11/10-21 Gruppe 1

Time 1:

Forje uke så vi på de ulike portene vi kan bruke, hvordan disse ser ut, litt om hvordan de funker og hva de gjør. Det vi skal gjøre denne uken er å se hvordan vi kan lage kretser med dem, som representerer ulike funksjoner.

Så, over på funksjoner. Det er et lite sprang fra å skjønne at to enere i en AND-port blir til en ener over til at en AND-port kan skrives som F = AB. Egentlig trenger dere ikke å skjønne hvorfor for å bestå dette emnet, heller...

En port har alltid en logisk funksjon, men noen (AND, XOR og XNOR) har matematiske funksjoner og. XOR er "≠", XNOR er "=" og AND kan bli brukt for å overføre mente, siden den gir true når man har to enere.

Den enkleste funksjonen er nok NOT, som bare har at F = A'. Apostrofen betyr invertert, eller "motsatt av". Når du ser denne syntaksen vet du altså at det er en NOT-port inni bildet.

OR representerer F = A+B. Dette er lett å huske siden OR sin sannhetstabell ligner litt på binær addisjon.

Hvis jeg skriver F = A'+B ser vi at A blir invertert før den går inni OR-porten. Hadde jeg skrevet F = (A+B)' blir outputen invertert, altså etter å ha gått gjennom OR-porten. Merk at det er viktig å stadfeste om det er input eller output som skal inverteres, for svarene blir ofte ulike som følge.

Se på forskjellene mellom F = A' B' og F = (AB)'. Denne funksjonen kan lages med en AND-port og en eller to NOT-porter.

F=A'B' (to invertere på input)		
A'	B'	F
1	1	1
0	1	0
1	0	0
0	0	0

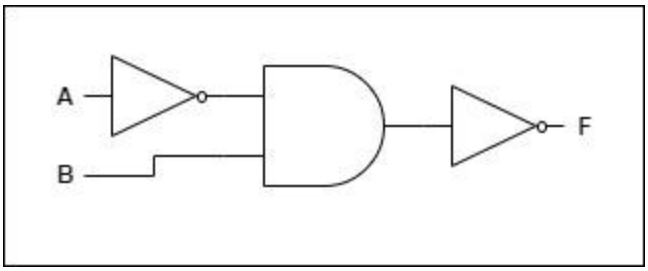
F=(AB)' (en inverter på output)		
A	B	F'
0	0	1
1	0	1
0	1	1
1	1	0

F=(AB)' minner oss om en annen port vi kjenner til. Hvilken da?

Viktig å huske på er at når vi leser en krets er det kun én funksjon som kan representere denne kretsen, men hvis vi blir gitt en funksjon finnes det ofte mange ulike kretser som kan representere denne funksjonen.

Når vi tegner kretser har vi som regel input på venstre side og output på høyre side. Du kan lese begge veier for å tolke en krets. Jeg liker å ta det fra venstre til høyre.

La oss prøve å analysere en krets:

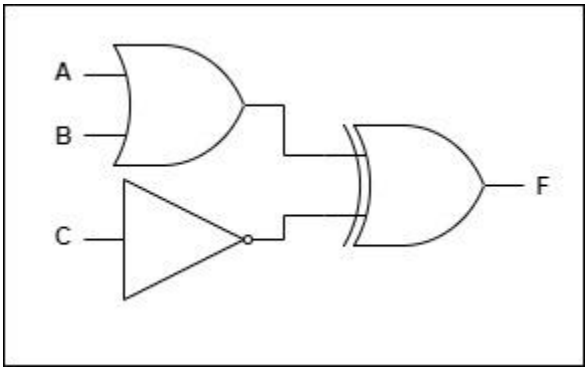


A’	B	F’
0 (1)	0	1
1(0)	0	1
0 (1)	1	0
1 (0)	1	1

Vi har tre porter og to input. A blir invertert, og går inn i en AND-port sammen med B. Deretter blir resultatet invertert. Prøv å lese fra venstre til høyre og tegn opp en sannhetstabell.

Kunne vi tegnet denne kretsen med færre porter?

Her er en annen krets, med tre input også med tre porter.

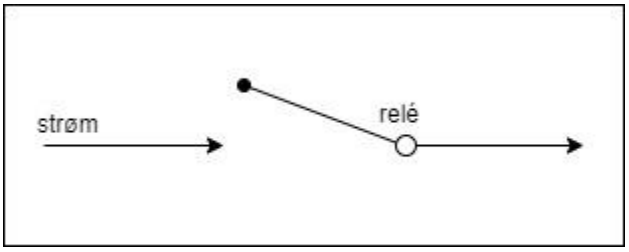


A	B	C’	F
0	0	0 (1)	1
1	0	0 (1)	0
0	1	0 (1)	0
0	0	1 (0)	0
1	1	0 (1)	0
1	0	1 (0)	1
0	1	1 (0)	1
1	1	1 (0)	1

Denne er litt vanskeligere. Ta en port av gangen og tenk over hvordan outputen blir i hvert steg. Husk at OR-porten gir 1 når enten A, B eller A og B er 1, mens XOR-porten bare gir 1 når enten A+B eller C er 1.

Time 2:

Vi snakket litt om transistorer sist gang. At porter er abstraksjoner av transistor-kretser og at det er transistor-kretsene som bestemmer når strømmen i PC-en skrus av eller på (0 eller 1). Før transistorer brukte vi vakumrør, som også var elektroniske, men skjøre og ble brukt opp, litt som lyspærer. Før vakumrør brukte vi reléer (relays), som var mekaniske.



Strømmen kom gjennom ledningen. Om strømmen kom videre var avhengig av om reléen var lukket eller åpen. På tegningen er den åpen, så her blir output null. Mekaniske deler er uberegnelige og blir lett ødelagte eller brukt opp, så disse brukes ikke lenger, men prinsippene er de samme fremdeles. Vi kan se for oss at denne reléen bruker litt tid på å lukkes og åpnes. Det samme gjør transistorer. Transistorer har ingen bevegelige deler, men de bruker allikevel litt tid på å "åpnes" og "lukkes". Veldig, veldig lite tid, men nok til at det gjør en forskjell.

Hvordan klarer datamaskinene våre å gjøre ting i riktig rekkefølge når det er tusener av transistorer som kanskje utfører oppgavene sine raskere eller saktere enn forventet?

Jo, vi bruker noe som heter klokkehastighet. Vi gir alle transistorer nok tid til å utføre oppgavene sine før resultatene blir ført videre. Litt som et samlebånd. Vi gir alle like lang tid til å gjøre sin oppgave, slik at det ikke hoper seg opp et sted.

Klokkesignalet fremstilles som en sånn her krenelering-aktig sak (borg-mur).



En klokkeperiode beregnes som tiden mellom en stigende flanke, til neste stigende flanke. Intervallene er konstante.

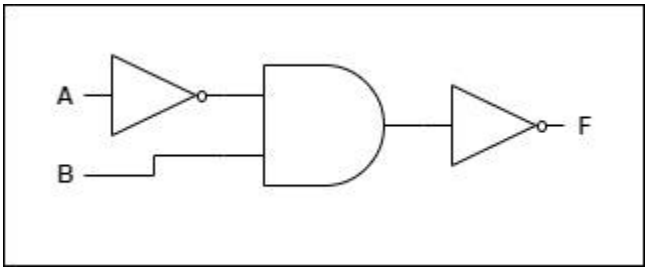
Frekvensen til maskinen kan regnes ut som f = 1/klokkeperiode.

f = 1/1ms = 1/0.001s = 1kHz

Du kan endre frekvensen til datamaskinen din slik at den kan kjøre raskere (overclocking). Jeg anbefaler ikke dette, siden det kan skade maskinen din ved at den kjører seg varm. De som lagde maskinen din kjenner nok begrensningene dens og har satt klokkehastigheten deretter.

Det er ofte flere klokker i en maskin, som kjører i forskjellige tempo. Da er det viktig at disse er synkrone. Altså at de har en fellesnevner, slik at de ikke kjører i utakt for alltid eller over lang tid. Hvis det tar lang tid å synkronisere klokkene er det vel kanskje ikke så mye vits med flere klokker.

Det er som regel forsinkelser gjennom porter. For å regne ut forsinkelse må vi se på den lengste veien gjennom kresten. Vi må altså finne den "logiske dybden" til kretsen.



Denne kretsen fra i sted har en logisk dybde på 3, siden A må gå gjennom tre porter for å bli F. B må bare gå gjennom to porter, men vi kan ikke regne med at hele kretsen skal gå like raskt som den korteste veien. Hvis hver port har en forsinkelse på f.eks 5ns får hele kretsen en delay på 15ns.