

INF 1400 - Oppsummering

Dawid Kuleczko

December 7, 2016

Contents

1	Digital representasjon og binær logikk:	3
1.1	Tallsystemer:	3
1.2	Konvertering:	3
1.2.1	Konvertering fra grunntall r til desimaltall:	3
1.2.2	Konvertering av desimal til binær:	4
1.2.3	Konvertering av kommatall i desimal til binær :	4
1.2.4	Konvertering fra desimal til grunntall “r”:	4
1.2.5	Sannhetstabell:	5
1.3	Huntington’s postulater	7
2	Boolsk algebra:	7
2.1	Boolske funksjoner med sannhetstabell	9
2.2	Forenkling av uttrykk	9
2.3	Maksterm/Minterm	10
2.3.1	Minterm:	11
2.3.2	Maksterm:	11
2.4	Designprosedyre	12
3	Karnaugh diagram:	12

4	Kobinatorisk logikk:	14
4.1	Binær addisjon:	14
4.2	Binær substraksjon:	15
4.3	Binære addere:	16
4.3.1	Halvadder:	16
4.3.2	Fulladder:	16
4.3.3	Et adder system	17
4.4	"Carry Lookahead"	18
4.5	Komparator:	18
4.6	Dekoder:	20
4.7	Enkoder:	20
4.8	Multiplekser (MUX):	21
4.9	Demultiplekser:	22
4.10	Aritmetisk logisk enhet (ALU):	22
5	Sekvensiell logikk:	23
5.1	Logisk dybde	23
5.2	Latcher(låsekretser):	24
5.3	Flip-Floper:	26
6	Tilstandsmaskiner:	33
6.1	Tilstandstabell	34
6.2	Tilstandsdiagram	35
6.3	Reduksjon av antall tilstander	37
6.4	Tilordning av tilstandskoder	39
6.5	Ubrukte tilstander	40
6.6	Designprosedyre(basert på D flip-floper)	41
6.7	Eksempel	41
7	Datamaskinarkitektur	41
7.1	Pipelining	41
7.1.1	ERROR 404 - Pipeline	42

8	CMOS - Komplementær metalloksidhalvleder	42
8.1	Transistor som bryter	42
8.1.1	nMOS	42
8.1.2	pMOS	42
8.1.3	pMOS + nMOS	43
8.2	MOS transistorer	43
8.3	Fakta og regler	44
8.4	Serie og parallelkobling	45
8.5	Komplementær logikk	47

1 Digital representasjon og binær logikk:

1.1 Tallsystemer:

I dette kurset skal vi legge spesielt vekt på binære tall, men heksadesimale tall og octale tall, samt tall fra andre tallsystemer, kan også forekomme.

Til daglig bruker vi desimal, altså 10-tallsystemet. Et desimalt tall er representert ved symbolene fra 0 til 9.

$$\text{eks. } (951)_{dec} = 9 \cdot 10^2 + 5 \cdot 10^1 + 1 \cdot 10^0$$

Et binært tall er representert ved symbolene 0 og 1.

$$\text{eks. } (101)_{bin} = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

Et heksadesimalt tall er representert ved symbolene fra 0 til F der tallene etter 9 er: 10 = A 11 = B 12 = C 13 = D 14 = E 15 = F

$$\text{eks. } (5BF)_{hex} = 5 \cdot 16^2 + 11 \cdot 16^1 + 15 \cdot 16^0$$

Et oktalt tall er representert ved symbolene fra 0 til 8.

$$\text{eks. } (853)_{oct} = 8 \cdot 8^2 + 5 \cdot 8^1 + 3 \cdot 8^0$$

1.2 Konvertering:

1.2.1 Konvertering fra grunntall r til desimaltall:

Generelt: $(...a_2a_1a_0, a_{-1}a_{-2}...) = ... + a_2 \cdot r^2 + a_1 \cdot r^1 + a_0 \cdot r^0, a_{-1} \cdot r^{-1} a_{-2} \cdot r^{-2} + ..$

$$\text{eks. } (1A5, 1C)_{16} = 1 \cdot 16^2 + A \cdot 16^1 + 5 \cdot 16^0 + 1 \cdot 16^{-1} + 12 \cdot 16^{-2} = (421, 109375)_{10}$$

1.2.2 Konvertering av desimal til binær:

Eks. $(23)_{10}$

$$23/2 = 11 + 1/2 \quad a_0 = 1$$

$$11/2 = 5 + 1/2 \quad a_1 = 1$$

$$5/2 = 2 + 1/2 \quad a_2 = 1$$

$$2/2 = 1 + 0/2 \quad a_3 = 0$$

$$1/2 = 0 + 1/2 \quad a_4 = 1$$

Vi leser det binære tallet nedfra $(10111)_2$

1.2.3 Konvertering av kommatall i desimal til binær :

Eks. $(0.3125)_{10}$

$$0.3125 * 2 = 0.625 \quad a_{-1} = 0$$

$$0.625 * 2 = 1.25 \quad a_{-2} = 1$$

$$0.25 * 2 = 0.5 \quad a_{-3} = 0$$

$$0.5 * 2 = 1.0 \quad a_{-4} = 1$$

Vi leser det binære tallet oppfra $(0.0101)_2$

1.2.4 Konvertering fra desimal til grunntall "r":

Samme måte som over bare med "r" isteden for 2.

Eks. $(12)_{10}$ til 3-tallsystemet:

$$12/3 = 4 + 0/3 \quad a_0 = 0$$

$$4/3 = 1 + 1/3 \quad a_1 = 1$$

$$1/3 = 0 + 1/3 \quad a_2 = 1$$

Tallet blir da $(110)_3$ i 3 tallsystemet.

1.2.5 Sannhetstabell:

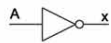






Logic Gates																																																																																																							
Name	NOT	AND	NAND	OR	NOR	XOR	XNOR																																																																																																
Alg. Expr.	\overline{A}	AB	\overline{AB}	$A + B$	$\overline{A + B}$	$A \oplus B$	$\overline{A \oplus B}$																																																																																																
Symbol																																																																																																							
Truth Table	<table><tr><th>A</th><th>X</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	X	0	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	1	0	1	1	1	0	1	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	1	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	1
A	X																																																																																																						
0	1																																																																																																						
1	0																																																																																																						
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					

Figure 1: Logiske porter

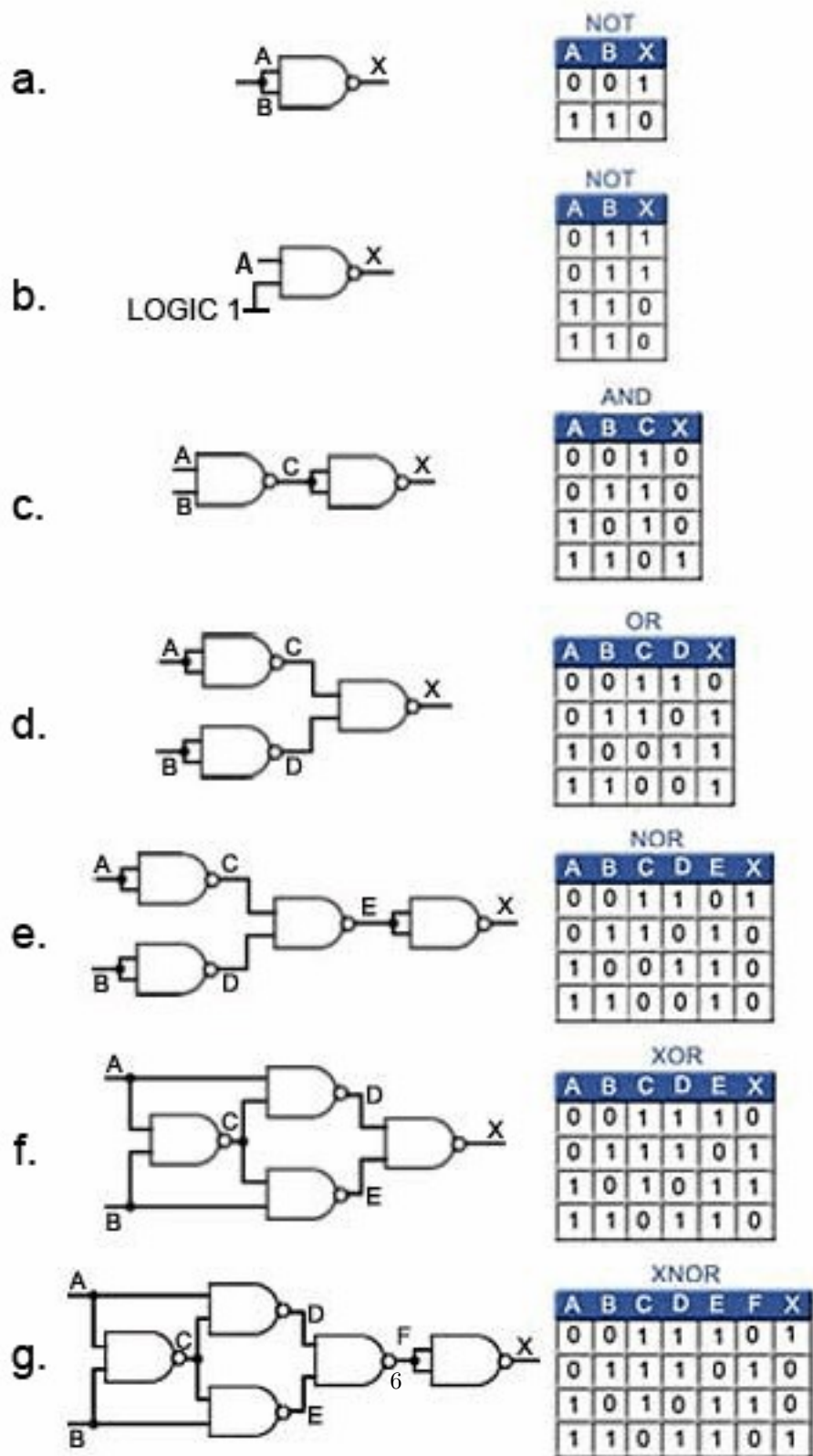


Figure 2: Logiske porter som 2 inputs NAND-porter

1.3 Huntington's postulater

Huntington postulater

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

Dualitet for postulaterne:

Kan bytte "+" med \cdot hvis man bytter "0" med "1"

Presedens:

Først utføres "0", så "+", så "•" og til slutt "+"

Figure 3: Huntington's postulater

2 Boolsk algebra:

DeMorgans teorem

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

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

På invertert form:

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

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

Figure 4: DeMorgans teorem

Komplement av funksjon

Inverterer begge sider og bruker DeMorgan

Eksempel:

$$F = x'yz' + x'y'z$$

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

$$F' = (x'yz')'(x'y'z)'$$

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

Eksempel:

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

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

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

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

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

Figure 5: Komplement av funksjon

Regneregler - oversikt

$$x + 0 = x$$

$$x + x' = 1$$

$$x + y = y + x$$

$$x + (y+z) = (x+y) + z$$

$$x(y+z) = xy + xz$$

$$x + x = x$$

$$x + 1 = 1$$

$$x + xy = x$$

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

$$x \cdot 1 = x$$

$$xx' = 0$$

$$xy = yx$$

$$x(yz) = (xy)z$$

$$x + (yz) = (x+y)(x+z)$$

$$x \cdot x = x$$

$$x \cdot 0 = 0$$

$$x(x+y) = x$$

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

Figure 6: Regne regler for boolsk algebra

2.1 Boolske funksjoner med sannhetstabell

En boolsk funksjon kan visualiseres i en sannhetstabell. En gitt funksjon har kun en sannhetstabell. Men, en gitt sannhetstabell har uendelig mange funksjonsuttrykk.

Eksempel: $F = x + y'z$

XYZ	F
000	0
001	1
010	0
011	0
100	1
101	1
110	1
111	1

2.2 Forenkling av uttrykk

En funksjon kan forenkles ved regneregler for å gjøre den lettere å håndtere og implementere.

Forenklingseksempler

Eksempel:

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

$$F = xx' + xy$$

$$F = 0 + xy$$

$$F = xy$$

Eksempel:

$$F = x + x'y$$

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

$$F = 1(x + y)$$

$$F = x + y$$

Eksempel:

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

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

$$F = x + 0$$

$$F = x$$

Figure 7: Forenklingseksempler av noen funksjoner

Forenklingseksempler II

Eksempel:

$$F = xy + x'z + yz$$

$$F = xy + x'z + yz(x + x')$$

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

$$F = xy(1 + z) + x'z(1 + y)$$

$$F = xy + x'z$$

Eksempel:

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

$$F = (x + y)(x' + z) \quad \text{Dualitet}$$

Figure 8: Forenklingseksempler av noen funksjoner

2.3 Maksterm/Minterm

Variable			Minterm		Maxterm	
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

Figure 9: Tabell for minterm og maksterm

Mintermer har notasjon m_x

$$F(x, y, z) = \sum (m_3, m_6) = \sum (3, 6) = x'yz + xyz'$$

Makstermer har notasjon M_x

$$F(x, y, z) = \Pi(M_3, M_6) = \Pi(3, 6) = (x + y' + z')(x' + y' + z)$$

2.3.1 Minterm:

I en funksjon kan en binær variabel x opptre som x eller x' . En funksjon kan være gitt på “sum av produkt” form. 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.

Hvis man generer en funksjon ut i fra sannhetstabellen får man en sum av mintermer

$$\text{Eksempel: } F = x'y'z + xy'z' + xyz'$$

XYZ	F
000	0
001	1
010	0
011	0
100	1
101	0
110	1
111	0

En sannhetstabell kan sees på som en liste av mintermer.

2.3.2 Maksterm:

En funksjon kan være gitt på “produkt av sum” form.

Eksempel: $F = (x+y)(x+y')y$ Hvert ”summeledd” som inneholder alle variablene kalles 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.

2.4 Designprosedyre

Det er ikke alltid at det enkleste funksjonsuttrykket resulterer i den enkleste port-implementasjonen.

Ved forenkling på portnivå må man vite hvilke porter man har til rådighet, og så justere funksjonsuttrykket mot dette. (håndverk)

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 / forenkler funksjonsuttrykket mot aktuelle porter

3 Karnaugh diagram:

Grafisk metode for forenkling av Boolske uttrykk

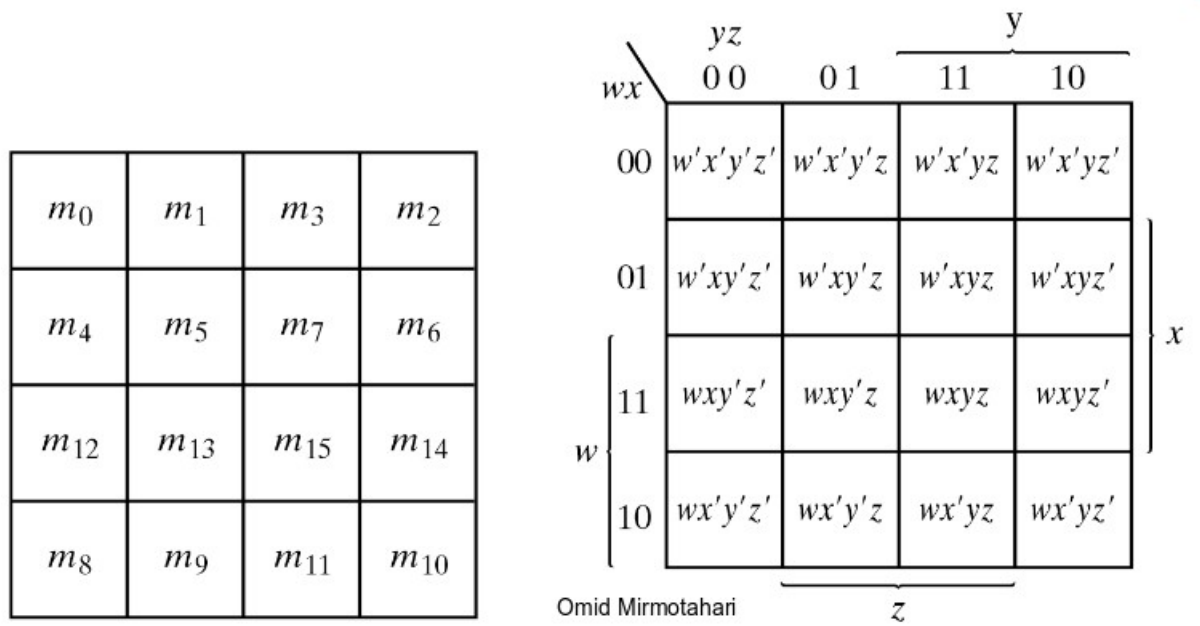
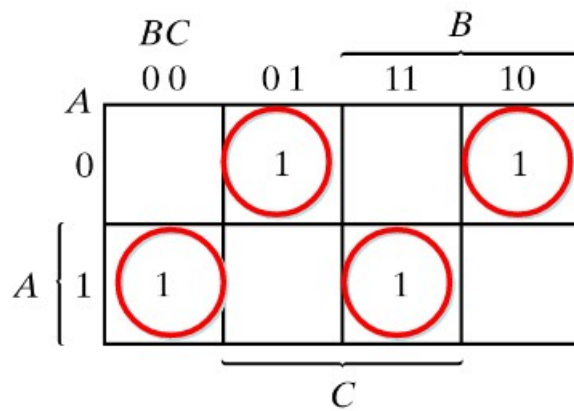


Figure 10: Karnaugh diagram med mintermene



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

Figure 11: XOR

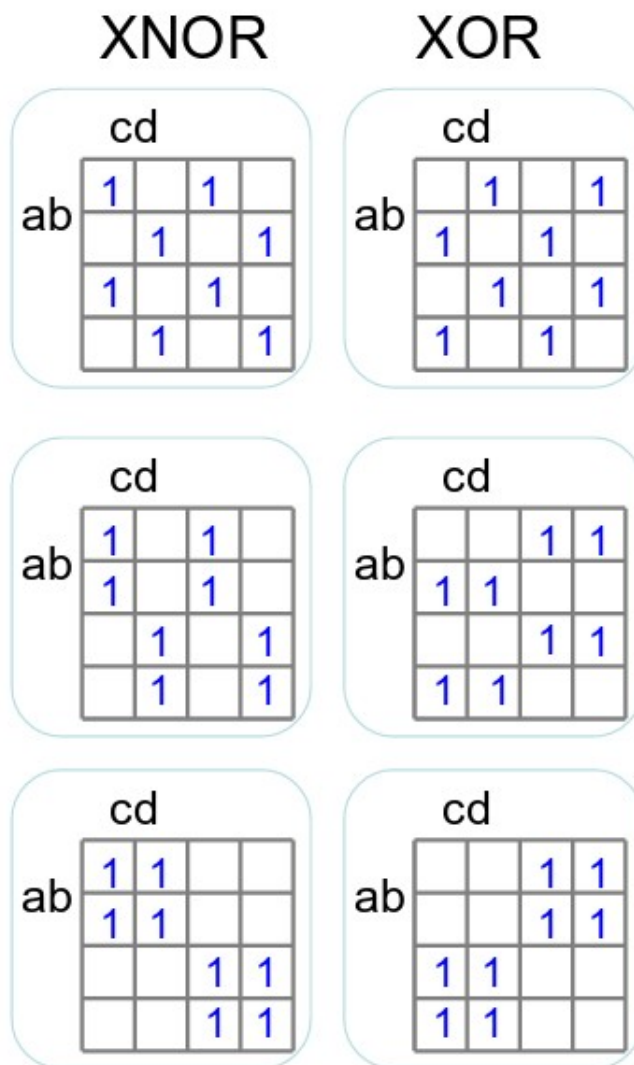


Figure 12: XNOR og XOR

4 Kobinatorisk logikk:

4.1 Binær addisjon:

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

$$\text{Eks: } \left[\begin{array}{r} 0101 \\ 1011 \\ \hline 10000 \end{array} \right] + 5 + 11 = 16$$

4.2 Binær substraksjon:

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

Figure 13: Representasjon av binære negative tall

For å subtrahere negative binære tall, bruker man toerkomplement metoden.

Det tallet som skal subtraheres med må inverteres og plusses på 1, deretter skal det plusses med den andre tallet. Tallet til overs går ut.

Eks:
$$\left[\begin{array}{r} 1101 \\ 1011 \\ \hline \end{array} \right] - \left[\begin{array}{r} 1101 \\ 0101 \\ \hline (1)0010 \end{array} \right] + 13-11 = 2$$

$13+5=18 = 1\ 0\ 0\ 1\ 0$

4.3 Binære addere:

4.3.1 Halvadder:

Halvaddere tar ikke mente inn.

Halvadder implementasjon

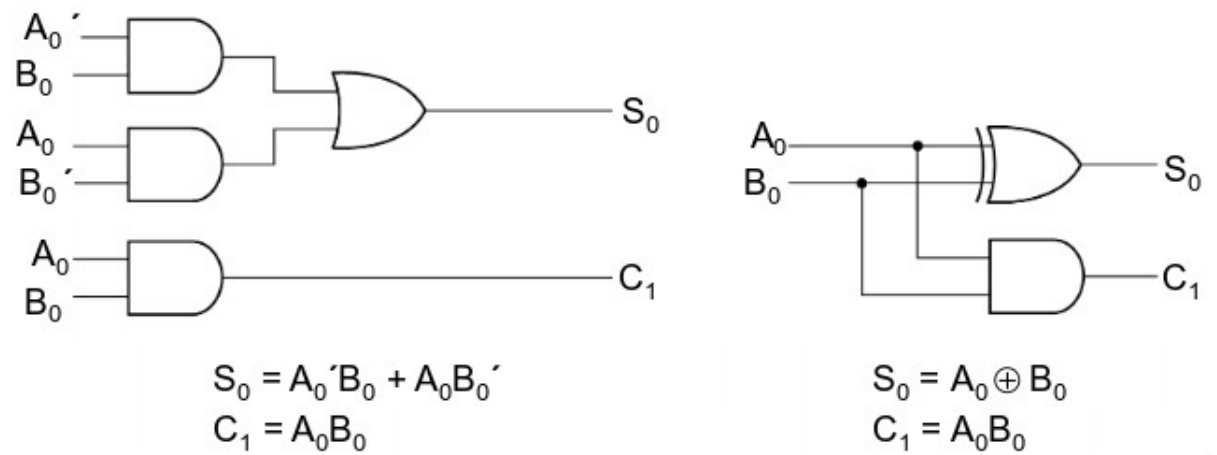


Figure 14: Halvadder implementasjon

4.3.2 Fulladder:

Fulladder tar mente inn.

Fulladder implementasjon

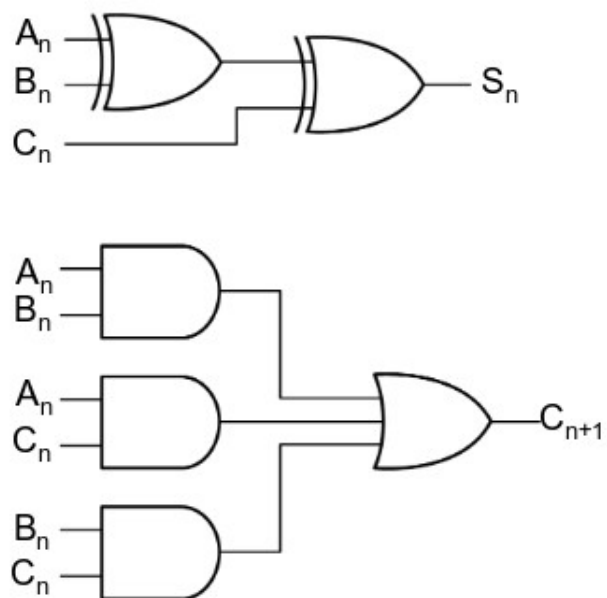


Figure 15: Fulladder implementasjon

4.3.3 Et adder system

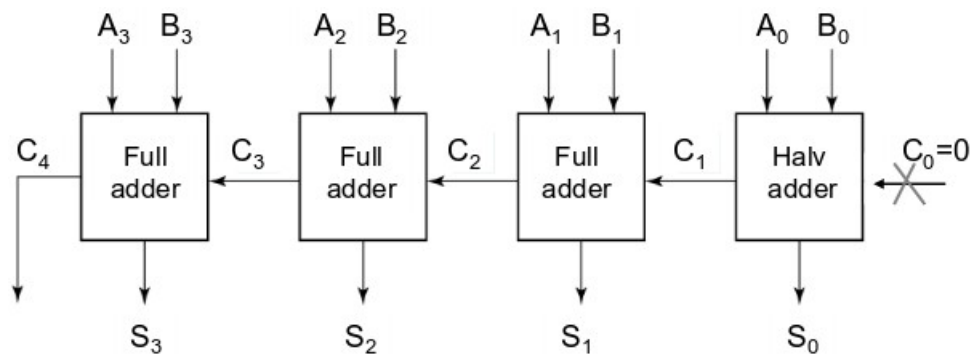


Figure 16: Et system av halv og fulladdere

Adderer 0101 og 1011

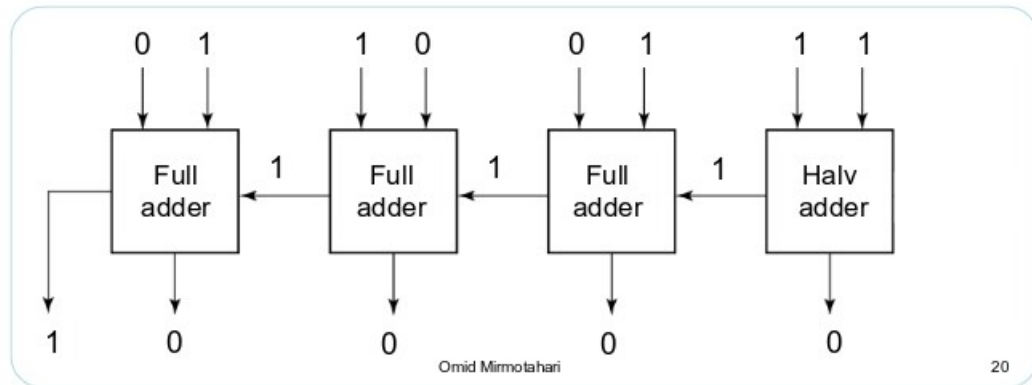


Figure 17: Et eksempel på addisjon av to binære 4-bits tall

4.4 "Carry Lookahead"

Ønsker å unngå menteforplantning – gir økt hastighet

G_i – generate: brukes i menteforplantningen P_i – propagate: påvirker menteforplantningen

4.5 Komparator:

Komparator – sammenligner to tall A og B

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

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

Figure 18: A=B

Utgang $A > B$ slår til hvis:

$(A_3 > B_3)$ eller

$(A_2 > B_2 \text{ og } A_3 = B_3)$ eller

$(A_1 > B_1 \text{ og } A_2 = B_2 \text{ og } A_3 = B_3)$ eller

$(A_0 > B_0 \text{ og } A_1 = B_1 \text{ og } A_2 = B_2 \text{ og } A_3 = B_3)$

Kan skrives:

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

Figure 19: $A > B$

Utgang $A < B$ slår til hvis:

$(A_3 < B_3)$ eller

$(A_2 < B_2 \text{ og } A_3 = B_3)$ eller

$(A_1 < B_1 \text{ og } A_2 = B_2 \text{ og } A_3 = B_3)$ eller

$(A_0 < B_0 \text{ og } A_1 = B_1 \text{ og } A_2 = B_2 \text{ og } A_3 = B_3)$

Kan skrives:

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

Figure 20: $A < B$

4.6 Dekoder:

Dekoder – tar inn et binært ord og gir ut alle mintermer Kan generere generelle logiske funksjoner direkte fra mintermene på utgangen Eksempel: 3bit inn/8bit ut

Eksempel: 3bit inn

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

Figure 21: Sannhetstabell til en 3 bits decoder

4.7 Enkoder:

Enkoder er motsatt av dekode

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

Figure 22: Sannhetstabell for en 8x3 enkoder

Prioritets-enkoder:

Hvis flere "1" ere inn - ser kun på inngang med høyst indeks (prioritet)

4.8 Multiplekser (MUX):

Velger hvilke innganger som slippes ut

Hver inngang kan bestå av **ett** eller **flere** bit

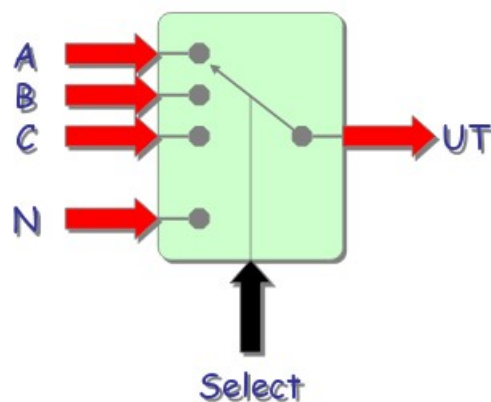


Figure 23: MUX

Eksempel: 2-1 MUX

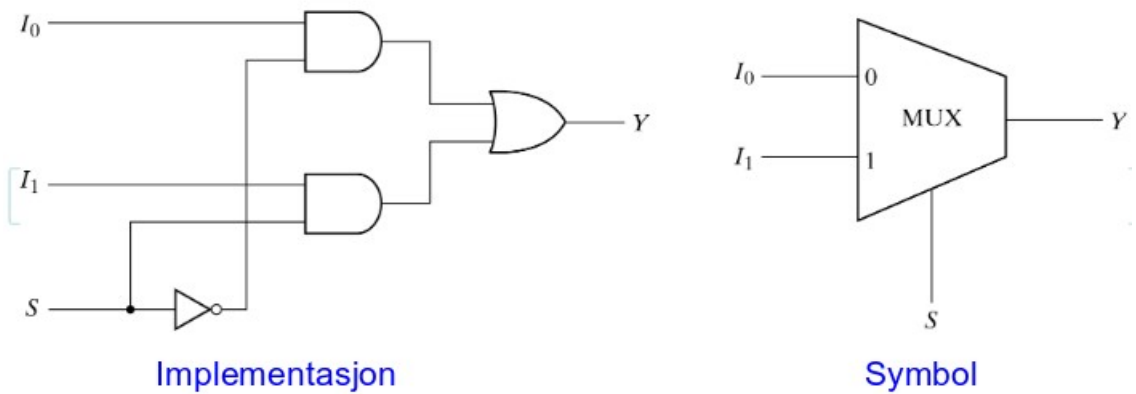


Figure 24: En 2-1 MUX

4.9 Demultiplekser:

Motsatt av MUX, velger hvilke utganger som slippes ut

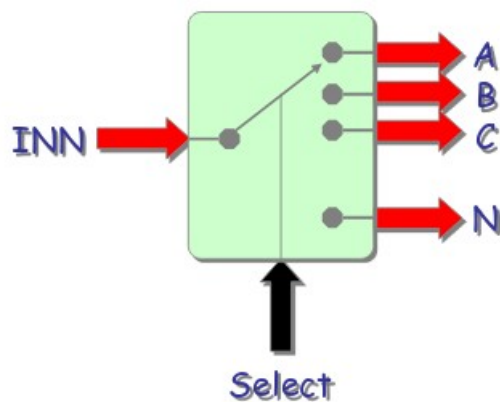


Figure 25: Demultiplekser

4.10 Aritmetisk logisk enhet (ALU):

En elektronisk krets som utfører aritmetiske og logiske operasjoner

5 Sekvensiell logikk:

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 av tidligere inngangs-/utgangsverdier.

Mekaniske brytere gir ikke “rene” logiske nivå ut i overgangsfasen. Slike signaler må ofte “renses” ved bruk av låsekretser.

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.

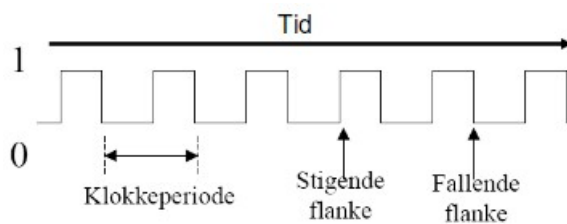


Figure 26: Flanker

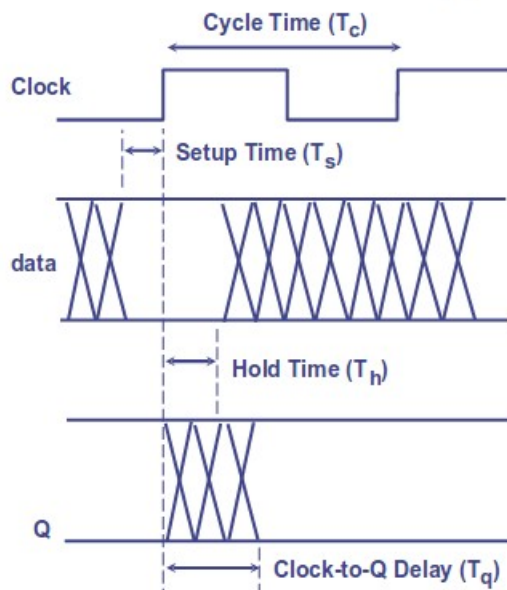
Ønsker så høy klokkefrekvens som mulig, for at hver enkelt operasjon bruker så kort tid som mulig

Maksimal klokkefrekvens bestemmes av: - Lengde på singalveiene - Last - Forsinkelsene gjennom porter(delay) - Teknologi

5.1 Logisk dybde

Antall porter et singal passerer fra inngang til utgang, ved å redusere dette minsker forsinkelsen gjennom kretsen.

1-fase klokking



T_c = klokkeperioden

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

T_h = tiden etter klokkeflanker hvor inngangssignalet må fortsatt være stabil

T_q = tiden det tar fra klokkeflanken til utgangen er klar

Figure 27: Klokking

5.2 Latcher(låsekretser):

De ulike latchene:

SR - Latch

Set Reset - Latch

Setter Q til "1" hvis den får "1" på inngang S.

Når inngang S går tilbake til "0" skal Q forbli på "1"

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"

Tilstanden "1" på både S og R brukes normalt ikke

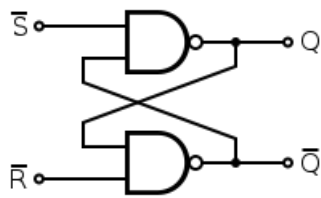


Figure 28: SR-Latch med NAND

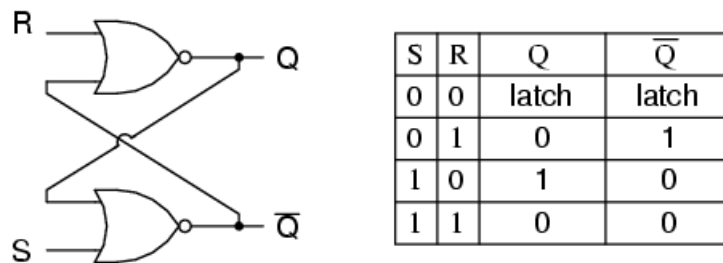


Figure 29: SR-Latch med 2 NOR

D - Latch

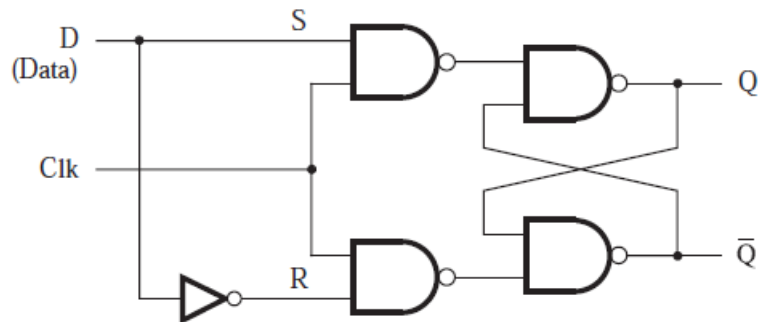
Data - Latch

Dataflyten gjennom en D-latch kontrolleres av et klokkesignal

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

I det øyeblikket klokkeinngangen går fra “1” til “0” låser utgangen seg på sin nåværende verdi.

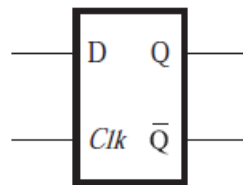
Forandringer på inngangen vil ikke påvirke utgangsverdien så lenge klokkesignalet er “0”



(a) Circuit

Clk	D	$Q(t+1)$
0	x	$Q(t)$
1	0	0
1	1	1

(b) Characteristic table



(c) Graphical symbol

Figure 30: D-Latch

5.3 Flip-Flop:

Et 1-bits hukommelseselement som kan lagre ett bit kalles en flip-flop

En ny verdi kan bare leses inn og lagres når klokkesignalet går fra 0 til 1 (eller 1 til 0, avhengig av konstruksjonen)

En flip-flop'er har enten en eller to innganger (pluss klokkesignal) og en utgang

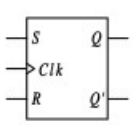
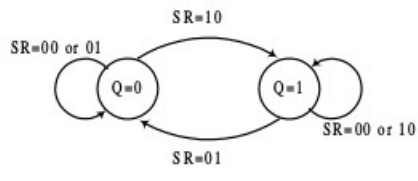
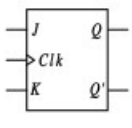
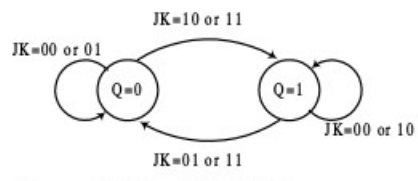
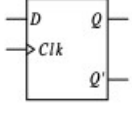
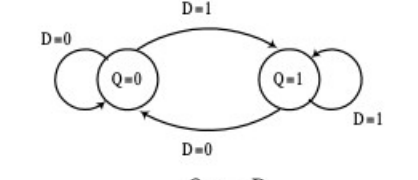
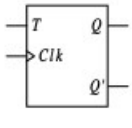
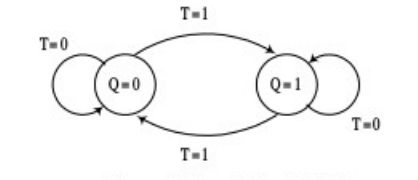
Name / Symbol	Characteristic (Truth) Table	State Diagram / Characteristic Equations	Excitation Table																																																								
SR 	<table><tr><th>S</th><th>R</th><th>Q</th><th>Q_{next}</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td><td>×</td></tr><tr><td>1</td><td>1</td><td>1</td><td>×</td></tr></table>	S	R	Q	Q_{next}	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	×	 $Q_{next} = S + R'Q$ $SR = 0$	<table><tr><th>Q</th><th>Q_{next}</th><th>S</th><th>R</th></tr><tr><td>0</td><td>0</td><td>0</td><td>×</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>×</td><td>0</td></tr></table>	Q	Q_{next}	S	R	0	0	0	×	0	1	1	0	1	0	0	1	1	1	×	0
S	R	Q	Q_{next}																																																								
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	×																																																								
Q	Q_{next}	S	R																																																								
0	0	0	×																																																								
0	1	1	0																																																								
1	0	0	1																																																								
1	1	×	0																																																								
JK 	<table><tr><th>J</th><th>K</th><th>Q</th><th>Q_{next}</th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	J	K	Q	Q_{next}	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	0	 $Q_{next} = J'K'Q + JK' + JKQ'$ $= J'K'Q + JK'Q + JK'Q' + JKQ'$ $= K'Q(J' + J) + JQ'(K' + K)$ $= K'Q + JQ'$	<table><tr><th>Q</th><th>Q_{next}</th><th>J</th><th>K</th></tr><tr><td>0</td><td>0</td><td>0</td><td>×</td></tr><tr><td>0</td><td>1</td><td>1</td><td>×</td></tr><tr><td>1</td><td>0</td><td>×</td><td>1</td></tr><tr><td>1</td><td>1</td><td>×</td><td>0</td></tr></table>	Q	Q_{next}	J	K	0	0	0	×	0	1	1	×	1	0	×	1	1	1	×	0
J	K	Q	Q_{next}																																																								
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	0																																																								
Q	Q_{next}	J	K																																																								
0	0	0	×																																																								
0	1	1	×																																																								
1	0	×	1																																																								
1	1	×	0																																																								
D 	<table><tr><th>D</th><th>Q</th><th>Q_{next}</th></tr><tr><td>0</td><td>×</td><td>0</td></tr><tr><td>1</td><td>×</td><td>1</td></tr></table>	D	Q	Q_{next}	0	×	0	1	×	1	 $Q_{next} = D$	<table><tr><th>Q</th><th>Q_{next}</th><th>D</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	Q	Q_{next}	D	0	0	0	0	1	1	1	0	0	1	1	1																																
D	Q	Q_{next}																																																									
0	×	0																																																									
1	×	1																																																									
Q	Q_{next}	D																																																									
0	0	0																																																									
0	1	1																																																									
1	0	0																																																									
1	1	1																																																									
T 	<table><tr><th>T</th><th>Q</th><th>Q_{next}</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	T	Q	Q_{next}	0	0	0	0	1	1	1	0	1	1	1	0	 $Q_{next} = TQ' + T'Q = T \oplus Q$	<table><tr><th>Q</th><th>Q_{next}</th><th>T</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	Q	Q_{next}	T	0	0	0	0	1	1	1	0	1	1	1	0																										
T	Q	Q_{next}																																																									
0	0	0																																																									
0	1	1																																																									
1	0	1																																																									
1	1	0																																																									
Q	Q_{next}	T																																																									
0	0	0																																																									
0	1	1																																																									
1	0	1																																																									
1	1	0																																																									

Figure 31: Typer Flip-floper

Flip-Flop'er kommer i to varianter:

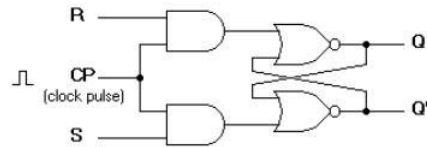
- Positiv flanketrigget
- Negativ flanketrigget

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".

De ulike Flip-Flopene:

SR - Flip-flop Set Reset - Flip-Flop



(a) Logic diagram

Q	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	indeterminate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	indeterminate

(b) Truth table

Clocked SR flip-flop

Figure 32: Kretsoppbygging til en SR-Flip-Flop

JK - Flip-flop J(Set) K(Reset) - Flip-Flop

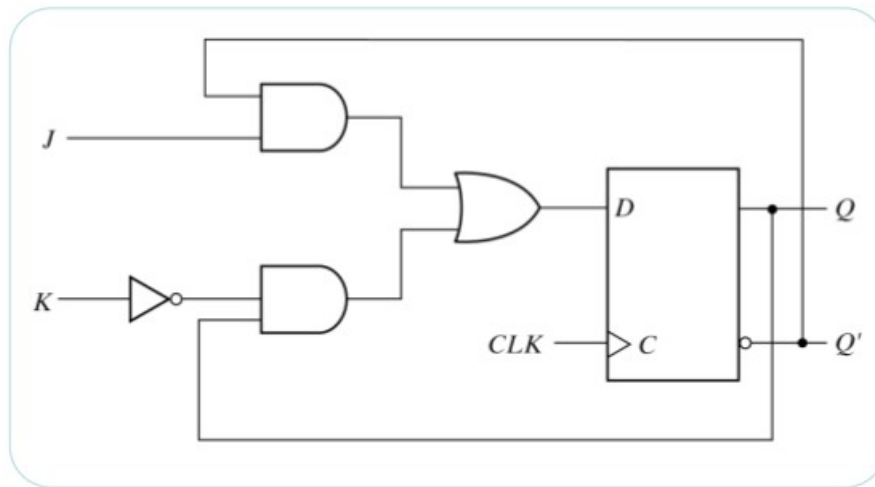
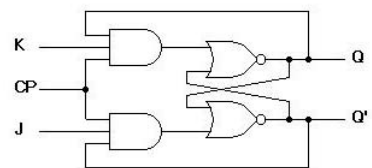
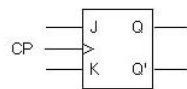


Figure 33: Kretsoppbygging til en JK-Flip-Flop



(a) Logic diagram



(b) Graphical symbol

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

(c) Transition table

Clocked JK flip-flop

Figure 34: Typer Flip-floper

D - Flip-flop Data - Flip-Flop

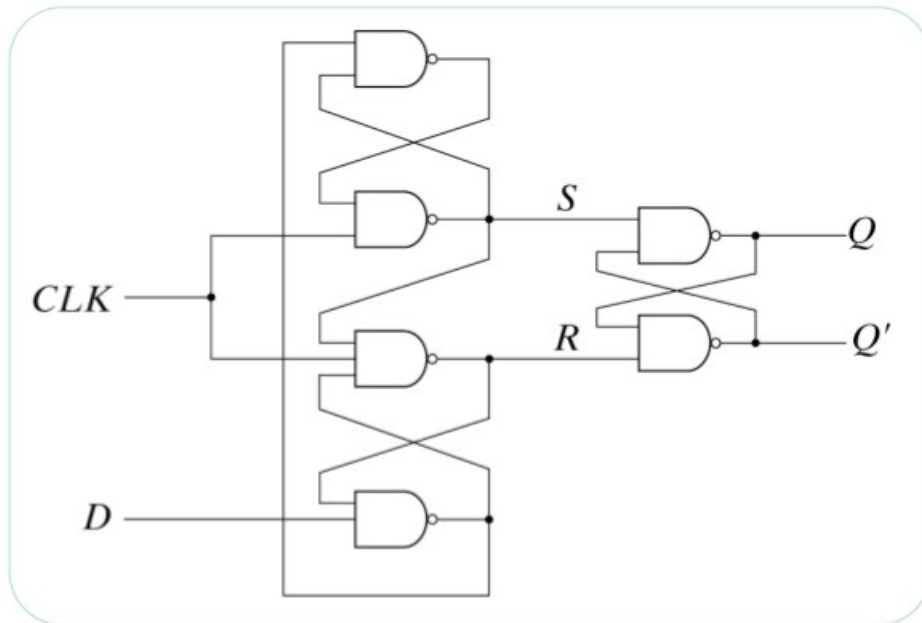
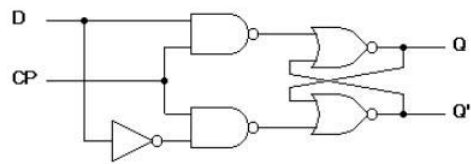
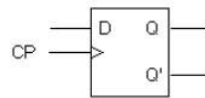


Figure 35: Kretsoppbygning til en D-Flip-Flop



(a) Logic diagram with NAND gates



(b) Graphical symbol

Q	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

(c) Transition table

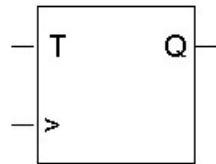
Clocked D flip-flop

Figure 36: Kretsoppbygning til en D-Flip-Flop

T - Flip-flop Toggle - Flip-Flop

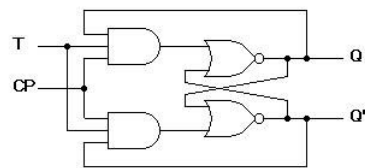
T flip-flop

- ◆ Full name: Toggle flip-flop
 - ⇒ Common in counter design
 - ⇒ Can construct a T from a D and mux
- ◆ When $\text{CLK} \uparrow$
 - ⇒ The output toggles if the input is a "1"
 - ⇒ ...and holds its present value if the input is a "0"

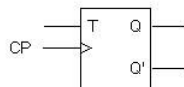


$T(t)$	$Q(t)$	$Q(t + \Delta t)$
0	0	0
0	1	1
1	0	1
1	1	0

Figure 37: T - Flip-Flop



(a) Logic diagram



(b) Graphical symbol

Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

(c) Transition table

Clocked T flip-flop

Figure 38: T - Flip-Flop

6 Tilstandsmaskiner:

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

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

Brukes mye innen:

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

Modellen av en tilstandsmaskin:

- Tilstander

er et begrep som benyttes til å beskrive systemets tilstand.

er et verdsett/attributter som beskriver systemets egenskaper

- Hendelser

endrer systemet fra en tilstand til en annen

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(resultatet) er en respons på en hendelse

6.1 Tilstandstabell

Sannhetstabell for tilstandsmaskiner

Q1	Q0	X2	X1	X0	D1	D0	Out
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	0	0	1	0	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	0	0	0	0
0	0	1	0	1	0	0	0
0	0	1	1	0	0	0	0
0	0	1	1	1	0	0	0
0	1	0	0	0	0	1	0
0	1	0	0	1	0	1	0
0	1	0	1	0	0	0	0
0	1	0	1	1	1	0	0
0	1	1	0	0	0	0	0
0	1	1	0	1	0	0	0
0	1	1	1	0	0	0	0
0	1	1	1	1	0	0	0
1	0	0	0	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	0	0	0	0
1	0	0	1	1	1	0	0
1	0	1	0	0	0	0	0
1	0	1	0	1	0	0	1
1	0	1	1	0	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	0	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	0	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	0	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	1	0	0

Figure 39: Eksempel på en tilstandstabell fra oblig 2

6.2 Tilstandsdiagram

For å visualisere oppførselen til systemer brukes gjerne tilstandsdiagramer

- Sirkler angir tilstander
- Piler angir tilstandsending
- Hendelse og aksjoner settes over piler som angir tilstandsendingen

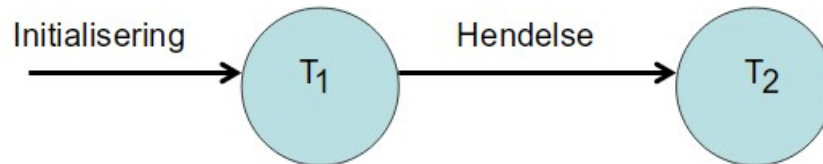


Figure 40: Tilstandsdiagram

- Sirkler angir tilstander
- Piler angir tilstandsending
- Hendelse og aksjoner settes over piler som angir tilstandsendingen

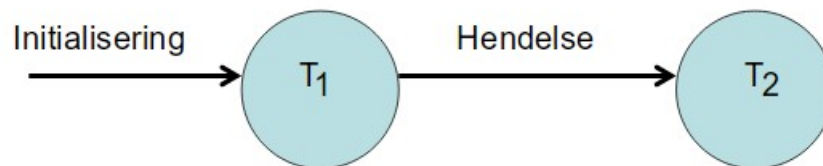


Figure 41: Tilstandsdiagram

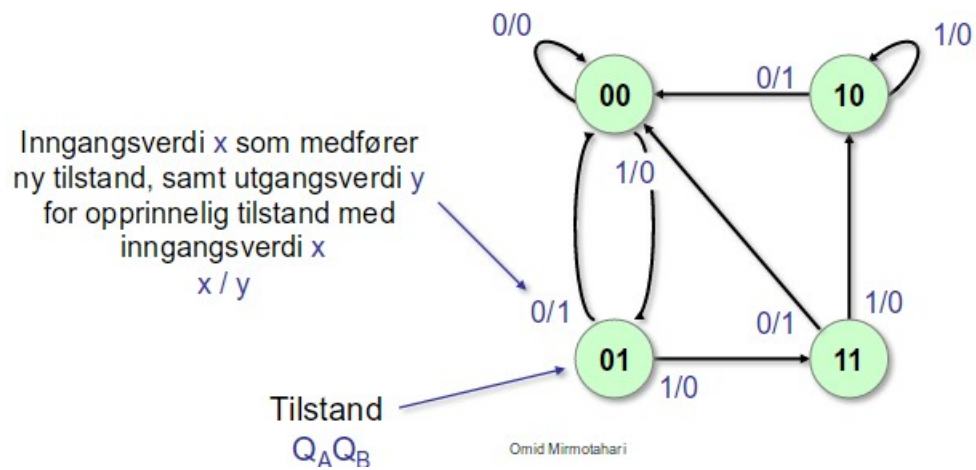


Figure 42: Eksempel på en tilstandsdiagram

6.3 Reduksjon av antall tilstander

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:

Nåværende tilstand		Neste tilstand	
	Inngang		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

Figure 43: Eksempel på reduksjon av tilstander

Vi ser at tilstand G er lik E

Vi kan fjerne tilstand G og erstatte hopp til G med hopp til E

Etter det ser vi at tilstand F blir lik D, da fjerner vi F

Slik kan vi redusere tilstandene.

6.4 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

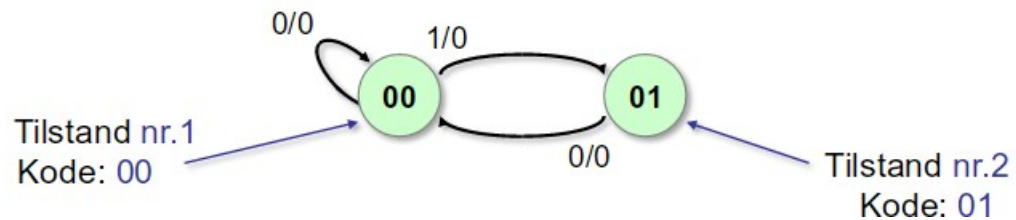
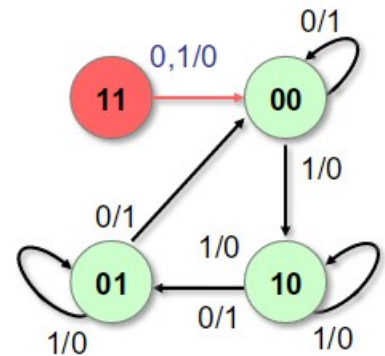


Figure 44: Tilordning av tilstandskoder

6.5 Ubrukte tilstander

I en tilstandsmaskin med N flip-floppe 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.

Figure 45: Ubrukte tilstander

6.6 Designprosedyre(basert på D flip-floper)

- 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

Figure 46: Designprosedyre

6.7 Eksempel

7 Datamaskinarkitektur

7.1 Pipelining

Innfører samlebåndsprinsipp for eksekvering av instruksjoner

Hver instruksjon må splittes opp i uavhengige deler(subinstruksjonener) som utføres etter hverandre

Hver subinstruksjon kan utføres uavhengig av de andre subinstruksjonene

Neste instruksjon settes i gang før forrige instruksjon er helt ferdig.

Hver instruksjon tar like lang tid å utføre, men prosessoren utfører flere instruksjoner i et gitt tidsrom.

En 4-trinns pipeline er den korteste som finnes for en CPU Men det å ha en 4-trinns pipeline betyr ikke at man får 4 ganger raskere prosessering. Det går alltid bort noe tid til administrering av instruksjoner.

7.1.1 ERROR 404 - Pipeline

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, noe som kalles HAZARD

Det er tre typer HAZARD:

-Resource Hazard

-Data Hazard

-Control Hazard

8 CMOS - Komplementær metalloksidhalvleder

8.1 Transistor som bryter

8.1.1 nMOS

En nMOS transistor har tre terminaler; Gate(inngang), Source og Drain. En nMOS transistor kan betraktes som en bryter, avhengig av inngang vil det kunne gå strøm mellom drain og source. Når inngangen er 0 går det ingen strøm mellom drain og source, og vi sier at transistoren er AV. Når inngangen er 1 kan det gå strøm mellom drain og source, og vi sier at transistoren er PÅ.

Lavest spenning - Source

Høyest spenning - Drain

En positiv strøm vil alltid gå fra drain til source.

8.1.2 pMOS

En pMOS transistor har også tre like terminaler akkurat som pMOS. Når inngangen er logisk 0 kan det gå strøm mellom source og drain, og vi sier at transistoren er PÅ. Når inngangen er logisk 1 går det ingen strøm mellom source og drain, og vi sier at transistoren er AV.

Høyest spenning - Source

Lavest spenning - Drain

En positiv strøm vil alltid gå fra source til drain.

8.1.3 pMOS + nMOS

Dersom vi setter en pMOS og en nMOS transistor sammen og kobler til spenningsreferansene V_{DD} og V_{SS} får vi en CMOS inverter.

CMOS teknologi er med andre ord grunnleggende inverterende.

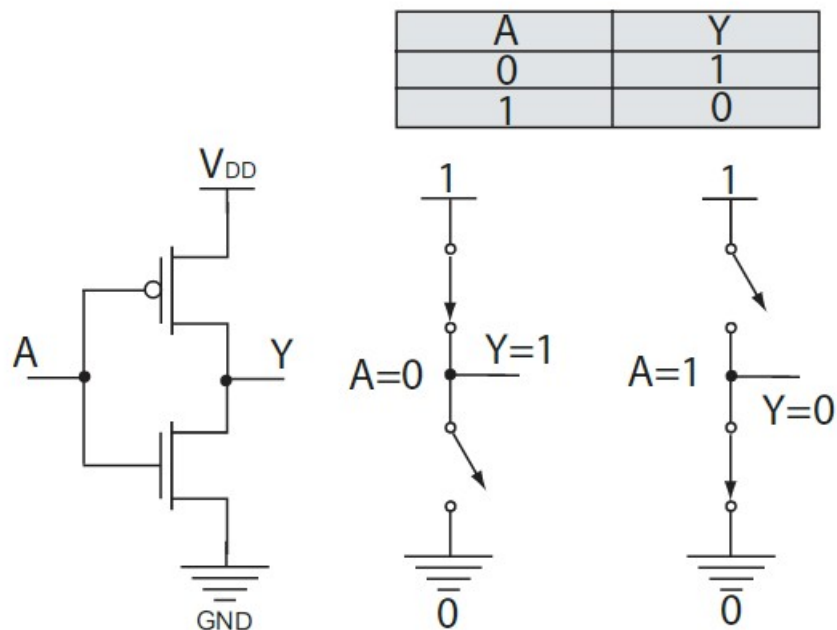


Figure 47: Inverter skjematikk og sannhetstabell

8.2 MOS transistorer

MOSFET - Metal On Semiconductor Field Effect Transistor - er integrerte transistorer.

nMOS - fordi Source og Drain terminalene er koblet til n-type silisium. Disse

områdene kalles for diffusjon og er kraftig dopet med et stort antall frie elektroner. Diffusjonsområdene ligger i en svakt dopet silisium halvleder som kalles substrat. Mellom gates og p-substrat er det et isolerende sjikt som separerer gaten fra substrat slik at det ikke skal gå strøm mellom gate og substrat.

pMOS - fordi Source og Drain terminalene er koblet til p-type silisium. Disse områdene kalles diffusjon og er kraftig dopet med et stort antall frie hull. Ved å sette på en positiv spenning (logisk 1) på gate terminalen vil elektroner tiltrekkes overfalten på halvlederen og invertere p-substrat til n-positiv substrat.

8.3 Fakta og regler

Det er vanlig i digital CMOS å benytte minimumsstørrelser på ulike strukturer, dette medfører et redusert areal og mindre tidsforsinkelse. Liten tidsforsinkelse gir raske kretser som kan fungere med svært høye klokkefrekvenser.

Vi definerer opptreksnettverk og nedtreksnettverk som på dersom det finnes en strømvei mellom utgangen og spenningsreferansen, $V_{dd}(1)$ og $GND(0)$.

Et nedtrekk er PÅ dersom det finnes en serie av nMOS transistorer som alle er PÅ og som forbinder utgangen med GND, i motsatt tilfelle er nedtrekk AV.

For et opptrekk som er PÅ finnes det en serie av pMOS transistorer som alle er PÅ og som forbinder utgangen med V_{dd} , i motsatt tilfelle er opptrekket AV.

I komplementær CMOS logikk vil alltid en og bare en av opptrekk- og nedtreksnettverke være på.

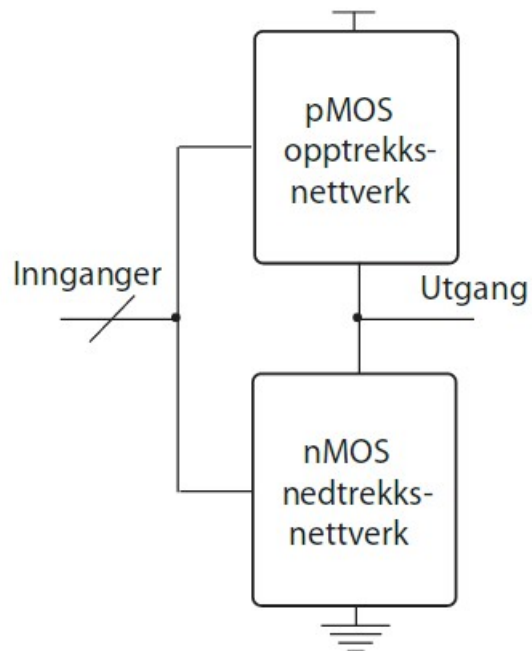


Figure 48: Generell logisk port med opptrekk bestående av pMOS transistorer og nedtrekk bestående av nMOS transistorer

8.4 Serie og parallelkobling

nMOS i serie = NAND dersom GND = 0 på minst en inngang, $A*B$

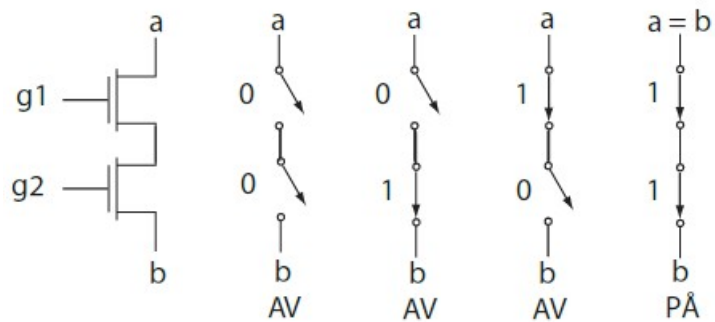


Figure 49: nMOS transistorer i serie

nMOS i parallell = NOR dersom $V_{dd} = 1$ på minst en inngang, $A+B$

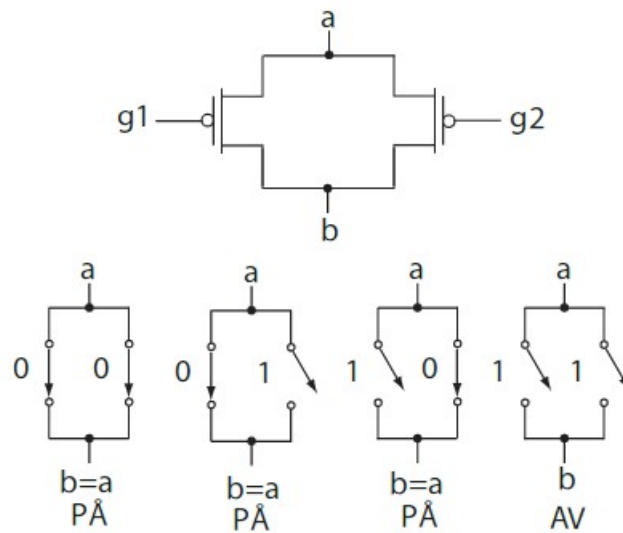


Figure 50: pMOS transistorer i parallellkobling

pMOS i serie = NOR dersom $V_{dd} = 1$ på minst en inngang, $A+B$

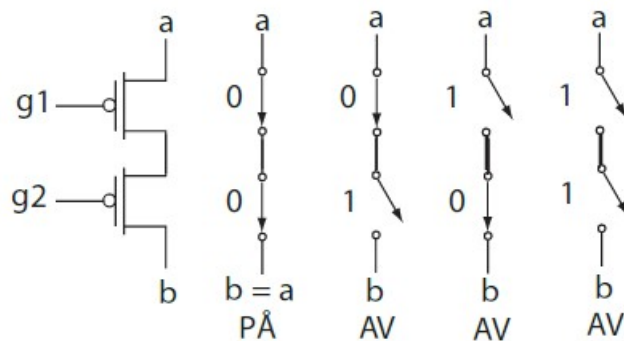


Figure 51: pMOS transistorer i serie

pMOS i parallell = NAND dersom GND = 0 på minst en inngang, $A \cdot B$

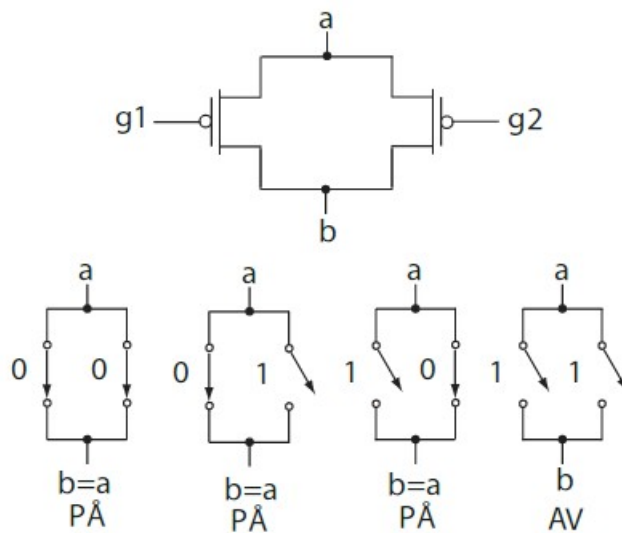


Figure 52: pMOS transistorer i parallellkobling

8.5 Komplementær logikk

Eks på en funksjon implementert ved CMOS $Y = \overline{(A \times B) + (C \times D)}$

Nedtrekket vil bestå av nMOS transistorer og vi har at Y bare kan bli 0 når $(A \times B) + (C \times D) = 1$ Som forutsetter at $A \cdot B$ eller $C \cdot D$ er på. Nedtrekket må

da bestå av to grener med seriekoblede nMOS transistorer.

Opptrekket vi bestå av pMOS transistorer og vi har at Y bare kan bli 1 når $(A \times B) + (C \times D) = 0$ Som forutsetter at A og/eller B ($A \times B = 0$) og C og/eller D ($C \times D = 0$) er PÅ, altså 0. Opptrekket må da bestå av to grener med parallellkoblede pMOS transistorer. Til slutt må disse to parallellgrenen settes i serie slik at forutsetningen for opptrekket blir oppfylt.

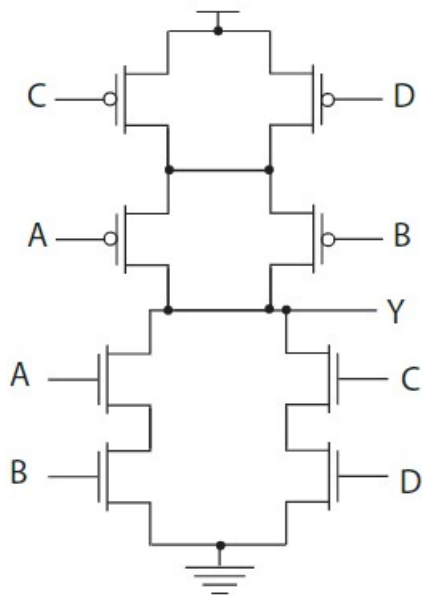


Figure 53: Komplementær CMOS port for funksjonen $Y = \overline{(A \times B) + (C \times D)}$