Løsningsforslag i digitalteknikkoppgaver INF2270 uke 5 (29/1-4/2 2006)

Oppgave 1)
Bør kunne løses rett fram, likevel:

a) E = *abcd* + *a'bc* + *acd* + *bcd*:

ab \ cd	00	01	11	10
00				
01			1	1
11			1	
10			1	

De variablene som **ikke** varierer i grupperingene ANDes sammen og AND-termene ORes så sammen:
E=*a'bc* + *acd*

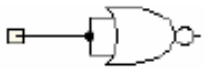
b) F= *xyz* + *x'y* + *wxz*

wx \ yz	00	01	11	10
00			1	1
01			1	
11		1	1	
10			1	1

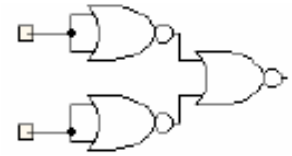
F = *yz* + *x'y* + *wxz*

Oppgave 2)
Observerer at: x
NOR y = (x+y)' = x'y'

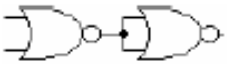
NOT



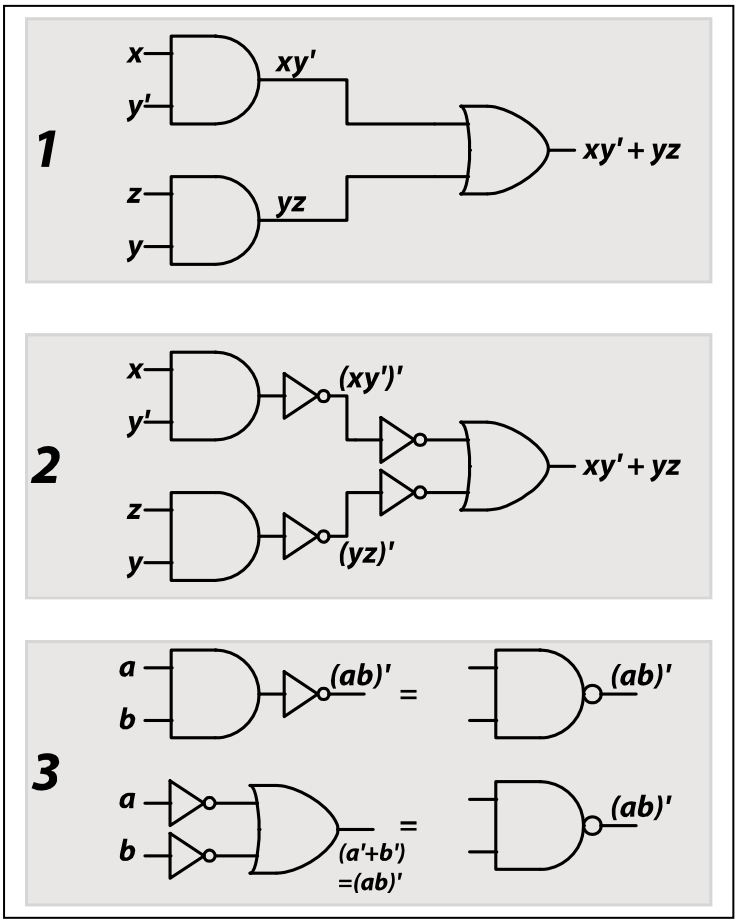
AND



OR



Når det gjelder NAND-implementasjon kan følgende sees på som eksempel:



Vi har:

1. Dette en Sum-av-produkter- eller AND-OR-implementasjon som kan gjøres rett etter utlesning fra Karnaugh-diagram. Dvs. alle termene er produkter (AND-termer) som summères sammen (dvs. OR).
2. Hvis man legger inn to invertere etter hverandre endrer man ikke funksjonen. Synonymt med (a')' = a
3. En AND-port + Inverter er det samme som en NAND. Det er også en OR-port med inverterte innganger. Bruker man DeMorgans teorem ser man også at a'+b'=(ab)'.

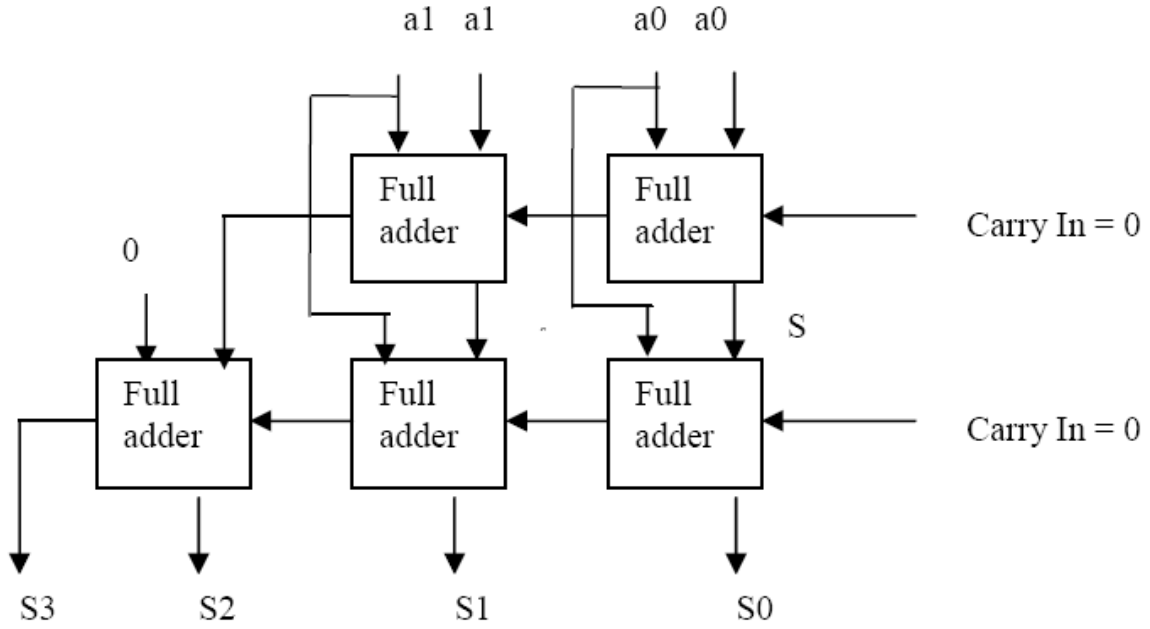
Dette betyr at i alle AND-OR-implementasjoner, hvor man først har et nivå med AND-porter og så et nivå med OR-porter, kan man bytte ut alle portene med NAND. Unntaket er når en term består av bare en variabel og tilsvarende signal i implementasjonen går direkte til OR-porten. Da må dette signalet inverteres. F.eks. for G = a + bcd + ba' + ca så må a-inngangen inverteres.

Oppgave 3)
Sannhetsverditabell:

a_3	a_2	a_1	a_0	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

F kan ikke forenkles (ses ved å sette opp Karnaughdiagram og se at det ikke er mulig å sette sammen rektangler større enn én rute)

Oppgave 4)
Kaller de to input-bitene i tallet a for hhv a1 og a0. Siden a multiplisert med 3 kan ha maks verdi på 9, trengs 4 bit for å representere svaret. Disse bitene kalles S3, S2, S1 og S0.
Ideen er å summere a til seg selv 3 ganger siden 3a = a+a+a



Oppgave 5)
Gis ingen løsningforslag, siden oppgaven vil bli en del av oblig1

Oppgave 6) (utfyllende tekst siden dette ikke er fullstendig forelest tidligere)
Fra tilstandsdiagrammet på slide 10 fra forelesningen 29. januar får man følgende tilstandstabell:

Nåværende tilstand	Neste tilstand		Nåværende utgang	
	Inn = 0	Inn = 1	Inn = 0	Inn = 1
S0	S2	S1	0	0
S1	S1	S0	1	1
S2	S1	S2	0	0

Omskrevet med binære koder for tilstandene; S0 = AB= 00, S1 = AB = 01 og S2 = AB = 10, inngangen kalles ***y***, og utgangen kalles ***F***:

Nåværende tilstand		Inngang	Neste tilstand		Utgang (nå)
A(t)	***B(t)***	***y***	***A(t+1)***	***B(t+1)***	***F***
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	0	0	1
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	X	X	X
1	1	1	X	X	X

X'ene for AB=11 er der fordi tilstand 11 (tilstand S3) ikke eksisterer i diagrammet og dermed ikke forekommer. Det er likevel hensiktsmessig å ta denne tilstanden med siden det er en tilstandskombinasjon og disse kanskje kan brukes til å forenkle de boolske uttrykkene for A, B og F. Videre, hvis man tar hensyn til eksitasjonstabellen til JK-flip-flopen (slide 7 fra forel. 29. jan):

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

Eksitasjonstabellen forteller om hvilke inngangsverdier; J og K, må ha nå for å få en ønsket endring på utgangen, Q(t) → Q(t+1). F.eks. i første linjen hvor Q(t) er 0 og vi ønsker at utgangen skal være 0 i neste klokkeperiode, Q(t+1). Da må J=0 og K er 'Don't Care', det vil si ubetydelig.

Hvis man bruker eksitasjonstabellen og utvider tilstandstabellen får man tabellen under. Merk her hvordan eksitasjonstabellen til JK-FlipFlop'en er brukt for å finne inngangsbetingelsene til flip-flopene, J_A, K_A, J_B, K_B. F.eks. for flip-flop A i første linje, så er A i nåværende tilstand A(t)=0 og A i neste tilstand A(t+1)=1, ut fra eksitasjonstabellen må da J-inngangen på flip-flopen være 1, mens K-inngangen er ubetydelig, altså 'don't care'. Fyller ut tilsvarende for resten av endringene på A og B (kodet med hhv. blått og rødt).

Nåværende tilstand		Inngang	Neste tilstand		Flip-Flop Innganger				Utgang (nå)
***A*(*t*)**	***B*(*t*)**	***Y***	***A*(*t*+1)**	***B*(*t*+1)**	J_A	K_A	J_B	K_B	***F***
0	**0**	0	**1**	**0**	**1**	**X**	**0**	**X**	0
0	**0**	1	**0**	**1**	**0**	**X**	**1**	**X**	0
0	**1**	0	**0**	**1**	**0**	**X**	**X**	**0**	1
0	**1**	1	**0**	**0**	**0**	**X**	**X**	**1**	1
1	**0**	0	**0**	**1**	**X**	**1**	**1**	**X**	0
1	**0**	1	**1**	**0**	**X**	**0**	**0**	**X**	0
1	**1**	0	**X**	**X**	**X**	**X**	**X**	**X**	X
1	**1**	1	**X**	**X**	**X**	**X**	**X**	**X**	X

Videre finner man uttrykk for J_A, K_A, J_B, K_B, samt F ut fra kombinasjonene for *A*(*t*), *B*(*t*) og *y* i tabellen: (her er X'ene, dvs. don't care-betingelsene nyttige)

J_A:

By / ***A***	00	01	11	10
0	1			
1	X	X	X	X

$J_A = B'y'$

K_A:

By / ***A***	00	01	11	10
0	X	X	X	X
1	1		X	X

$J_B = y'$

J_B:

By / ***A***	00	01	11	10
0		1	X	X
1	1		X	X

$J_B = A'y + Ay' = A \oplus y$

K_B:

By / ***A***	00	01	11	10
0	X	X	1	
1	X	X	X	X

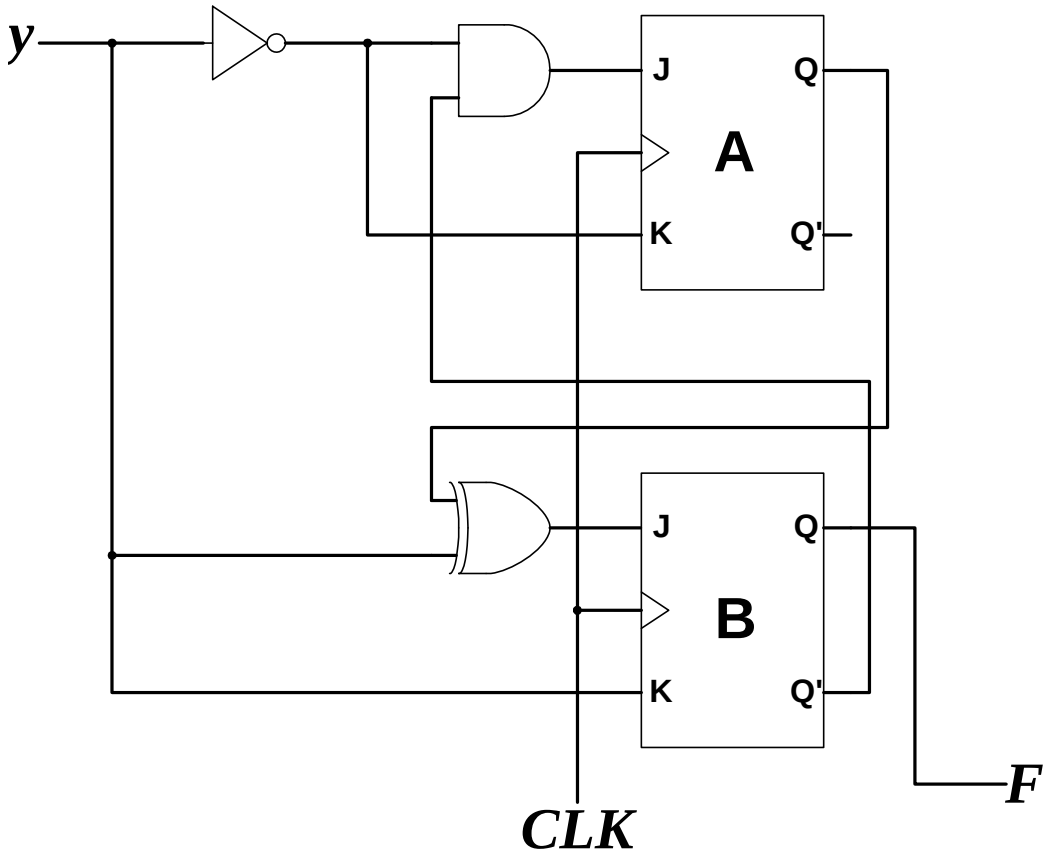
$K_B = y$

F:

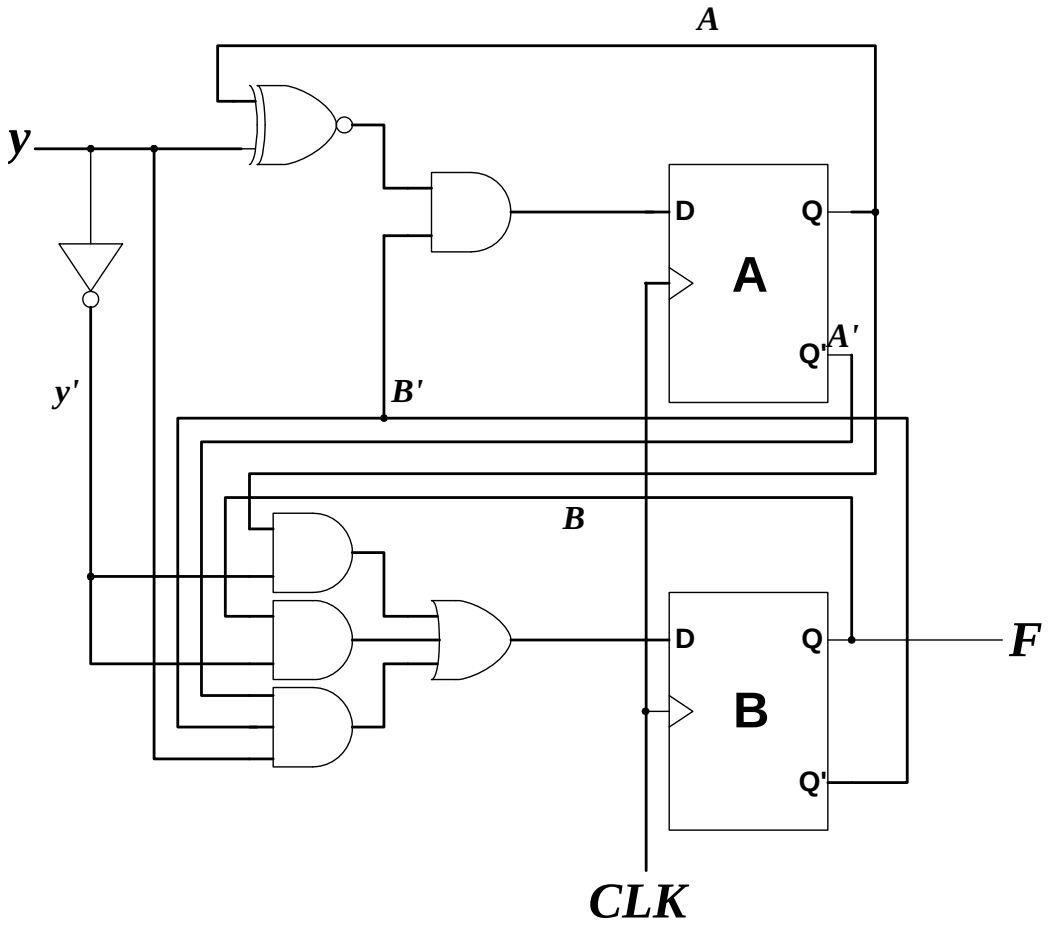
By / ***A***	00	01	11	10
0			1	1
1			X	X

F = B

Videre følger implementering for disse uttrykkene. Til sammenligning er implementeringen med D-FlipFloper som vist på forelesningen. Man kan lett se at logikken (dvs. antall porter) er betydelig mindre komplisert med JK-FlipFloper enn med D-FlipFloper. Dette fordi JK-FlipFlopen innehar mer funksjonalitet og mange don't care-betingelser i eksitasjonstabellen.



Over er en implementering med JK-flip-flop, legg merke til enkelheten til logikken (dvs. antall porter) kontra implementeringen under. Kretsene gjør den samme funksjonen.



Implementering med D-flip-flop som vist på forelesningen 29. januar (inngangen på flipflop A er forøvrig forenklet i forhold til det som ble vist på forelesningen)