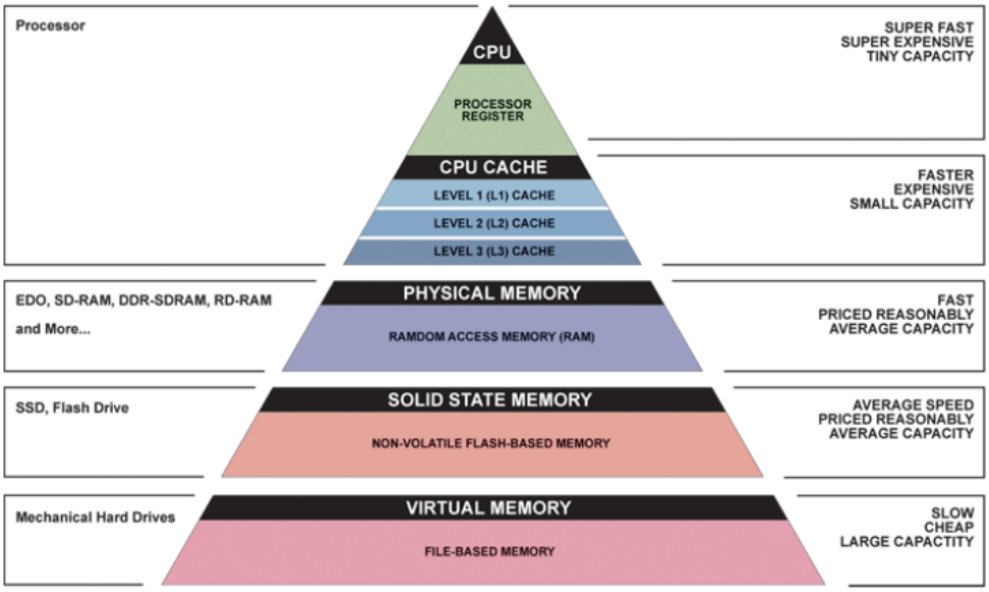
11.1 Maskinkode

Programmer som kjører på PCen vil på et lavt nivå utføres ved hjelp av maskinkode. Denne bitstringen vil deretter oversettes til instruksjoner og utføres sekvensielt i CPUen.



Figur 69: Maskinkode

11.2 Minne



Figur 70: Minne

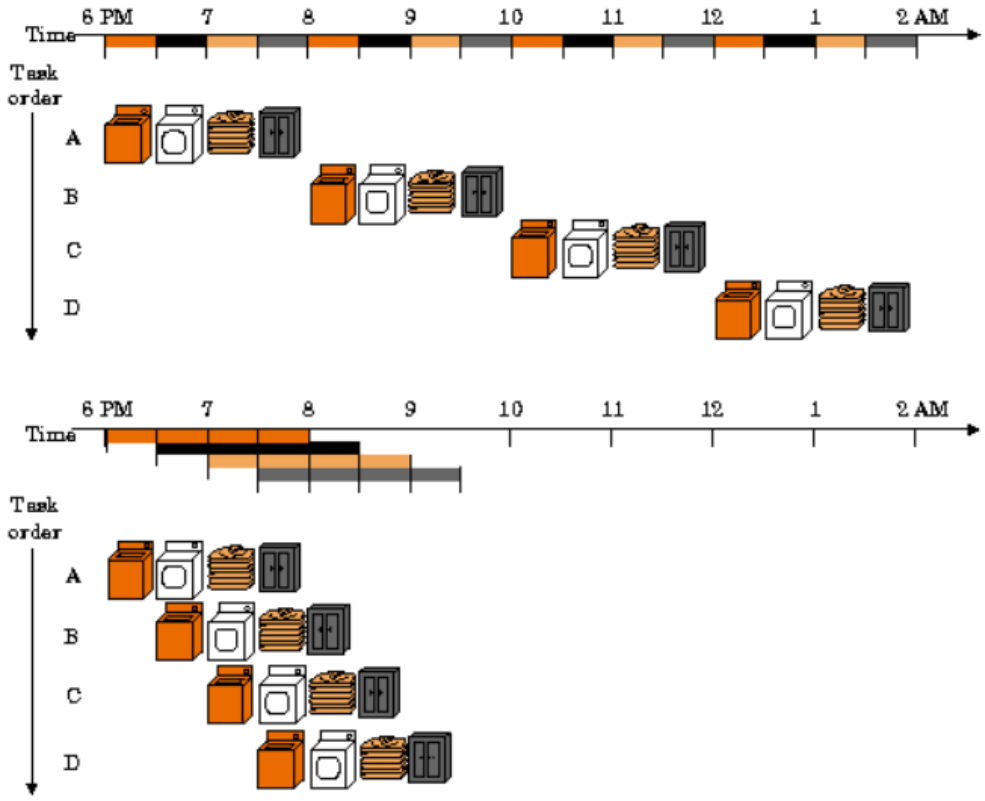
11.3 Pipelining

- 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 instruksjoner i et gitt tidsrom!

11.3.1 Analogi

Vasking av klær

1. Vaske i vaskemaskin
2. Trøke i tørketrommel
3. Brette sammen
4. Sette dem i skapet

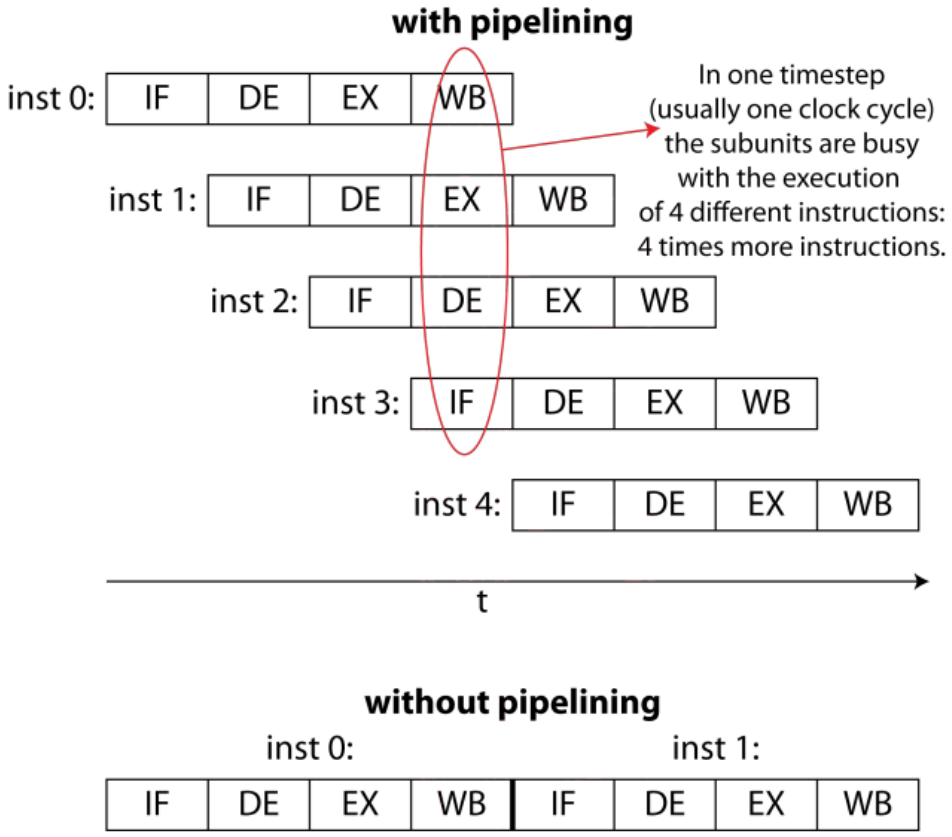


Figur 71: Pipelining

11.3.2 Pipelining instruksjonssett

Eksempelvis kan vi ha følgende subinstruksjoner i 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)



Figur 72: Pipelining

11.3.3 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 instruksjoner.

11.3.4 Komplikasjoner ved pipelining (Hazards)

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. Structural/Resource hazard
   Forekommer dersom fordi noe av prosessorens maskinvare er nødvendig for to eller flere instruksjoner på samme tid. Disse ressursene er altså ikke tilgjengelig fordi de brukes et annet sted.

2. Data hazard
   Forekommer dersom man prøver å skrive/lese til til samme minnelokasjoner.

3. Control Hazard
   Forekommer ved forgreining. Prosessoren vet ikke utfallet til en grein når det er behov for å sette inn en ny instruksjon.