1 A)

$$Y = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,B\,\overline{C}\,D + A\,C$$

Faktoriserer:

$$\overline{(\overline{B} + B)}A\,\overline{C}\,\overline{D} + \overline{A}\;\overline{C}\,\overline{D} + (\overline{B} + B)\,\overline{A}\,\overline{C}\,D + \overline{A}\,\overline{C}\,D + A\,C$$

Bruker regel 6 på paranteser

$$\overline{A}\,\overline{C}\,\overline{D} + \overline{A}\;\overline{C}\,\overline{D} + \overline{A}\,\overline{C}\,D + \overline{A}\,\overline{C}\,D + A\,C$$

Faktoriserer

$$\overline{A}\,\overline{C}\,(\overline{D} + D + \overline{D} + D) + \;A\,C$$

Bruker regel 6 på paranteser

$$= \overline{A}\,\overline{C} + AC$$

Utrykket på standard SOP-form ser slik ut:

$$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,B\,\overline{C}\,D + A\,\overline{B}\,C\,D + A\,\overline{B}\,C\,D + A\,\overline{B}\,C\,\overline{D} + A\,B\,C\,\overline{D} + A\,B\,C\,D$$

Setter opp sannhetstabellen for utrykket

A	B	C	D	Y
0	0	0	0	1
0	1	0	0	1
0	0	0	1	1
0	1	0	1	1
1	0	1	1	1
1	0	1	0	1
1	1	1	0	1
1	1	1	1	1

Utrykket er falskt i alle andre tilfeller (0)

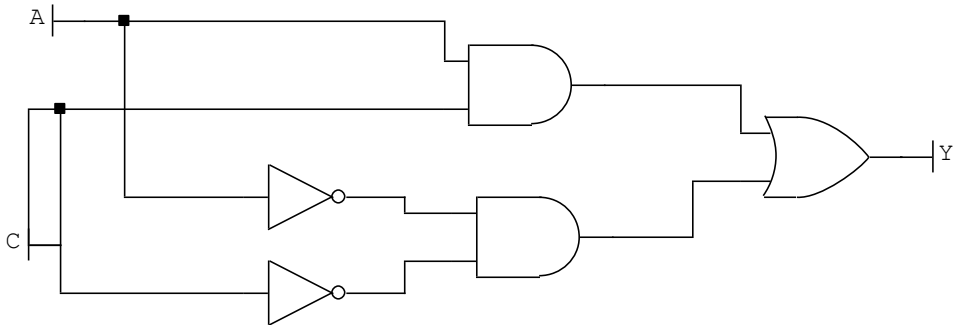
CD \ AB	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	0	0	1	1
10	0	0	1	1

Svart gruppe: $\overline{A}\,\overline{C}$

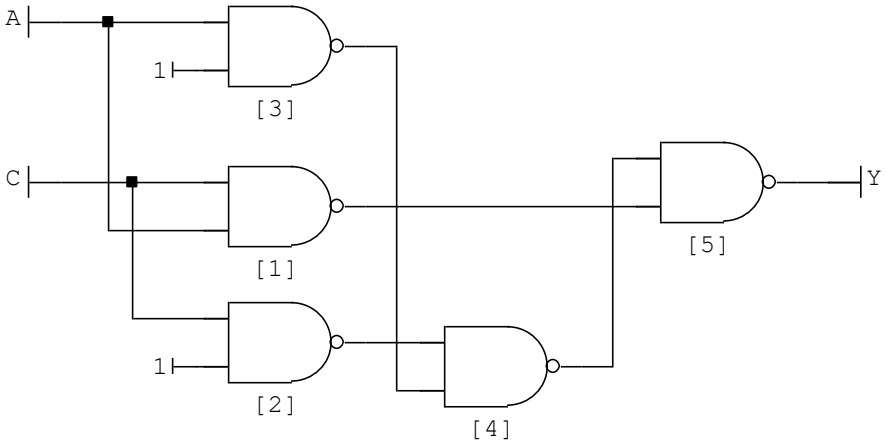
Rød gruppe: $A\,C$

$$Y = \overline{A}\,\overline{C} + AC$$

Kretstegning:



Kretstegning med bare NAND-porter:



B)

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

Ser i hvilke tilfeller utgangen (X) er sann, og setter opp utrykket ut ifra dette

$$\boldsymbol{X = \bar{A}\,\bar{B}\,C + \bar{A}\,B\,C + A\,\bar{B}\,\bar{C} + A\,\bar{B}\,C + A\,B\,C}$$

Setter dette inn i Karnaugh-diagrammet

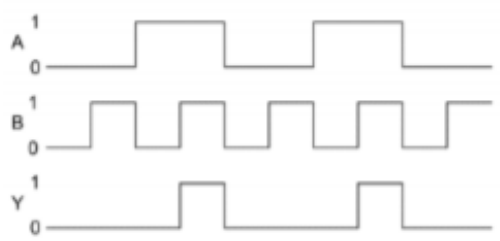
C \ AB	00	01	11	10
0	0	0	0	1
1	1	1	1	1

Svart gruppe: C

Rød gruppe: $A\bar{B}$

$$\boldsymbol{X = A\bar{B} + C}$$

C)

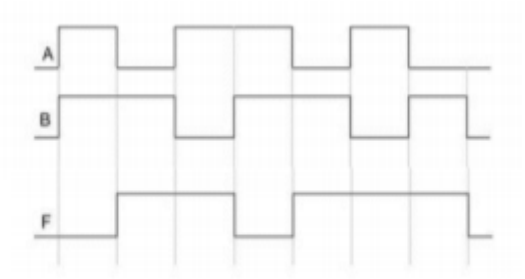


Figur 2.1

Leser av tids-diagrammet og setter opp en sannhetstabell

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Ser av sannhetstabellen at det er **logisk og** som utføres



Figur 2.2

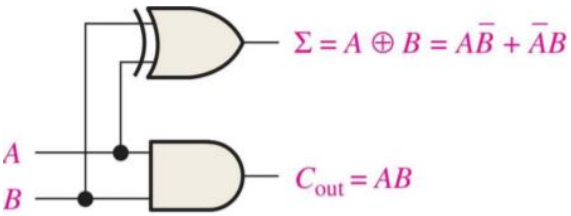
Leser av tids-diagrammet og setter opp en sannhetstabell

A	***B***	***F***
0	0	0
0	1	1
1	0	1
1	1	0

Ser av sannhetstabellen at det er **XOR-logikken** som utføres.

D)

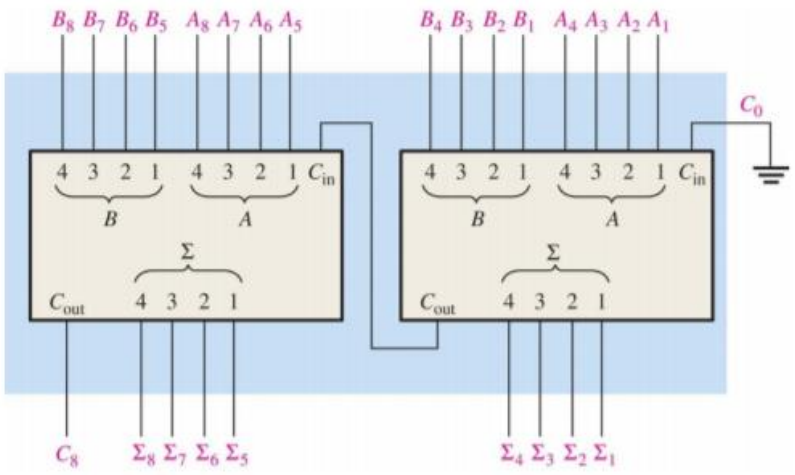
En full adderer består essensielt av to halv-adderere. Så jeg vil først se på virkemåten til en halv-adderer. Kretsskjema for en halv-adderer ser slik ut:



Den har to innganger, hvor disse går til en XOR-port. Denne porten tar for seg summeringen, som vi kan se stemmer av sannhetstabellen. Inngangene går også til en AND-port som tar for seg «carry»-bitet. «Carry» bitet oppstår når vi prøver å summere 1 og 1. Summen vil da bli 0, og Carry-bitet blir 1.

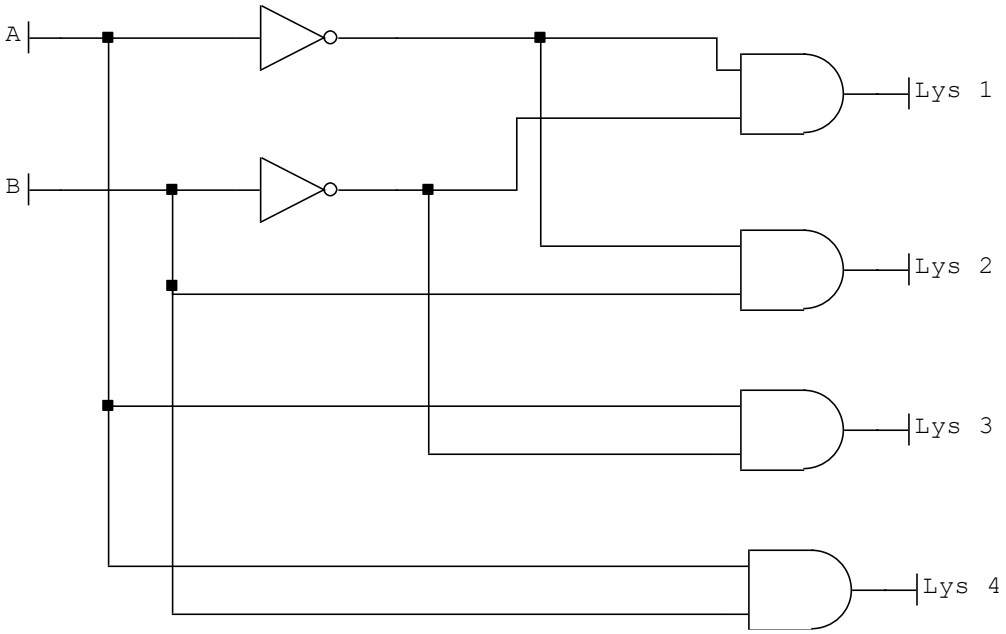
En full-adderer består som nevnt av to halv-adderere. Vi har tre input bits (som summeres) men vi har også et output «carry» bit sammen med den andre outputen (summe). Over er det beskrevet at en XOR tar for seg summeringen av to bits. For i tillegg å kunne legge til input-carry benyttes en andre XOR-port som summerer utgangen fra den første XOR-porten og input-carry bitet. Vi må i tillegg ha logikk som regner ut output-carry bitet. Dette gjøres ved hjelp av to AND-porter og en OR-port. Satt sammen utgjør dette da to halv-adderere.

For å summere to byte (Bit A og bit B), må vi benytte noe som kalles en kaskade-kobling. Denne består av to fire-bit addere. En fire bit-adderer består igjen av fire full-adderere. En oversikt over kaskadekobling ser slik ut:



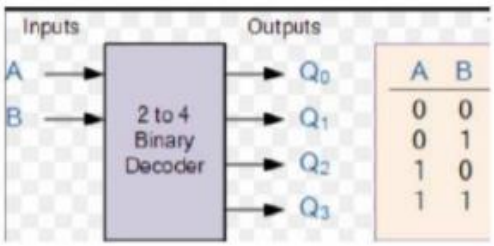
E)

Et kretsskjema av en 2 til 4 linjer dekoder kan se slik ut



Her er alle kombinasjoner av av/på koblet på hvert sitt lys

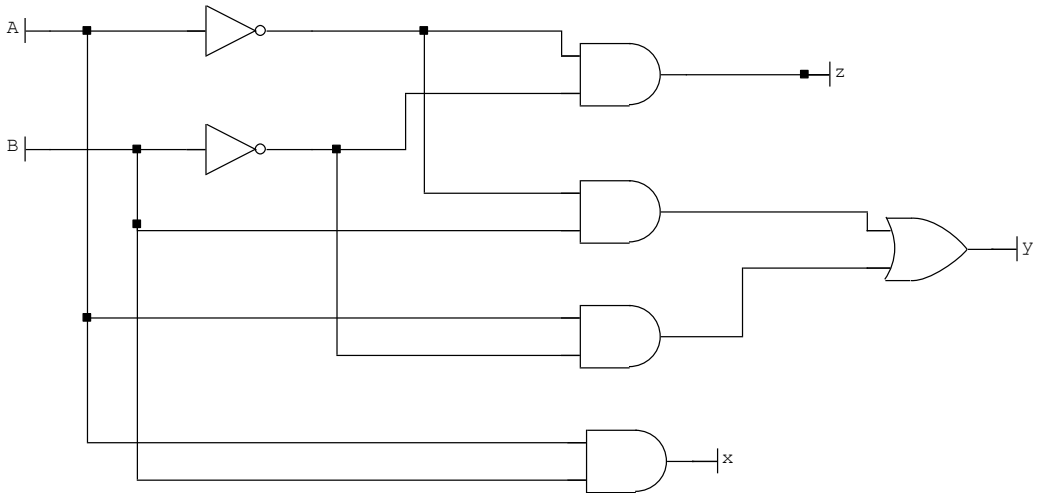
F)



Figur 2.3

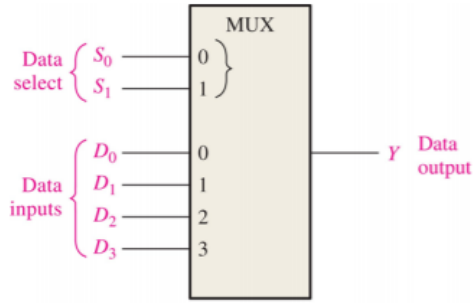
En to til fire-binære dekoder gir en verdi til hver av de fire utgangene basert på hva de to inngangene er. De to inngangene kan være alle kombinasjoner av (av/på, 0/1). Utgangene vil dermed være enten av eller på avhengig av hvilken kombinasjon vi har på inngangene. Ser vi på figuren over vil (0,0) ved inngangene gjøre at Q_0 blir aktivert (1). Videre vil (0,1) på inngangene gjøre at Q_1 blir aktivert. Slik fortsetter det, og (1,1) aktiverer utgang Q_3.

Som vi ser, blir bare en av utgangene aktivert om gangen. Man sier at kombinasjonen på inngangen blir dekodet og en av utgangene produserer er resultat deretter. For å lage en XOR-port ved hjelp av denne kretsen og en OR-port kan vi koble utgangene Q_1 og Q_2 på inngangene til OR-porten. Utgangen til denne OR-porten vil dermed ha samme verdi som en XOR-port avhengig av inngangene A og B. En kretstegning kan se slik ut:



G)

For å løse dette problemet ville jeg benyttet en **Multiplekser-krets.** Disse kalles ofte også for **Data-selectors.** Det fordi vi kan kjøre flere linjer med data inn, og få en enkelt linje med data ut igjen. En multiplekser-krets har et sett med datalinjer inn, og et sett med linjer (Data-select) som bestemmer hvilken av datalinjene som skal rutes til utgangen. En helt enkel multiplekser krets kan se slik ut.

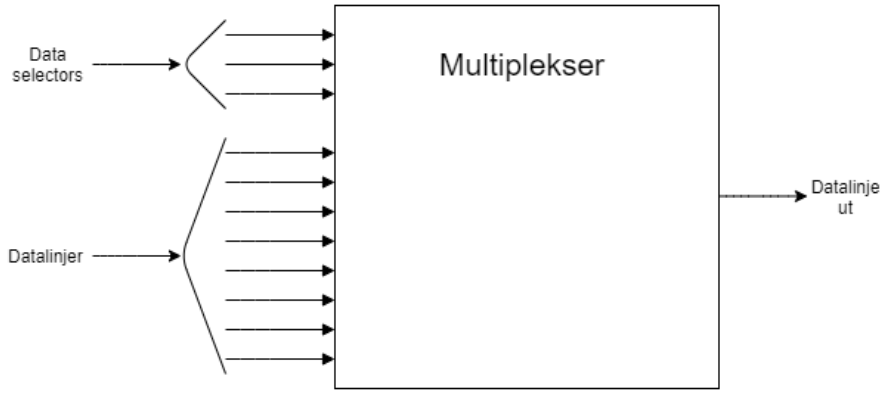


Data-Select Inputs		
S_1	S_0	Input Selected
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Til høyre ser vi sannhetstabbelen for denne enkle multiplekseren som godt illustrerer hvordan select-bitene styrer hvilken ingang som skal rutes til utgangen.

Hvis vi har 8 slike datalinjer som skal velges til en datalinje følger vi samme prinsippet som ovenfor. Forskjellen er at vi her må benytte flere select-bits. (3 bits) for å kunne få nok kombinasjoner til å dekke hver datalinje. Når vi benytter 3 bits får vi 2^3 antall kombinasjoner av select-bitene, noe som tilsvarer en for hver av de åtte datalinjene som kommer inn. For kombinasjonen (0,0,0) vil første datalinje bli rutet til utgangen, for (0,0,1) blir andre datalinje valgt osv.

En skisse av prinsippet kan se slik ut:



H)

Vipper er digitale kretser som kan ha flere tilstander. Her inngår en, to eller ingen stabile tilstander. En SR-vippe (set-reset) kategoriseres ofte som en «vanlig» vippe. En SR-vippe har to stabile tilstander, det er enten 0 eller 1. (Høy eller lav). Følgelig egner SR-vippen seg godt som et dataminne. Den kan «holde» på en 0 eller 1 helt til den blir tilbakestilt eller satt på nytt. Felles med andre typer vipper kan den også brukes i tellere og liknende.

SR-vippen egner seg også godt i oppgaven å eliminere såkalt «contact bounce». Et slikt fenomen oppstår når man skal slå av eller på en mekanisk bryter. Polen på bryteren kan vibrere opp ned flere ganger før den får ordentlig kontakt, og det blir sendt ut stigende/fallende spenningsverdier et lite sekund før bryteren er av. Dette er ofte spenningsverdier som er tilstrekkelig til å kunne leses av for eksempel et digitalt system. Ved å koble et system sammen med en SR-vippe vil disse «vibreringene» ikke utgjøre noen forskjell da vippen settes ved det første signalet den får. Den endrer da følgelig ikke tilstand før det blir tilbaketilt igjen.

En SR-vippe blir satt til 1 når $\bar{S}, \bar{R}$ henholdsvis er (0,1) og til 0 når vi har de omvendte ingangene. **Hvis begge innganger er 0 (S=1 og R=1)** forteller man i praksis at vippen skal bli satt til både 0 og 1

samtidig. Dette blir et problem når ingangene faller tilbake til normaltilstand, og særlig hvis dette skjer samtidig. Da vil det i praksis være umulig å avgjøre hvilken av kondisjonene som skal være gjeldende. Tilstsanden til vippen når både S og R er 1 vil derfor være ugyldig.

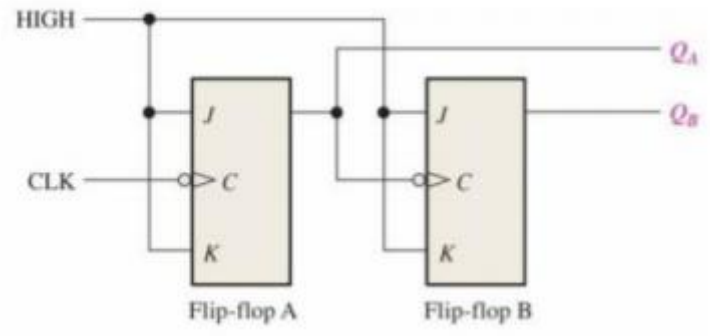
I)

En portstyrt D-vippe har mye til felles med en SR-vippe, men har istedenfor en inngang D. Typisk for en slik vippe er at utgangen følger D-inngangen så lenge vippen er «enabled». Vippen har i tillegg til D -inngangen en «enable» inngang. Denne inngangen må være høy for at det skal kunne skje noen endring på utgangen. Hvis enable-bitet er 0, skjer det altså ingen endring på utgangen. Er enable-bitet 1(Høy) følger utgangen Q inngangen D.

En portsstyrt D-vippe er, som SR-vippen godt egnet til å brukes som minne. Vippen kan særlig brukes for å holde en bit. 0 eller 1. Denne egner seg ekstra godt fordi vi rett og slett kan sette Enable-bitet til lav, og vippens tilstand vil holde seg slik som den var helt til bitet blir 1 (høy igjen). Oppsummert har en portstyrt D-vippe bare en datainngang. Utgangen følger denne så lenge «enable» er 1. (høy).

J)

Vipper, eller Latcher kan brukes til en rekke ting. Noen er nevnt ovenfor flere ganger, og er eksempelvis dataminne. Flere bruksområder er som puls-generator, pulstog med en gitt frekvens, frekvensdeling og som en teller.

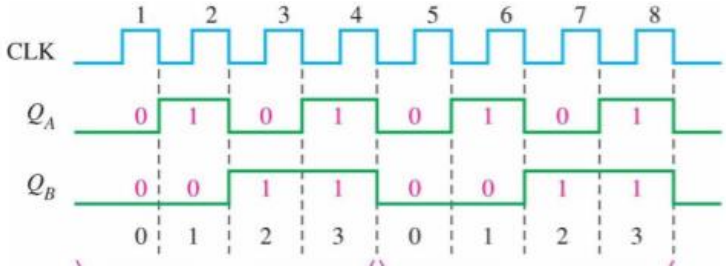


En JK-vippe kan ses på som en oppgradering av SR-vippen. J tilsvarer S(set) og K tilsvarer R(reset). Oppgraderingen ligger hovedsakelig i at denne vippen takler situasjonen hvor både S og R blir en. (som jo er en ulovlig tilstand i en SR-vippe). Når S=R=1 tilstanden inntreffer sier vi at vippen «toggler» det vil si at den bytter tilstand. 0->1 og 1->0. JK-vippen er også en synkron vippe fordi den benytter et klokke-signal for tidsendring. Endring i tilstander kan enten skje ved positiv eller negativ flanke på klokkesignalet. Figuren over trigger på negativ flanke, det kan vi lese av ikke-tegnet på inngangen til klokkesignalet. (CLK).

En slik vippe fungerer også som en frekvensdeler, det vil si at frekvensen til signalet på utgangen er halvparten av klokkesignalet ved inngangen. Det fordi vippen i toggle-tilstand «trigger» på hver flanke. Jo flere vipper vi har etter hvrandre, jo flere ganger vil signalet ved utgangene dele seg. Har vi

eksempelvis 1000Hz inn på CLK, vil signalet ut fra flip-flop A være halvparten så stort, og ha en frekvens på 500Hz. Signalet ut fra flip-flop B vil igjen være halvvert, og tilsvare en frekvens på 250 Hz.

Som nevnt ovenfor kan vippen også brukes som en teller. Figuren ovenfor består av to flip-flopper og vil da kunne representere 4 tilstander. (de to tilstandene på hver lagt sammen). Ved å studere tidsdiagrammet kan vi se for oss hvordan vippene fungerer som en teller



Vi ser at ved første klokkepuls er både Q_a og Q_b0. Dette representerer en 0. Ved negativ flanke skjer det en endring, og Q_a blir 1. Q_b er fortsatt 0, noe som til sammen representerer 1. Ved neste negative flanke blir verdiene motsatt, altså (10) noe som tilsvarer tallet 2. Tilslutt blir både Q_a og Q_b 1, noe som tilsvarer tallet 3. Vippene har da gått igjennom alle kombinasjonene, og telt fra 0 til 3.