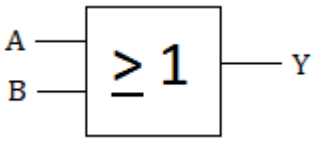
1.3 - Logiska grindar, kombinatorik och boolesk algebra

1.3.1 - Introduktion

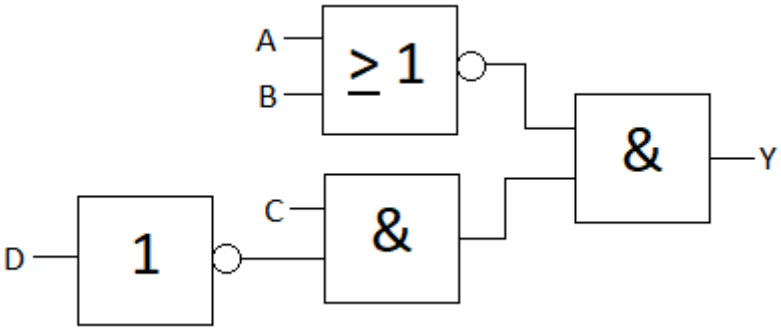
- Logiska grindar är de mest grundläggande byggstenarna i digitaltekniken, som används för att bygga upp alla digitala system, allt från små digitala kretsar till mikroprocessorer.
- Logiska grindar är digitala kretsar, där en utgång Y är logiska funktioner av en eller flera ingångar, såsom A, B och C.
- De logiska funktionerna kan beräknas via så kallad boolesk algebra, vilket är algebra för binära tal.



OR-grind ritad med rektangulär symbolik. A och B utgör insignaler och Y utgör utsignal.

- Grindarna kan ritas ut via två symbolsystem, som kallas militär respektive rektangulär symbolik. I Sverige används vanligtvis rektangulär symbolik, se figuren till höger.
- För att demonstrera hur grindarnas olika funktioner nedan så betecknas deras utgångar med Y, A och B är deras ingångar om två insignaler finns och X den enda ingången på NOT-grinden.

OR-grind ritad med amerikansk symbolik. A och B utgör insignaler och Y utgör utsignal.



Logiskt grindnät ritat med rektangulär symbolik. Detta grindnät består av ett flertal olika grindar, som heter NOT- AND- och NOR-grinden.

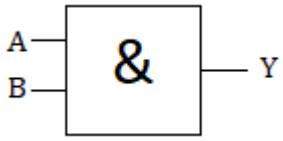
1.3.2 – Genomgång av de olika logiska grindarna

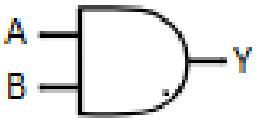
- I detta avsnitt behandlas de olika logiska grindarna, AND-, OR-, NOT-, XOR-, NAND-, NOR- och XNOR-grinden samt buffern. Samtliga kretsars symboler, sanningstabeller och logiska funktioner ritas upp, följt av beskrivningar av respektive krets.

- Vi kommer fokusera på de rektangulära symbolerna, som är de vanligaste symbolerna i Europa, men man bör även vara medveten om de militära symbolerna, som främst används i Nordamerika. Många datablad innehar nämligen dessa symboler, inte de rektangulära.

1. AND-grinden

Symboler och sanningstabell:





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

Logisk funktion:

$$Y = A * B,$$

där A samt B är insignaler och Y är utsignal.

- Andra vanliga notationer för AND-grindens logiska funktion är

$$Y = A \,\&\, B$$

samt

$$Y = A \wedge B$$

Beskrivning:

- Båda insignaler A och B måste vara lika med 1 för att utsignalen Y skall bli lika med 1:

$$A = B = 1 \rightarrow Y = 1,$$

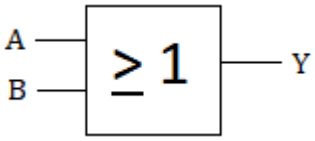
annars om någon av A och B är noll, så blir Y lika med 0:

$$A \vee B = 0 \rightarrow Y = 0,$$

där $\vee$ betyder eller.

2. OR-grinden

Symboler och sanningstabell:



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

Logisk funktion:

$$Y = A + B,$$

där A samt B är insignaler och Y är utsignal.

- Andra vanliga notationer för OR-grindens logiska funktion är

$$Y = A|\, B$$

samt

$$Y = A \vee B$$

Beskrivning:

- A eller B måste vara lika med 1 för att utsignalen Y skall bli lika med 1:

$$A \vee B \rightarrow Y = 1,$$

där $\vee$ betyder eller.

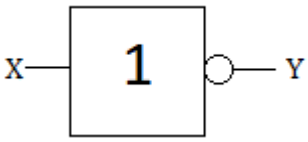
- Det räcker alltså med att endast en av A och B är 1 för att utsignalen Y skall bli 1.

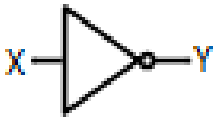
- Däremot om både A och B är 0, så blir utsignalen Y lika med 0:

$$A = B = 0 \rightarrow Y = 0$$

3. NOT-grinden

Symboler och sanningstabell:





X	Y
0	1
1	0

Logisk funktion:

$$Y = X',$$

där X är insignal och Y är utsignal.

- En annan vanlig notation för NOT-grindens logiska funktion är

$$Y = \sim X,$$

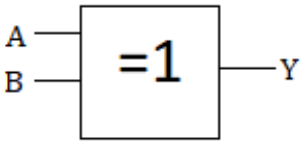
där ~ betyder negation (invertering).

Beskrivning:

- Insignalens värde X inverteras, vilket medför att utsignalen Y är motsatt värde. Som exempel, en insignal X = 1 medför en utsignal Y = 0. På samma sätt medför en insignal X = 0 att utsignalen Y = 1.

4. XOR-grinden

Symboler och sanningstabell:



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

Logisk funktion:

$$Y = A \oplus B = AB' + A'B,$$

där A samt B är insignaler och Y är utsignal.

- En annan vanlig notation för XOR-grindens logiska funktion är

$$Y = A \text{ ^ } B,$$

där ^ betyder XOR (Exklusive OR).

Beskrivning:

- Insignaler A och B måste inneha olika värden för att utsignalen Y skall bli 1:

$$A \neq B \rightarrow Y = 1$$

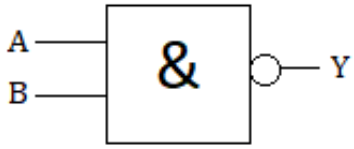
- Antingen måste A eller B vara 1 medan den andra är 0 för att utsignalen Y skall bli 1.

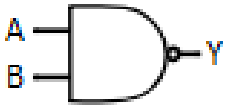
- Däremot om insignaler A och B innebär samma värde, så blir utsignalen Y lika med 0:

$$A = B \rightarrow Y = 0$$

5. NAND-grinden:

Symboler och sanningstabell:





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

Logisk funktion:

$$Y = (A * B)',$$

där A samt B är insignaler och Y är utsignal.

- En annan vanlig notation för NAND-grindens logiska funktion är

$$Y = \sim(A * B),$$

där ~ betyder negation (invertering).

Beskrivning:

- Inverterad AND, vilket innebär samma funktion som en NAND-grind, fast utsignalen Y blir inverterad. Därmed gäller att utsignalen Y blir lika med 0 om båda insignaler A och B är 1:

$$A = B = 1 \rightarrow Y = 0,$$

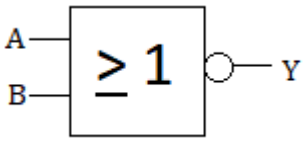
annars om någon av A och B är noll, så blir Y lika med 1:

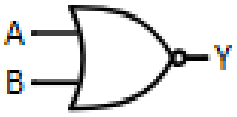
$$A \vee B = 0 \rightarrow Y = 1,$$

där $\vee$ betyder eller.

6. NOR-grinden

Symboler och sanningstabell:





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

Logisk funktion:

$$Y = (A + B)',$$

där A samt B är insignaler och Y är utsignal.

- En annan vanlig notation för NOR-grindens logiska funktion är

$$Y = \sim(A * B),$$

där ~ betyder negation (invertering).

Beskrivning:

- Inverterad OR, vilket innebär samma funktion som en OR-grind, fast utsignalen Y blir inverterad. Därmed gäller att utsignalen Y blir lika med 0 ifall A eller B är 1:

$$A \vee B \rightarrow Y = 0,$$

där $\vee$ betyder eller.

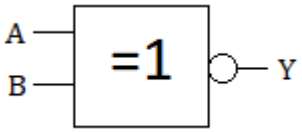
- Däremot om både A och B är 0, så blir utsignalen Y lika med 1:

$$A = B = 0 \rightarrow Y = 1$$

7. XNOR-grinden

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

Symboler och sanningstabell:



Logisk funktion:

$$Y = (A \oplus B)' = (AB' + A'B)',$$

där A samt B är insignaler och Y är utsignal.

- En annan vanlig notation för XNOR-grindens logiska funktion är

$$Y = \sim(A \wedge B),$$

där ~ betyder negation (invertering) och ^ betyder XOR.

Beskrivning:

- Inverterad XOR, vilket innebär samma funktion som en XOR-grind, fast utsignalen Y blir inverterad. Därmed gäller att utsignalen Y blir lika med 1 om insignalerna A och B innehar samma värde:

$$A = B \rightarrow Y = 1$$

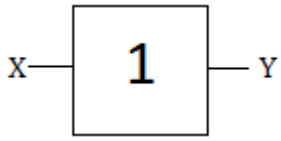
- Däremot om insignalerna A och B är olika, exempelvis att A är 1 och B är 0, så blir utsignalen Y lika med noll:

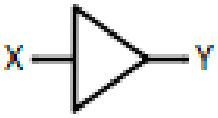
$$A \neq B \rightarrow Y = 0$$

- A och B måste alltså inneha samma värde för att utsignalen Y skall bli 1.

8. Buffern

Symboler och sanningstabell:





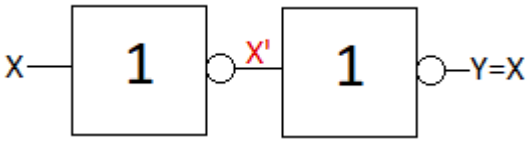
X	Y
0	0
1	1

Logisk funktion:

$$Y = X,$$

där X är insignal och Y är utsignal.

Beskrivning:



Buffern konstruerad med två kaskadkopplade NOT-grindar.

- Buffern utgör egentligen inte en egen grind, utan består två kaskadkopplade NOT-grindar, se figuren till höger.
- Därmed inverteras bufferns insignal X till X' av den första NOT-grinden. Detta värde inverteras sedan tillbaka till X, vilket medför att bufferns utsignal Y logiskt sett är lika med X.
- Vid en första anblick kan därför buffern antas vara onödig, då den inte verkar inneha någon funktion. Dock är buffern mycket viktig inom digitalteknik för att eliminera brus som kan uppstå på signalerna. Detta åstadkommes genom att buffern via de två NOT-grindarna återställer insignalen X till sitt originalvärde, om detta har påverkats av brus.
- Signaler som har blivit påverkade av brus kan inneha ett värde som inte är exakt 0 eller 1. Dock inverterar NOT-grindar inte bara signaler som är exakt 0 eller 1. Det finns ett tröskelvärde, som vanligtvis ligger mellan 0 och 1, beroende på vilken tröskelspänning U_T som NOT-grindarnas transistorer har.
- Som regel kan vi dock anta att detta tröskelvärde ligger runt 0,5. Därmed kan alla signaler som innehar ett värde mellan 0 upp till 0,5 antas inverteras till 1, samtidigt som signaler från 0,5 upp till 1 kan antas inverteras till 0. Signaler som är exakt 0,5 är dock svåra att förutsäga och kan bli vilket som.
- Vi kan därmed anta att alla insignaler X som överstiger 0,5 blir inverterade till 0 av den första NOT-grinden ovan, vilket medför att den första NOT-grindens utsignal X' är lika med 0:

$$X > 0{,}5 \rightarrow X' = 0$$

- Samtidigt kan vi anta att alla insignaler X som understiger 0,5 blir inverterade till 1 av den första NOT-grinden, vilket medför att den första NOT-grindens utsignal X' då blir 1:

$$X < 0{,}5 \rightarrow X' = 1$$

- Som synes i figuren ovan, så utgörs den andra NOT-grindens insignal av den första NOT-grindens utsignal X'. Denna signal kommer sedan inverteras tillbaka till X, vilket medför att den andra NOT-grindens utsignal Y, som också är bufferns utsignal, är lika med X, då

$$Y = (X')' = X$$

- Som exempel, anta en signal vars originalvärde är 1, på grund av brus har minskat till 0,8. Antag att denna signal utgör insignalen X på buffern ovan:

$$X = 0{,}8$$

- Eftersom 0,8 överstiger 0,5, så kan den första NOT-grinden antas invertera detta värde till 0, vilket medför att dess utsignal X' är lika med 0:

$$X' = 0$$

- Därefter inverterar den andra NOT-grinden detta värde till 1, vilket medför att bufferns utsignal Y är lika med 1:

$$Y = 1$$

- Vi ser därmed att en insignal X som har blivit påverkad av brus, vilket medfört att dess värde har minskat från 1 till 0,8, blir återställd till 1 av buffern, då bufferns utsignal Y blir lika med 1:

$$X = 0{,}8 \rightarrow Y = 1$$

- Efter att ha passerat buffern, så har alltså signalen blivit återställs till dess originalvärde 1, vilket innebär att bruset har eliminerats.

- Därmed återställer buffern signaler som har blivit påverkade av brus till dess originalvärde.

- Som nämndes tidigare, så finns en tröskel runt 0,5, där värden under 0,5 kan antas inverteras till 1. Antag att en annan signal, vars ursprungsvärde är 1, har blivit mycket påverkat av brus, då den inte har passerat någon buffer, så att dess värde har minskat till 0,4. Då kommer bufferns insignal X alltså bli 0,4:

$$X = 0{,}4$$

- Eftersom insignalen X understiger 0,5, så kan bufferns första NOT-grind i detta fall antas omvandla insignalen X till X’ = 1:

$$X' = 1$$

- Därefter kommer bufferns andra NOT-grind invertera detta värde till 0, så att bufferns utsignal Y hamnar på 0:

$$Y = 0,$$

  vilket medför en signalförlust, då signaler från början var 1.

- Därmed måste buffrar placeras relativt tätt i digitala kretsar, exempelvis efter varje steg i en transmission, för att minska brus utan signalförluster.

1.3.3 - Regler för boolesk algebra

ABC = (AB)C = A(BC)

AB = BA

A + B = B + A

AA = A

A + A = A

AA’ = 0

A + A’ = 1

A(B+C) = AB + AC

A + AB = A(1+B) = A * 1 = A

A + B + C = (A + B) + C = A + (B + C) = B + (A + C)

1’ = 0

0’ = 1

(A’)’ = A

A + A’B = A + B

Om A är noll så blir A’ lika med 1 och resultatet blir B. Om A är 1 så blir A’ lika med 0 och resultatet blir A.

(A + B)’ = A’ * B’

(AB)’ = A’ + B’

- De två sista lagarna kallas De Morgans teorem och är väldigt viktiga för att realisera CMOS-grindar.

1.3.4 - Konstruktion av logiska grindnät

1. Vi skall rita grindnätet för den logiska funktionen

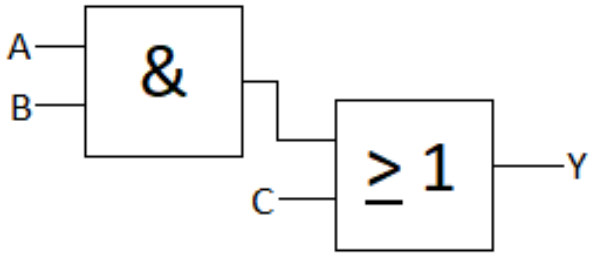
$$Y = AB + C$$

- Precis som i algebra inom matematiken så prioriteras multiplikation för addition. Därmed så fokuserar vi först på AB, som betyder A * B. Därefter kan vi fokusera på AB + C.

- AB är en AND-grind, där A och B är insignalerna, vilket man lätt ser eftersom det är ett gångertecken mellan A och B.

- AB + C är en OR-grind. Denna grind har två insignaler, utsignalen ur den ovannämnda AND-grinden samt C. Detta ser man lätt, eftersom det är ett plustecken mellan AB och C. Utsignalen är denna grind är utsignalen ur hela grindnätet.

- Vi kan därmed enkelt rita ut grindnätet för den logiska funktionen Y = AB + C:

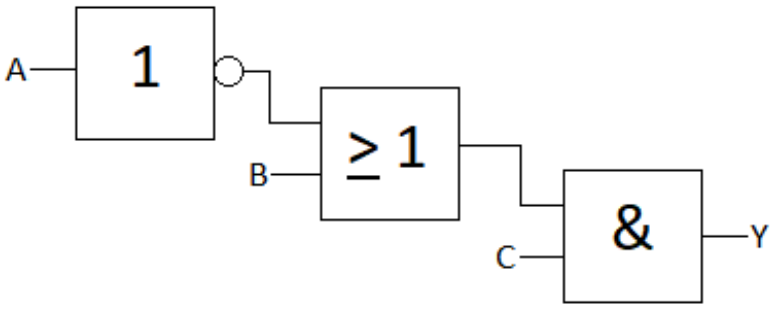


- Om vi istället hade använt amerikanska grindsymboler så hade grindnätet istället sett ut såhär:

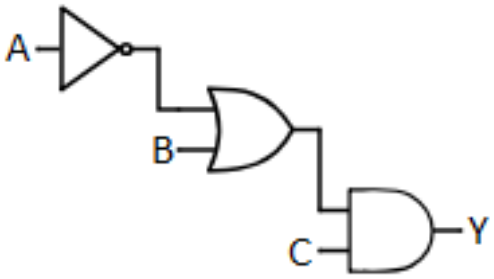
2. Vi skall rita grindnätet för den logiska funktionen

$$Y = (A' + B) * C$$

- Precis som i algebra inom matematiken så prioriteras parenteser framför allt annat. multiplikation för addition. Därmed så fokuserar vi först på A’ + B. Därefter kan vi fokusera på (A’ + B) * C.
- A’ medför att vi först måste invertera signalen A med en NOT-grind.
- A’ + B är en OR-grind, där A’ och B är insignalerna, vilket man lätt ser eftersom det är ett plustecken mellan A’ och B.
- (A’ + B) * C är en AND-grind. Denna grind har två insignaler, utsignalen ur den ovannämnda OR-grinden samt C. Detta ser man lätt, eftersom det är ett gångertecken mellan (A’ + B) och C. Utsignalen är denna grind är utsignalen ur hela grindnätet.
- Vi kan därmed enkelt rita ut grindnätet för den logiska funktionen Y = (A’ + B) * C:



- Om vi istället hade använt amerikanska grindsymboler så hade grindnätet istället sett ut såhär:



3. Vi skall rita grindnätet för den logiska funktionen

$$Y = (A'B + AB' + C')'$$

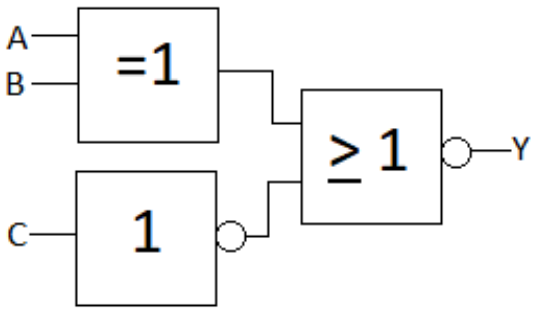
- Precis som i algebra inom matematiken så prioriteras parenteser framför allt annat. multiplikation för addition. Därmed så fokuserar vi först på A'B + AB'. Därefter kan vi fokusera på (A'B + AB' + C')'.

- A'B + AB' är en XOR-grind, där A och B är insignalerna. Detta kan man se eftersom A'B + AB' följer mönstret för en XOR-funktion. Eftersom det är en XOR-funktion så behöver vi inte invertera någon av insignalerna A eller B.

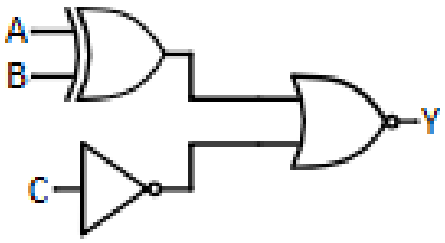
- C' medför att vi måste invertera insignalen C med en NOT-grind.

- (A'B + AB' + C')' är en NOR-grind. Denna grind har två insignaler, utsignalen ur den ovannämnda XOR-grinden samt C'. Detta ser man lätt, eftersom det är ett plustecken mellan A'B + AB' och C'.. Men eftersom det är en apostrof runt denna funktion (A'B + AB' + C')', så är DET en NOR-grind, inte en OR-grind. Utsignalen är denna grind är utsignalen ur hela grindnätet.

- Vi kan därmed enkelt rita ut grindnätet för den logiska funktionen (A'B + AB' + C')':



- Om vi istället hade använt amerikanska grindsymboler så hade grindnätet istället sett ut såhär:



4. Vi skall rita ut grindnätet för följande logiska funktion:

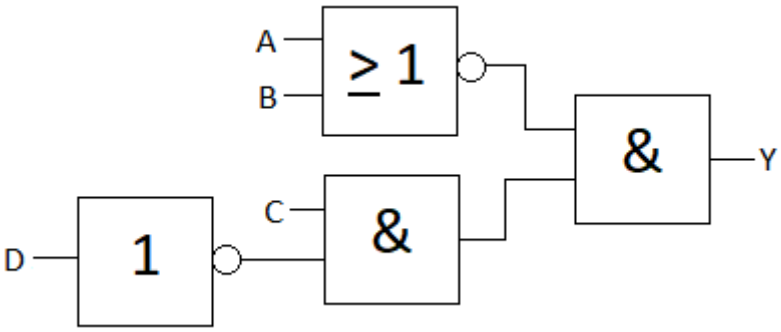
$$Y = (A + B)' * CD'$$

- (A + B)’ är en NOR-grind, där A och B är insignalerna, vilket man lätt ser eftersom det är ett plustecken mellan A och B, samt att utsignalen skall inverteras, vilket man ser på apostrofen.

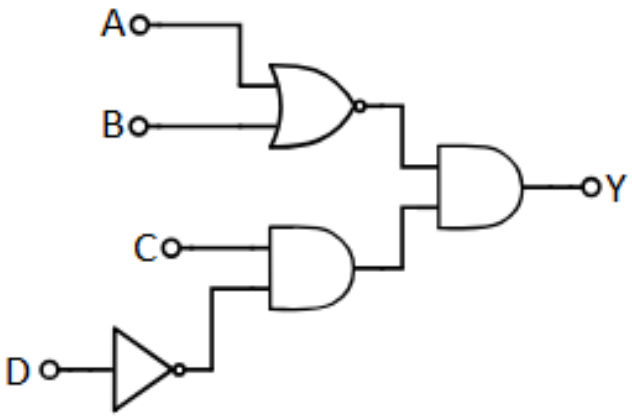
- CD’ är en AND-grind med insignalerna C och D’, eftersom CD’ = C * D’. För att få D’ så måste vi invertera signalen D med en NOT-grind innan den når AND-grinden.

- De två tidigare nämnda delarna skall användas som insignaler till en AND-grind, vilket man lätt kan se då det är ett gångertecken (asterisk) mellan dem.

- Därefter målar vi ut grindnätet för den logiska funktionen Y = (A + B)’ * CD’:



- Om vi istället hade använt amerikanska grindsymboler så hade grindnätet istället sett ut såhär:



5. Vi skall förenkla den logiska funktionen nedan med de Morgans teorem och rita ut grindnätet:

$$Y = (AB)' + (A' + B') * C'D'$$

- **Tips:** Minimera antal grindar genom att minimera antalet NOT-grindar.

- Vi har ett flertal A och B i funktionen, vilket medför att vi bör se om vi kan förenkla funktionen och därmed minska antalet grindar. Det första vi noterar är att vi kan förenkla funktionen ovan med de Morgans teorem, (AB)' = A' + B':

$$A' + B' = (AB)'$$

$$C'D' = (C + D)'$$

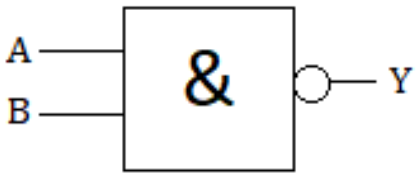
- Genom dessa två omvandlingar kan vi reducera antalet NOT-grindar från fyra till noll, då de enskilda signalerna inte behöver inverteras. Vi börjar med att omvandla A' + B' till (AB)'.

$$Y = (AB)' + (A' + B') * C'D' = (AB)' + (AB)' * C'D'$$

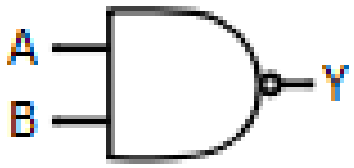
- Som synes så består funktionens båda delar av (AB)'. Vi bryter därför ut (AB)', så vi kan minimera funktionen:

$$Y = (AB)'(1 + C'D') = (AB)' * 1 = (AB)'$$

- Därefter återstår enbart en NAND-grind, med A och B som insignaler. Funktionens grindnät kan därmed ritas enligt nedan:



- Om vi istället hade använt amerikanska grindsymboler så hade grindnätet istället kunna ritas ut enligt nedan:



1.3.5 - Schmitt-triggers och hysteres

- En del logiska grindar har s.k. Schmitt-triggeringångar. Schmitt-triggers används för att återställa förvrängda utsignaler till sin ursprungliga form, dvs. rektangelform, HIGH/LOW. När en digital signal går från hög till låg så sker inte alltid detta tillräckligt fort, p.g.a. fördröjningar orsakade av omkringliggande resistorer och kondensatorer,

- Den funktion som Schmitt-triggers utför kallas hysteres och innebär alltså att en förvrängd insignal som matas in i en logisk krets kommer ha fått tillbaka sin ursprungliga form på utgången.

1.3.6 - Kombinatoriska kretsar och sekvenskretsar

- Det kretsar vi har sett hittills är exempel på olika kombinatoriska kretsar. Det finns också s.k. sekvenskretsar. Kombinatoriska kretsar har ingen minnesfunktion, medan sekvenskretsar har det.

- Utsignalen ur kombinatoriska kretsar är enbart beroende utav insignalerna, exempelvis AND-, OR- och NOT-grindar.

- Sekvenskretsar har minnesfunktion för att spara vilket tillstånd som kretsen hade alldeles innan det nuvarande tillståndet, dvs. vilka in- och utsignaler kretsen hade innan det nuvarande läget, där minst en av in- och utsignalerna har ändrats. Utsignalen ur sekvenskretsar är alltså beroende utav insignalerna precis som kombinatoriska kretsar, men sekvenskretsar är också beroende utav kretsens tidigare tillstånd.

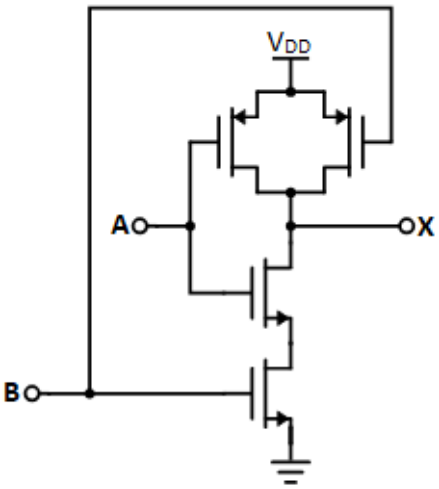
Vi kommer inte gå igenom sekvenskretsar djupare i denna kurs. För den som är intresserad så rekommenderas boken Digital Fundamentals av Thomas L. Floyd.

1.3.7 - NAND-logik

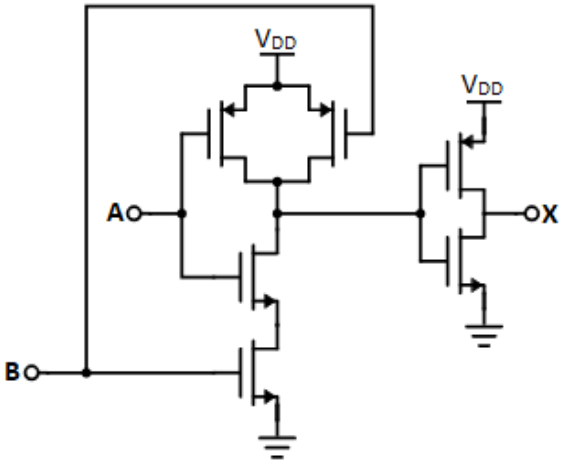
- Alla de logiska grindar vi sett tidigare kan byggas upp med enbart NAND-grindar eller NOR-grindar, vilket kallas NAND-logik respektive NOR-logik.

- I praktiken så används NAND-logik, främst på grund av att man då endast behöver använda en typ av grind, vilket är både enklare och billigare.

- Transistorer är i sin naturliga form en typ av inverterare, vilket medför att NAND-grinden (samt NOR-grinden) är snabbare än övriga logiska grindar. Detta beror på att övriga logiska grindar behöver extra tid för att utsignalen inte skall bli inverterad. Som exempel, en AND-grind behöver en extra inverterare på utgången, vilket medför att AND-grinden är långsammare än NAND-grinden, se figurerna nedan.



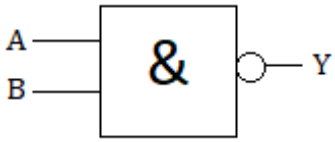
NAND-grinden är snabbare än exempelvis en AND-grind, eftersom det inte behövs en extra inverterare på utgången.



AND-grinden innehar en extra inverterare, vilket medför att denna grind är långsammare än NAND-grinden.

NAND

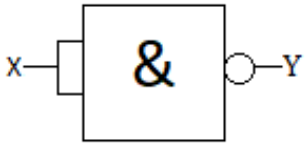
- Byggstenen för alla andra grindar.



$$Y = (A * B)'$$

NOT

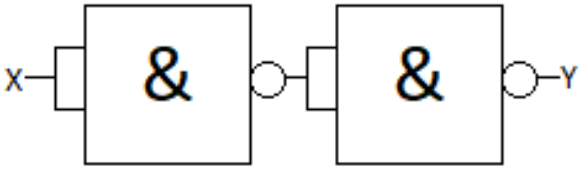
- Vi sammankopplar insignalen på de båda ingångarna och utnyttjar det faktum att X * X är lika med X i boolesk algebra.
- Samtliga NOT-grindar nedan är konstruerade enligt figuren nedan, dvs. med NAND-logik.



$$Y = (X * X)' = X'$$

Buffer

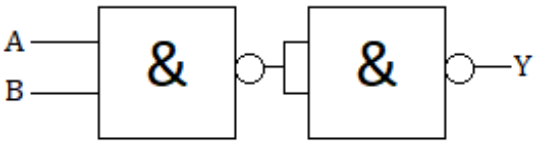
- Konstrueras enkelt genom att sammankoppla två NOT-grindar (i NAND-logik):



$$Y = (X')' = X$$

AND

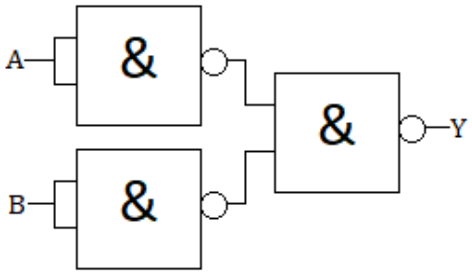
- Består av en NAND-grind, följd av en inverterare (NOT-grind).



$$Y = [(A * B)']' = A * B$$

OR

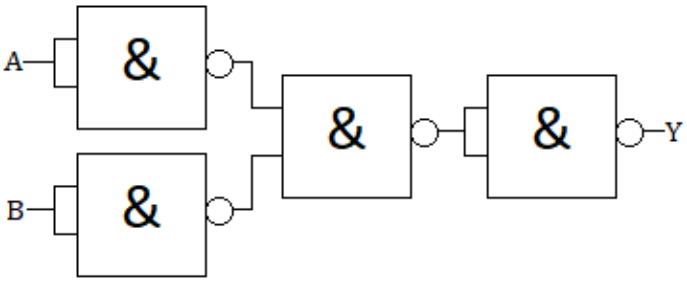
- För att skapa NOR-grinden med NAND-logik så utnyttjar vi De Morgans teorem, dvs. (A' * B')' = A + B.
- Vi inverterar därmed insignalerna A och B med NOT-grindar innan de når NAND-grinden.



$$Y = (A' * B')' = A + B$$

NOR

- Som OR, fast med en inverterare adderad i sista steget.



$$Y = (A' * B')' = A + B$$

XOR

- För att skapa XOR-grinden så måste vi omvandla uttrycket för XOR-grinden med de Morgans teorem, se nedan:

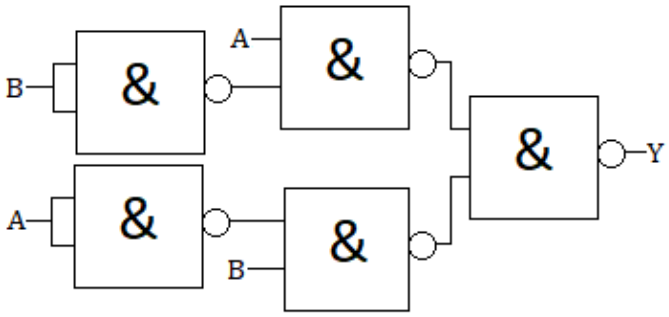
$$Y = AB' + A'B = [(AB')' * (A'B)']'$$

- Därmed ser vi att insignalerna på den sista NAND-grinden skall bestå utav (AB')' respektive (A'B)'. Detta kan vi enkelt ordna med var sin NAND-grind:

    1. Den första NAND-grinden skall realisera (AB')' = (A * B')'. Denna grind skall alltså ha insignalerna A och B'. Vi inverterar B' med en NOT-grind.

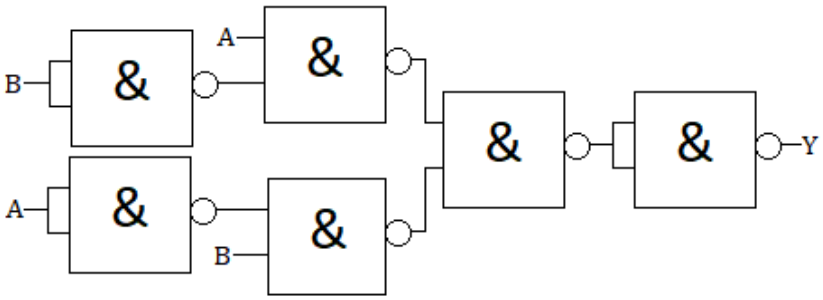
    2. Den andra NAND-grinden skall realisera (A'B)' = (A' * B)'. Denna grind skall alltså ha insignalerna A' och B. Vi inverterar B' med en NOT-grind.

- Därmed realiserar XOR-grinden med följande grindnät:



XNOR

- Som XNOR- men vi lägger till en inverterare (NOT-grind) på utgången:



$$Y = (AB' + A'B)' = [((AB')' * (A'B)')']'$$