Vorlesung 4

Themen dieser Vorlesung sind

- Komplexere CMOS Digitalzellen
- Kombinatorische Logik
- NAND, NOR…
- Multiplexer, Dekoder…

In dieser Vorlesung werden komplexere Digitalzellen beschreiben.

Erinnerung:

Es wird zwischen kombinatorischer und sequenzieller Logik unterschieden. Kombinatorische Logik wird nur aus den Schaltfunktionen (logische Gatter) aufgebaut. Der Ausgang hängt nur von den Eingängen ab. Für die sequenzielle Logik werden noch die Speicherzellen gebraucht. Sie werden am meistens mit den Flip-Flips (bzw. mit den Registern) realisiert.

Kombinatorische Schaltungen

Wir haben in Vorlesung 1, gezeigt dass man alle n-stelligen logische Funktionen als disjunktive Normalform (DNF) darstellen kann. DNF kann mit n-stelligen UND- und ODER-Gattern und mit Invertern realisiert werden. Betrachten wir jetzt als Spezialfälle die Funktionen von weiniger und gleich zwei Variablen (zweistellige Verknüpfungen). Diese Funktionen sind besonders wichtig, da sie für Bitweise-Operationen verwendet werden.

Konstanten

Fangen wir zuerst mit trivialsten Funktionen an – den Konstanten.

Logisch 0 (Kontradiktion) kann man schaltungstechnisch als eine Leitung realisieren, die an GND angeschlossen ist. Logisch 1 (Tautologie) als Leitung die mit VDD verbunden ist.

Die Verbindung wir oft nicht durch Kurzschließen, sondern mittels eines MOFET Widerstands realisiert (). Deswegen sind die logischen Konstanten echte Standardzellen, die im Synthese-Prozess platziert werden müssen.

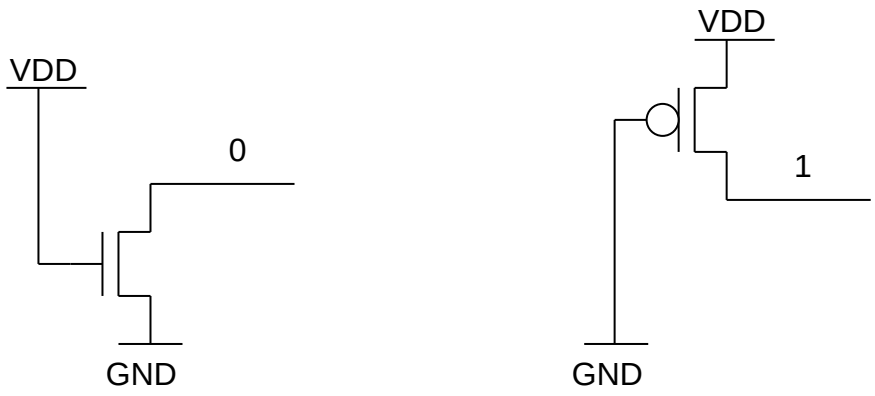


Abbildung 1: Realisierung von logischen Konstanten

Einstellige Funktionen

Es gibt zwei Funktionen einer Variable, Negation Y = !a und Identität Y = a.

Negation wird mit einem Inverter realisiert (Abbildung 2).

Identität kann entweder mit einer geraden Zahl von Invertern (buffer) oder mit einer Leitung realisiert werden. Ein buffer entlastet den Eingang und wird benutzt, wenn Signal a an viele logische Zellen angeschlossen ist.

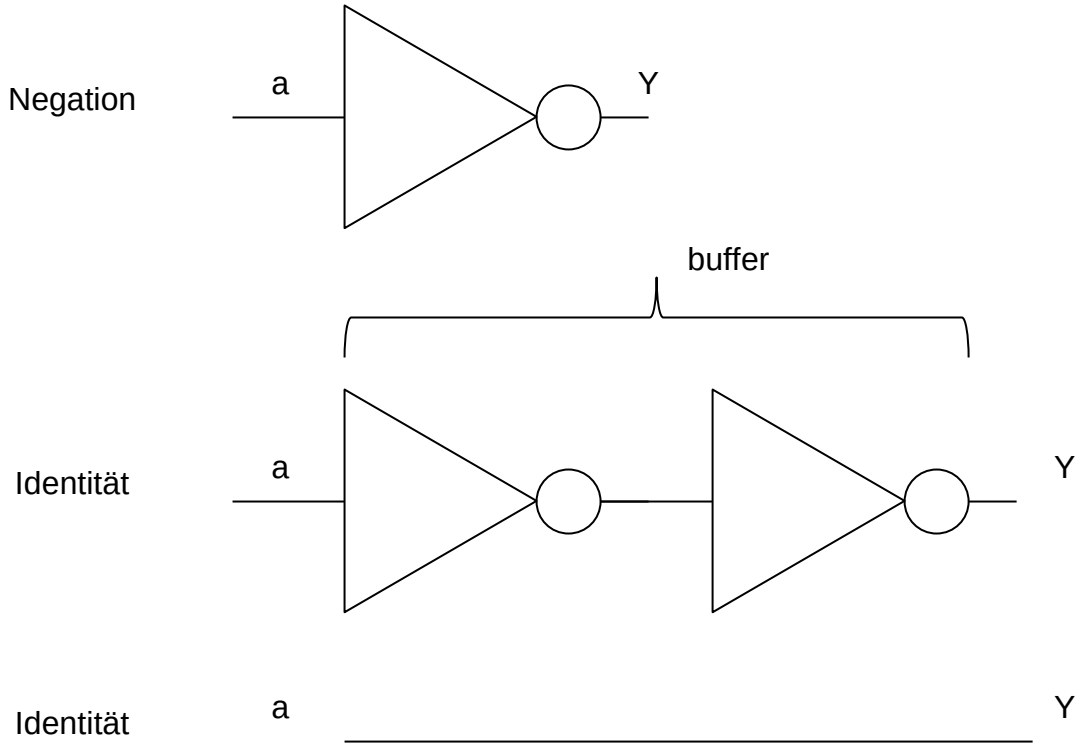
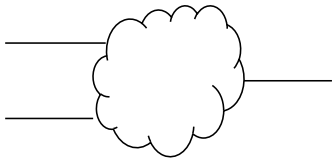


Abbildung 2: Realisierung von einstelligen Funktionen

Zweistellige Funktionen

Es gibt $2^4 = 16$ Booleschen Funktionen von zwei Variablen. Die Länge der Ergebnistabelle ist 4 und für jede Zeile haben wir zwei Möglichkeiten.



a	b	y
0	0	0/1
0	1	0/1
1	0	0/1
1	1	0/1

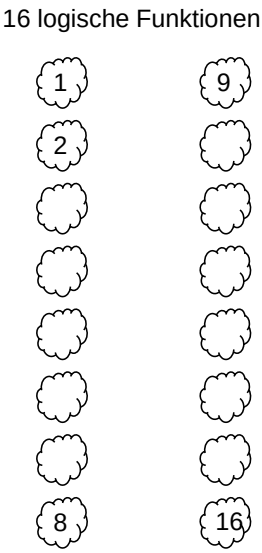
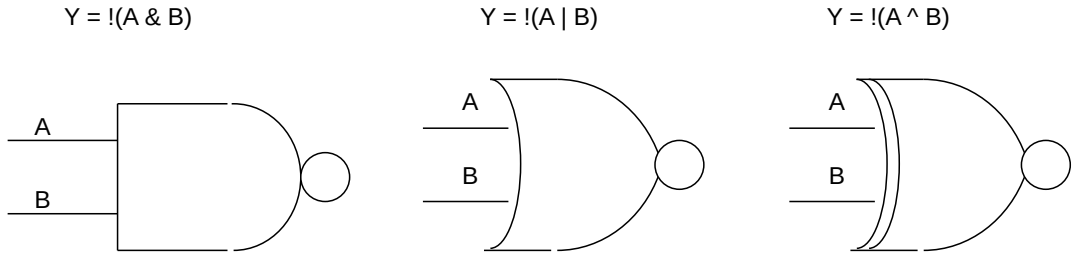


Abbildung 3: Es gibt $2^4 = 16$ Booleschen Funktionen von zwei Variablen.

Die wichtigsten Booleschen Funktionen mit zwei Variablen sind NAND, NOR und EXOR (Gleichwertigkeit, Äquivalenz) - Abbildung 4.



NAND

a	b	y
0	0	1
0	1	1
1	0	1
1	1	0

NOR

a	b	y
0	0	1
0	1	0
1	0	0
1	1	0

EXNOR

a	b	y
0	0	1
0	1	0
1	0	0
1	1	1

Abbildung 4; NAND, NOR und EXNOR

Wir können mittel Inverter, aus NAND, NOR und EXNOR, die Konjunktion AND, Disjunktion OR und die EXOR realisieren (Abbildung 5).

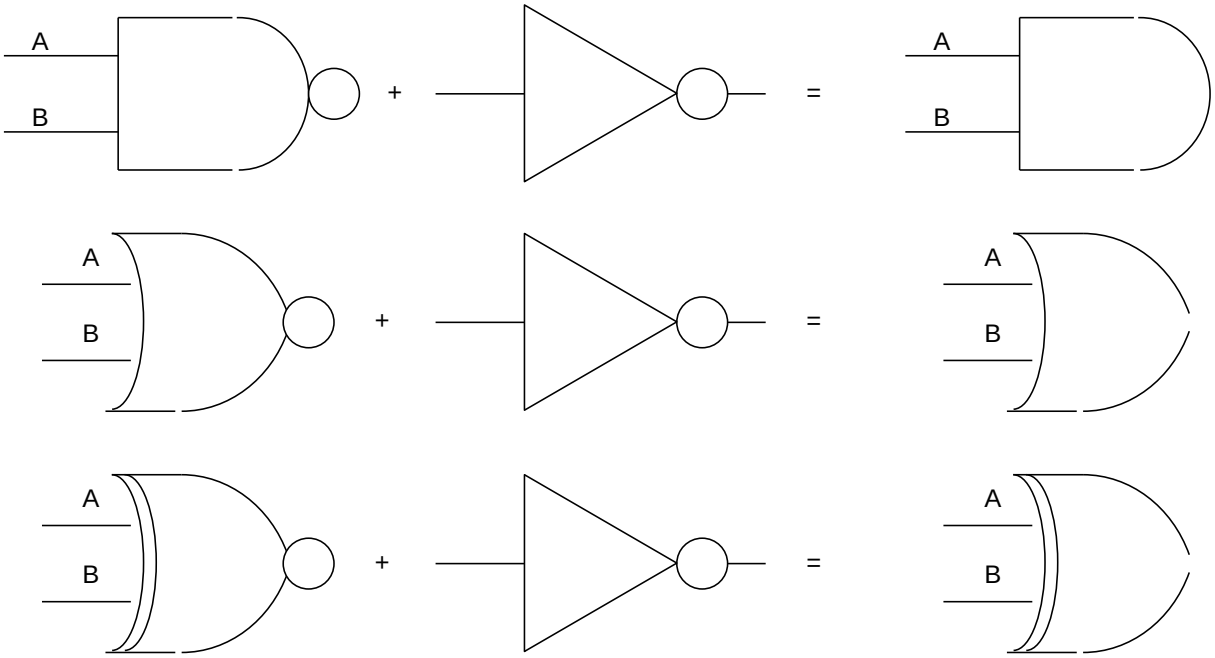


Abbildung 5: Realisierung von AND, OR und EXOR

Sind NAND, NOR, EXNOR und Inverter ausreichend, um alle zweistelligen Funktionen darzustellen (auch ohne DNF)?

Acht Booleschen Funktionen (Gruppe B) kann man durch Negation aus anderen acht (Gruppe A) bekommen - wie AND aus NAND (Abbildung 6).

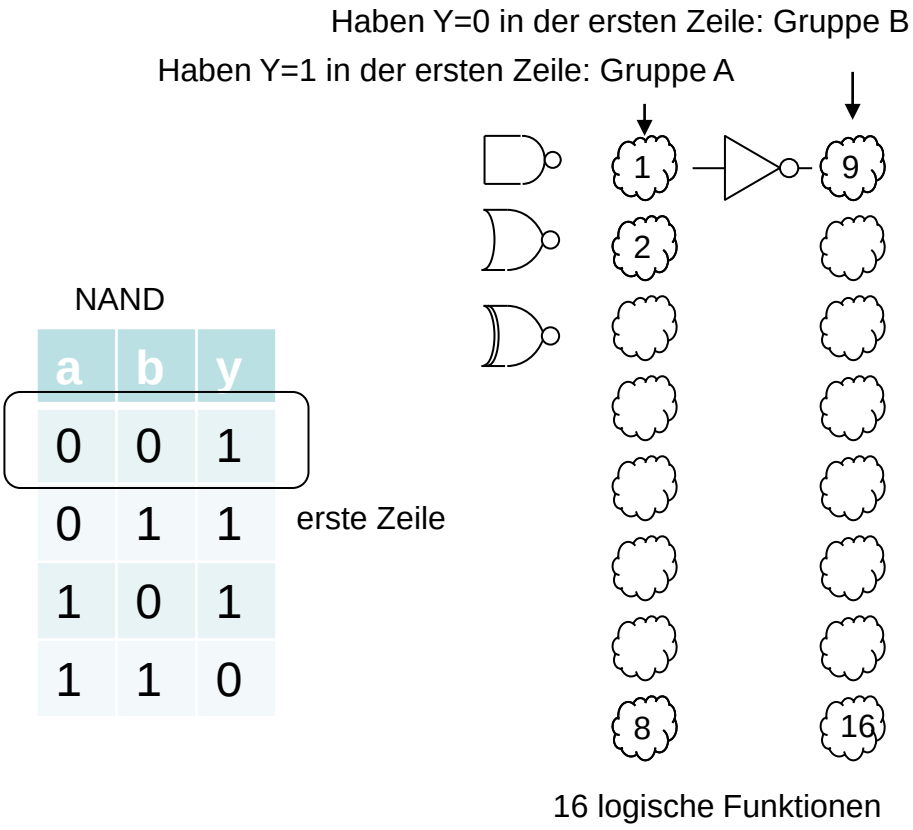


Abbildung 6: Acht Booleschen Funktionen (Gruppe B) kann man durch Negation aus anderen acht (Gruppe A) bekommen.

Zwei Funktionen (von der Gruppe A) sind eigentlich keine Funktionen von zwei sondern nur von einer Variable (Negation von a und Negation von b). Eine Funktion von 8 ist die Konstante 1 (Tautologie). Diese Funktionen sind in Abbildung 7 gezeigt.

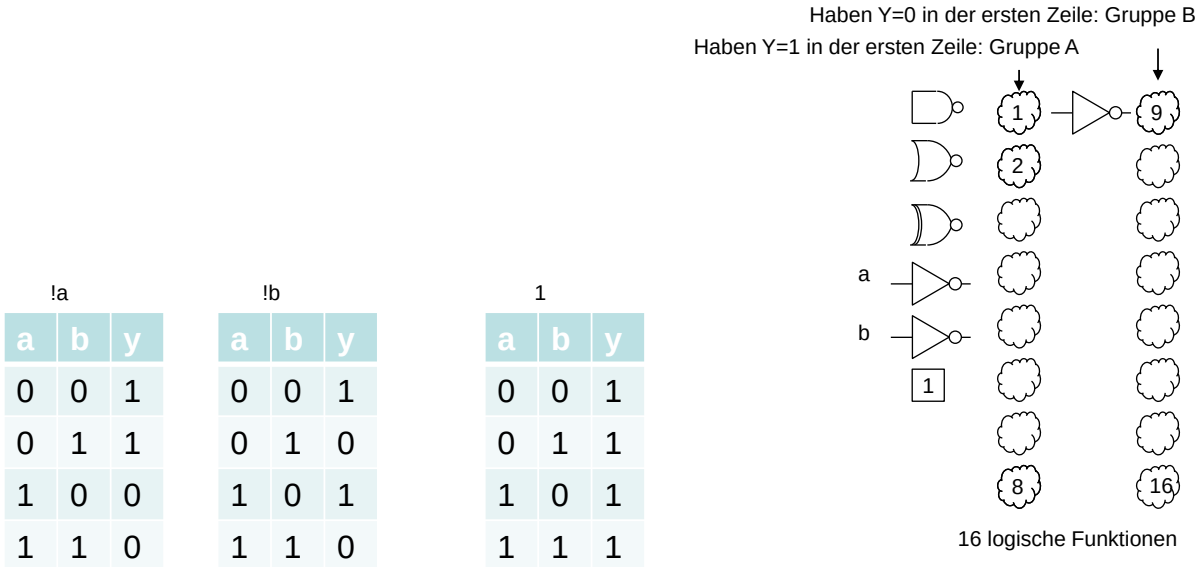
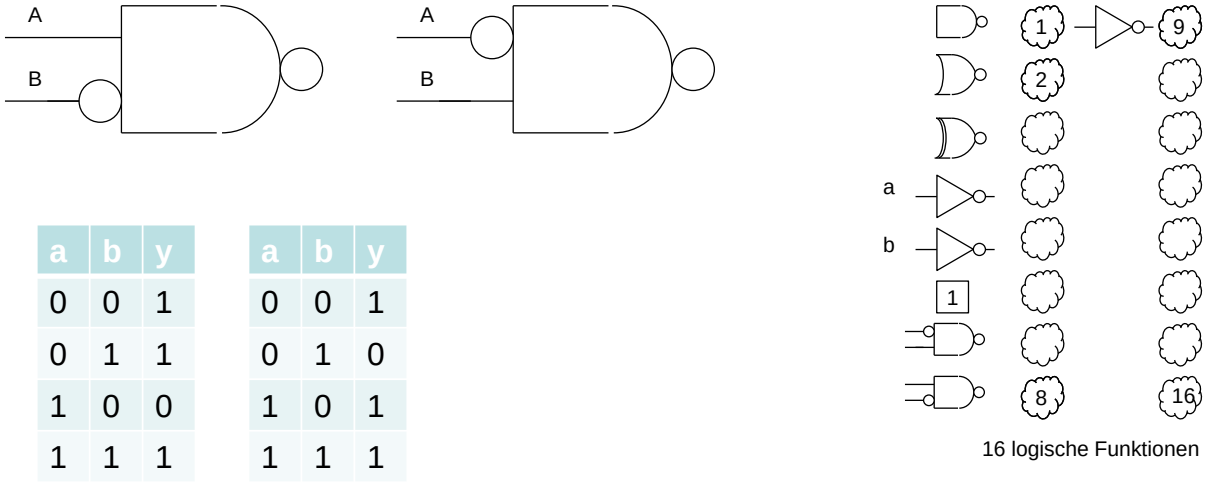


Abbildung 7

Mit NAND, NOR, EXNOR, !A, !B, und 1 (und deren Negationen) fehlen uns noch 2 Funktionen. Diese können aus NAND und NOR mit invertierten Eingängen hergeleitet werden (Abbildung 8). Diese Funktionen nennt man Inhibitionen.

Das heißt NAND, NOR, EXOR und Inverter sind ausreichend, um alle zweistelligen Funktionen zu realisieren.



a	b	y
0	0	1
0	1	1
1	0	0
1	1	1

a	b	y
0	0	1
0	1	0
1	0	1
1	1	1

Abbildung 8: Inhibitionen

EXNOR kann als DNF mit (N)AND, (N)OR und Inverter realisiert werden (Abbildung 9).

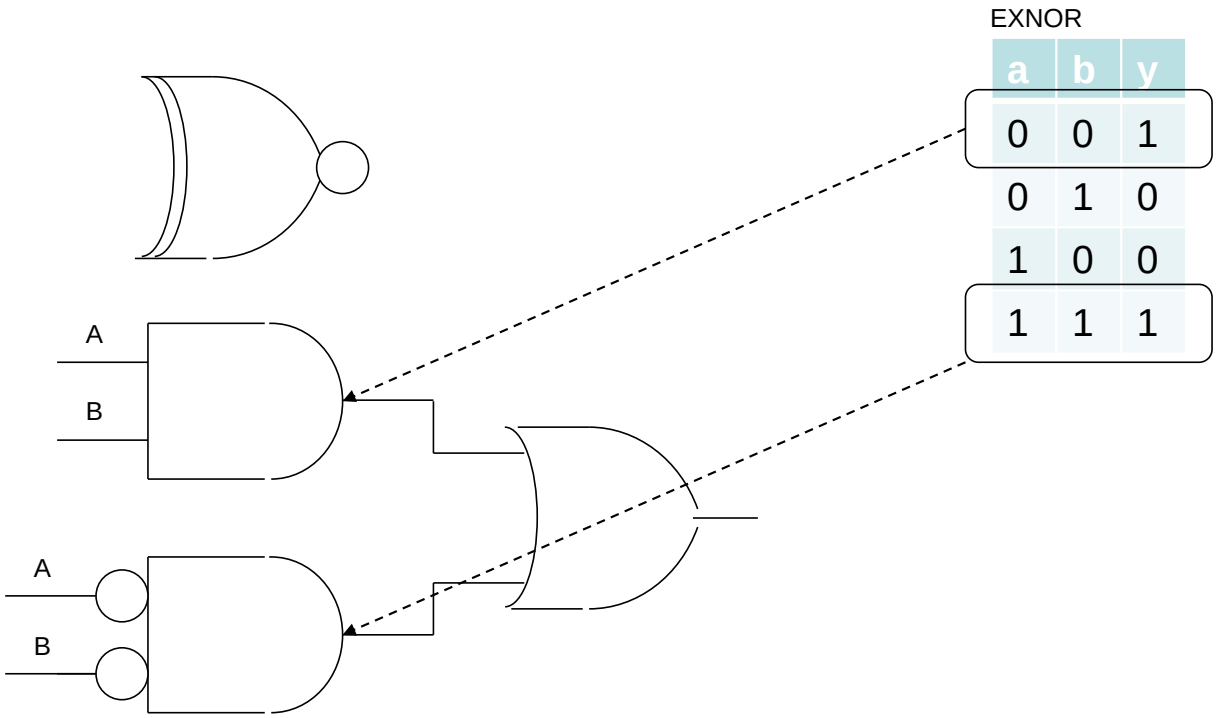


Abbildung 9: EXNOR als DNF

Es ist möglich NAND in NOR umzuwandeln.

De Morgansche Gesetze:

$$!(a \,\&\, b) = !a \mid !b$$

$$!(a \mid b) = !a \,\&\, !b$$

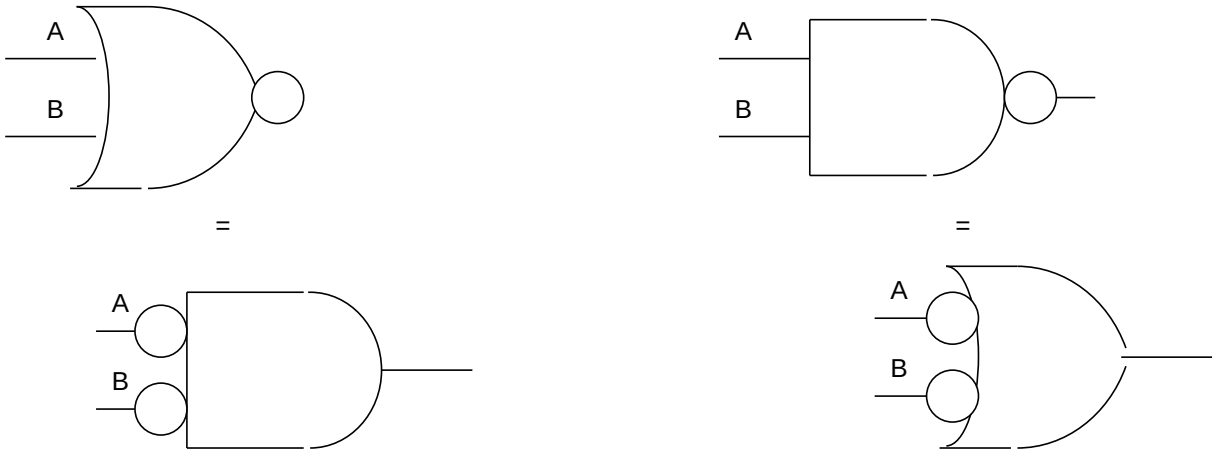


Abbildung 10: De Morgansche Gesetze

Inverter kann man mit NAND realisieren.

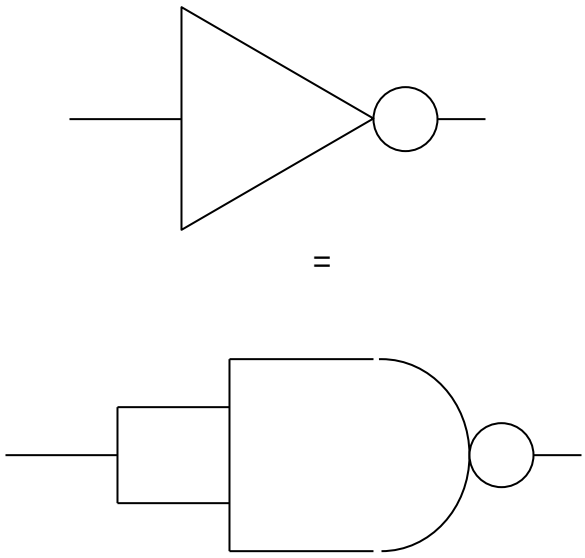


Abbildung 11: Inverter als NAND

Alle zweistelligen Funktionen können basierend auf NAND realisiert werden.
Da mehrstelligen AND und UND-Funktionen als Kaskade von zweistelligen AND und UND-Gattern realisiert werden können und da beliebige Funktion als DNF realisiert werden kann, können alle logischen Funktionen mit zweistelligen NANDs realisiert werden.

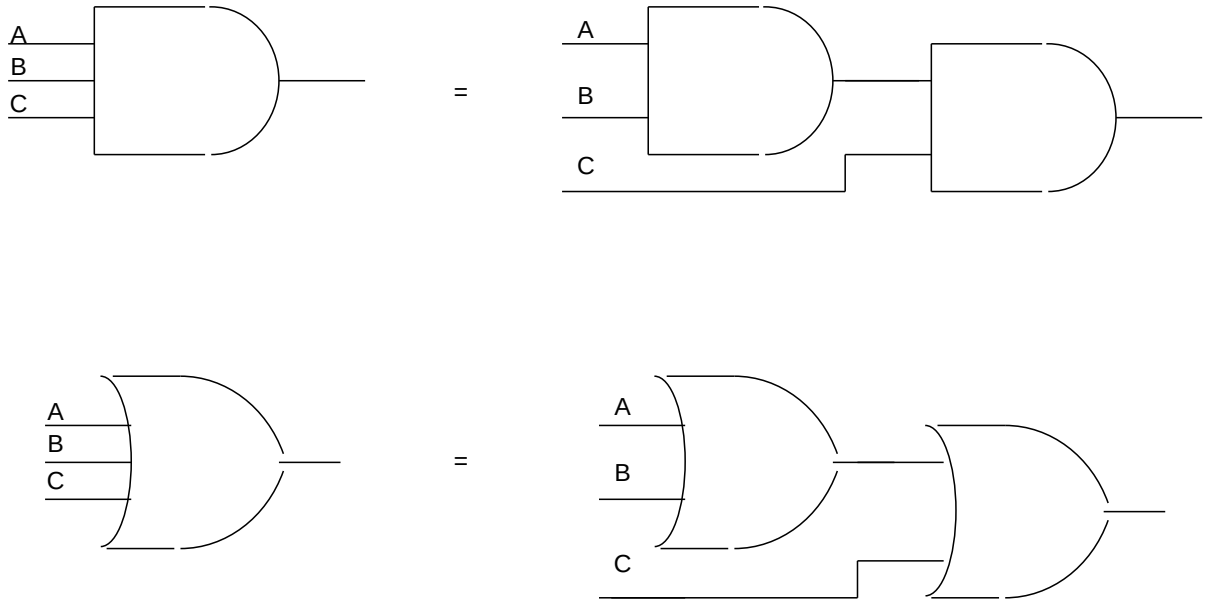


Abbildung 12: Mehrstellige UND und ODER Gatter

Interessante Frage:

Kann eine n-Stellige Funktion (Eingänge a_0, a_{n-1}) als zweistellige Funktion von einer n-1-stelligen Funktion (Eingänge a_0, a_{n-2}) und der Variable a_{n-1} dargestellt werden? (Abbildung 13)

In Allgemeinfall nein, aber in Wirklichkeit oft, wenn die Variablen $a_0 – a_{n-2}$ von a_{n-1} unabhängig sind.

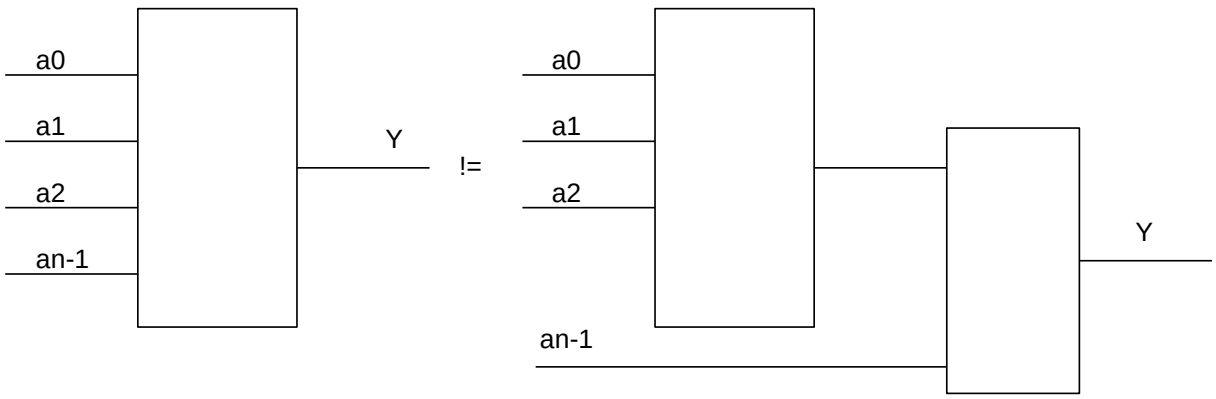


Abbildung 13:

NAND und NOR Implementierungen

CMOS Gatter

NAND und NOR als Schalter-Widerstand (RT-) Logik wurden bereits in Vorlesung 1 gezeigt. Es sind zwei Schalter in Serie oder Parallel geschaltet mit einem Pullup Widerstand (Abbildung 14).

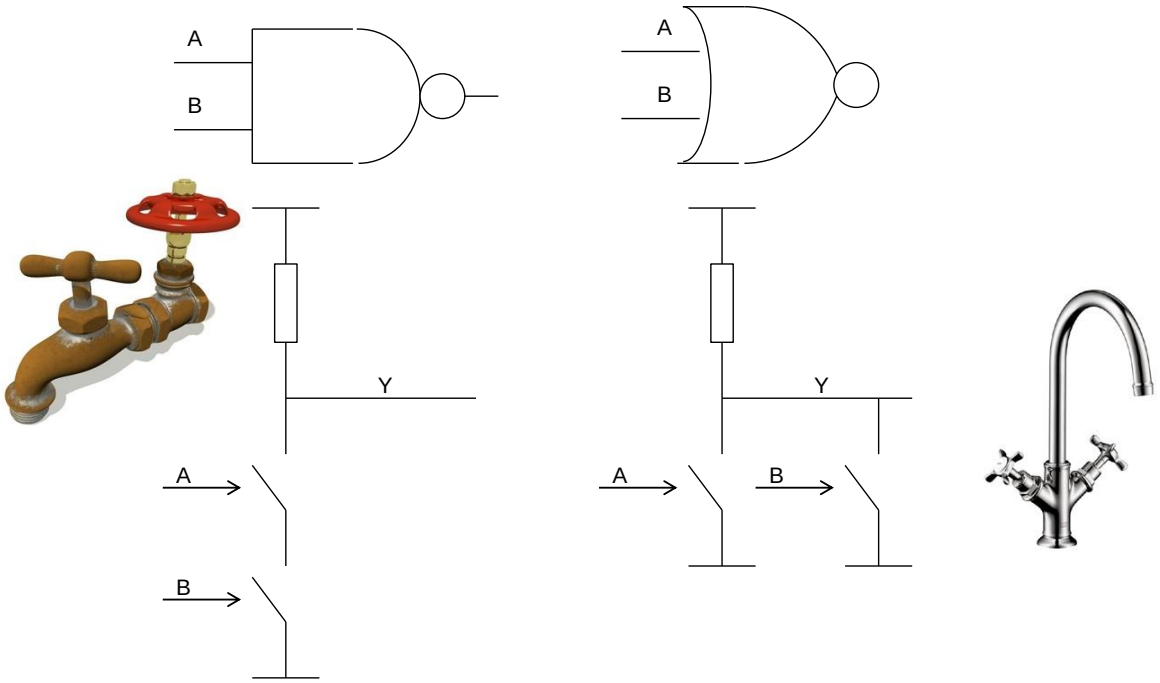


Abbildung 14: Links: RTL NAND. Rechts: NOR

In Vorlesung 2 wurde gezeigt, dass man einen Inverter als RT-Logik entweder mit einem NMOS und dem Pullup- oder mit einem PMOS und dem Pulldown Widerstand aufbauen kann. Wenn man die zwei Invertern kombiniert, bekommt man einen CMOS Inverter (Abbildung 15). Die Vorteile sind kein DC Strom und ein kleines Layout.

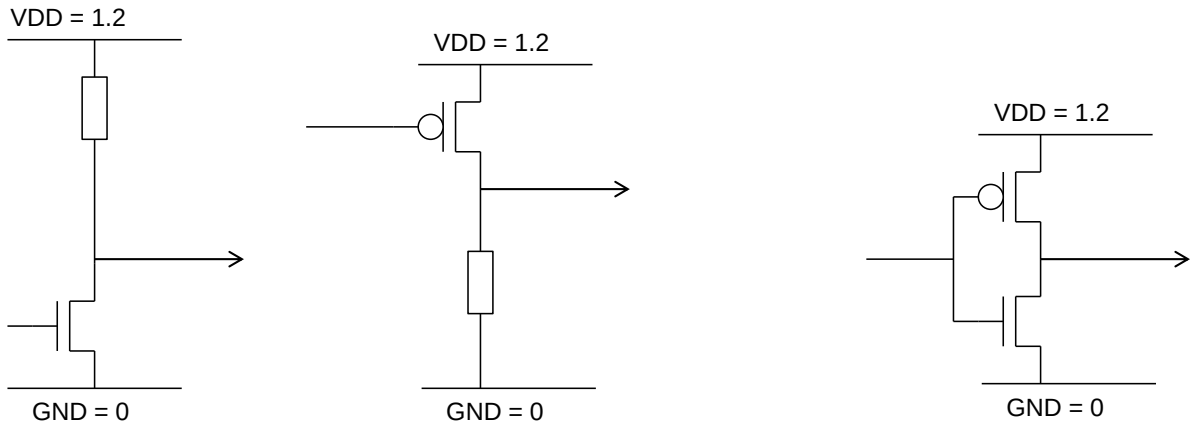


Abbildung 15: Inverter

Man kann auf ähnliche Weise auch NAND und NOR als CMOS aufbauen (Abbildung 16).

Hier ist folgendes zu beachten:

Wenn der NMOS Teil für bestimmte Zeilen in der Ergebnisstabelle leitet (für die Zeilen wo wir null-Ergebnis haben), muss der PMOS Teil für genau andere Zeilen in der Tabelle leiten. Es

darf nicht passieren, dass PMOS und NMOS Schaltnetz für manche Eingangskombinationen gleichzeitig leiten. Wir hätten dann einen großen „Querstrom“ und der Ausgang wäre undefiniert. PMOS und NMOS Teil sollen auch nie gleichzeitig offene Verbindungen sein. In dem Fall wäre der Ausgang von Versorgungslinien getrennt. Der logische Wert wäre ebenfalls undefiniert. Hier eine Bemerkung: Für ein Gate mit dem offenen Ausgang sagt man, dass es sich im hochohmigen Zustand befindet. Wir werden später auf die Anwendungen solcher Gatter eingehen. Im Moment werden wir ein solches Verhalten ausschließen.

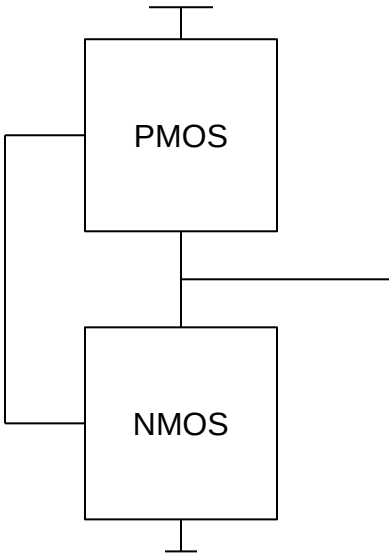


Abbildung 16: CMOS Gates

Wie wird ein CMOS Gate in Praxis realisiert? Jede Zeile mit dem Ergebnis 0 ist die Serienschaltung von zwei (oder mehreren) NMOS Transistoren, die nur für die Eingangswerte dieser Zeile leiten (Abbildung 17). Man muss alle Eingänge, die Null sind, zuerst invertieren und dann an NMOS Gates anschließen. Das ganze NMOS Netzwerk ist die Parallelschaltung von allen Reihenschaltungen die allen Zeilen = 0 entsprechen.

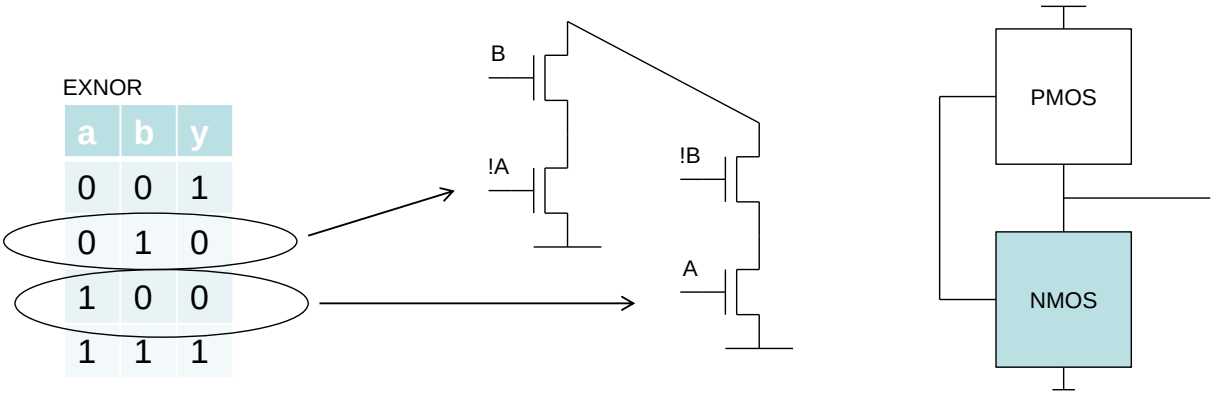


Abbildung 17: Realisierung vom NMOS Teil eines CMOS Gates.

PMOS Teil macht man dual (Abbildung 18).

Beachten wir, dass PMOS für niedriges Gate-Potential leitet

Man muss alle Eingänge, die logisch eins sind, invertieren.

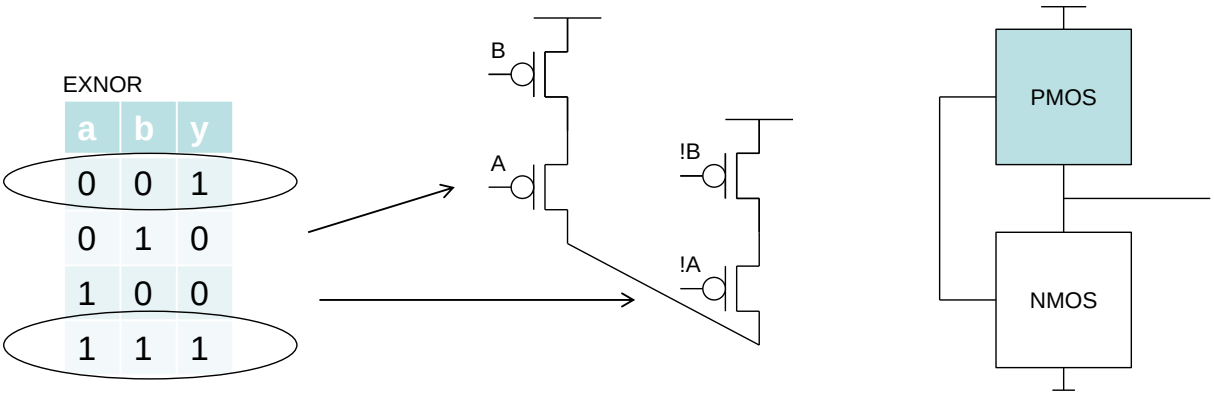


Abbildung 18: Realisierung vom PMOS Teil eines CMOS Gates

Abbildung 19 zeigt EXNOR, das nach dem obigen Rezept realisiert wurde.

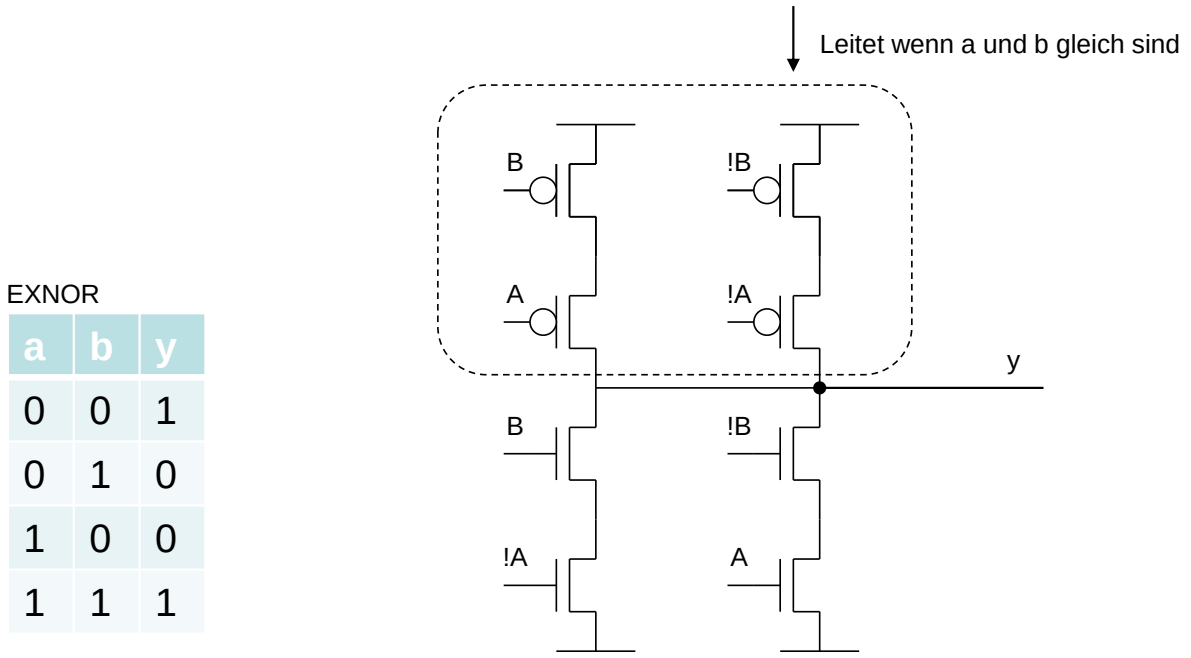


Abbildung 19: EXNOR als CMOS Gate

Oft kann man logische Funktion vereinfachen.

Abbildung 20 zeigt den Aufbau des CMOS NOR-Gates.

PMOS Netzwerk leitet für die Eingangskombination 00 – wir haben die Reihenschaltung. NMOS Netzwerk leitet immer außer für 00 – wir haben die Parallelschaltung.

Eigentlich, wenn man nach dem obigen Rezept vorgehen würde, wäre NOR zuerst sehr kompliziert, wie in der Abbildung 20 dargestellt wird. Man kann aber die Absorptionsregeln verwenden, die zur minimalen Schaltung auf in der Abbildung 22 führen. Beachten wir, dass der Schaltplan im Bezug auf Eingang A und B eine Asymmetrie hat. PMOS mit A befindet sich in unserer Realisierung näher zum Ausgang Y.

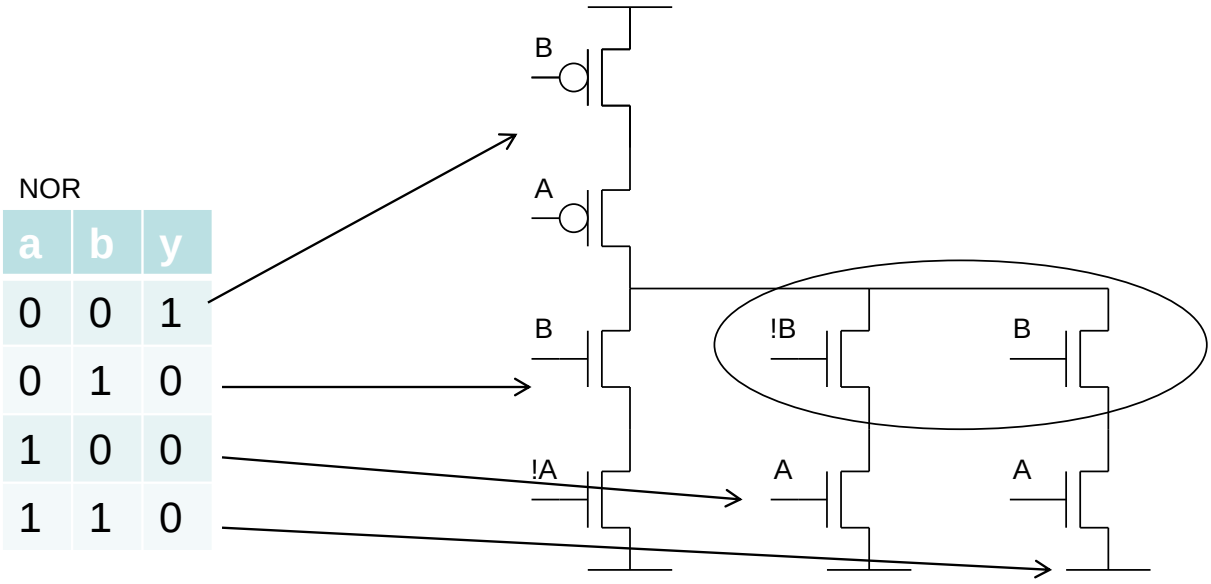


Abbildung 20: NOR

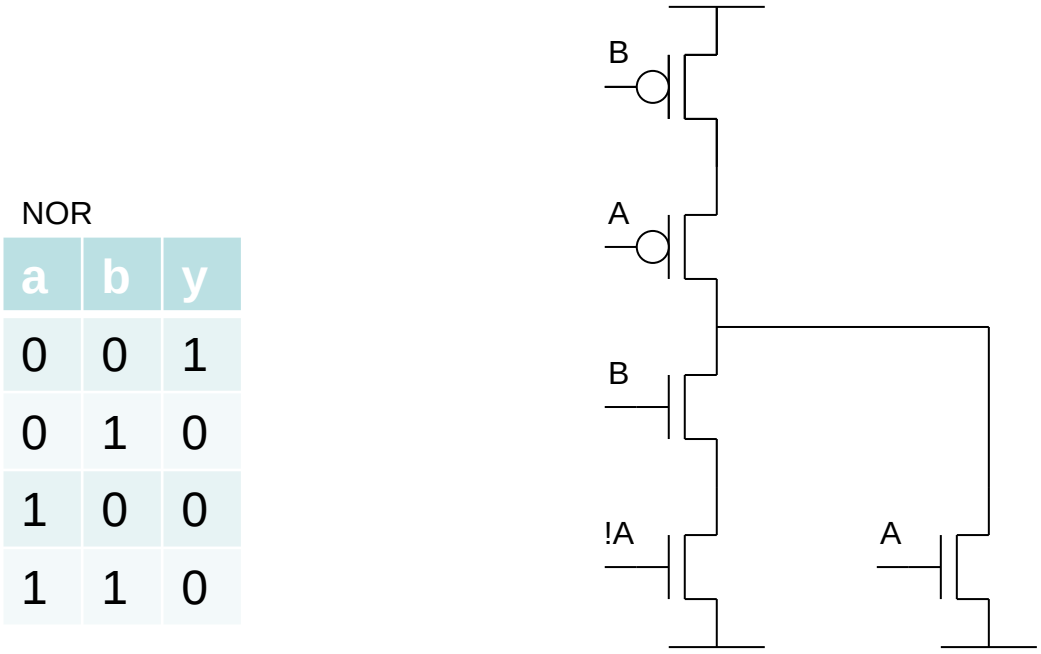


Abbildung 21: NOR

NOR

a	b	y
0	0	1
0	1	0
1	0	0
1	1	0

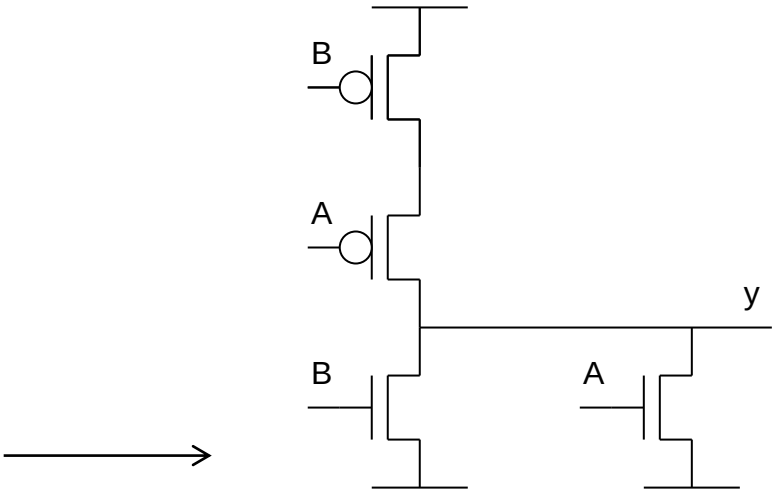


Abbildung 22: NOR

Abbildung 23 zeigt NAND

NAND

a	b	y
0	0	1
0	1	1
1	0	1
1	1	0

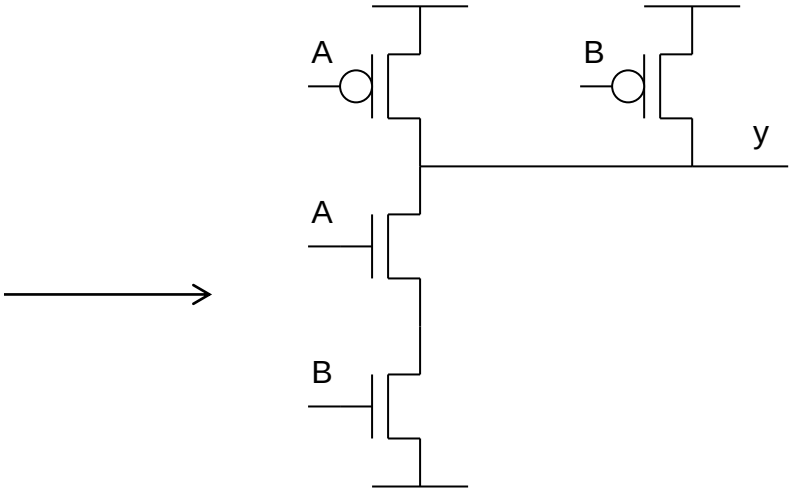


Abbildung 23: NAND

Es ist leicht die NAND und NOR mit mehreren Eingängen zu realisieren.

Die Schaltpläne sind in Abbildung 24 gezeichnet.

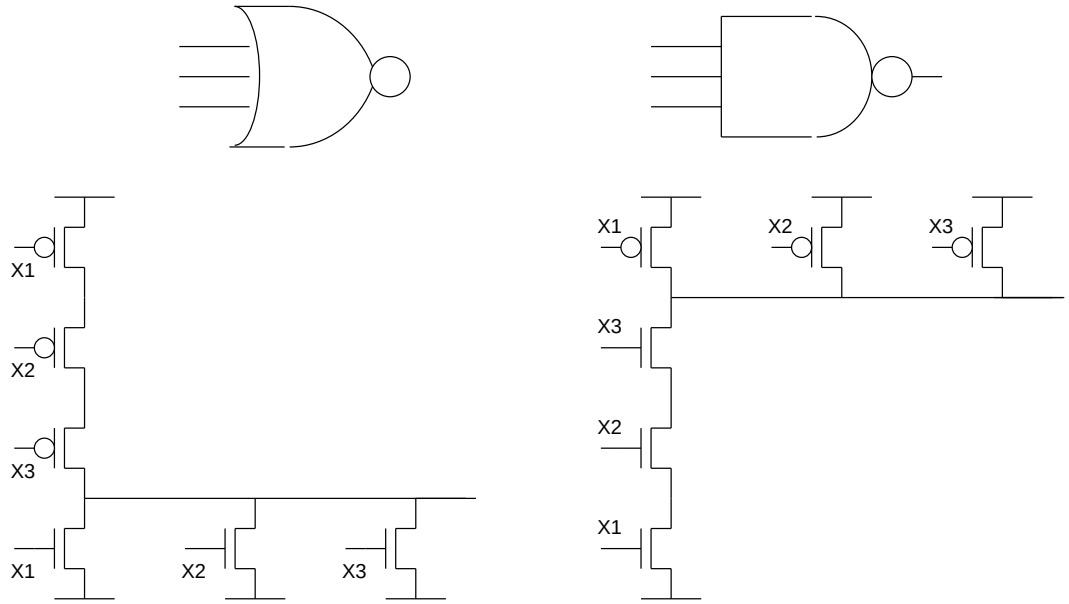


Abbildung 24: NAND und NOR mit mehreren Eingängen.

Regel für den Aufbau eines CMOS Gates

Einige Regeln für die Konstruktion von CMOS Gattern aus Ergebnistabelle.

NMOS Teil leitet für die Zeilen mit null-Ergebnis.

PMOS Teil leitet für die Zeilen mit eins-Ergebnis.

PMOS und NMOS Teile dürfen nie gleichzeitig leiten, sonst hätten wir einen großen Querstrom und der Ausgang wäre undefiniert.

PMOS und NMOS Teil sollen auch nie gleichzeitig offene Verbindungen sein. In dem Fall wäre der Ausgang von den Versorgungslinien getrennt. Der logische Wert wäre undefiniert.

Ein Gate mit offenem Ausgang befindet sich im hochohmigen Zustand.

Es gibt oft mehrere Möglichkeiten eine Funktion mit Transistoren zu realisieren.

Beispiel EXNOR. Entweder verwenden wir die Schaltung von Abbildung 19 / Abbildung 25. Wir brauchen hier 4 Transistoren in zwei Invertern und 8 Transistoren im EXNOR Netzwerk.

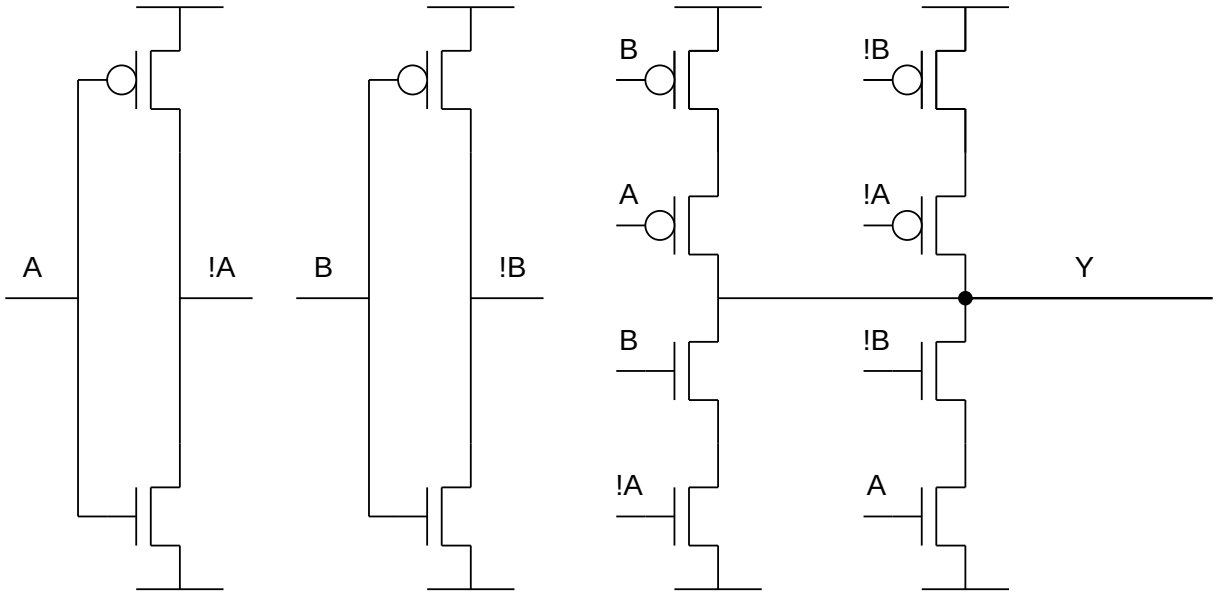
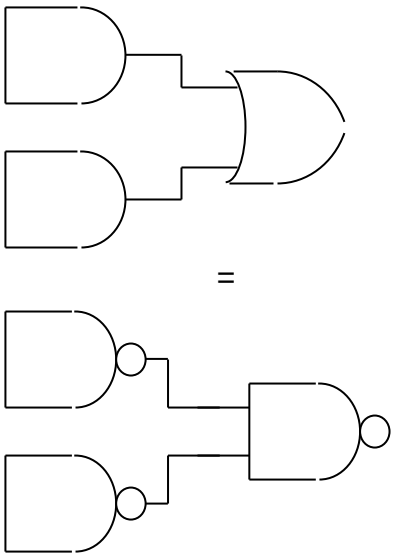


Abbildung 25: EXNOR als CMOS Gate (Variante 1)

Zweite Möglichkeit: wir schreiben die disjunktive Normalform und realisieren sie mit Invertern, AND und OR Gattern.

EXOR = (!A & !B) | (A & B)

Hier bietet sich an, OR in (N)AND umzuwandeln und die Schaltung zu vereinfachen (Abbildung 26). Wir brauchen dann 3 NAND – Gattern und zwei Invertern. Das sind 4 Transistoren in zwei Invertern und 12 Transistoren in drei NANDs. Also die erste Realisierung verwendet weniger Transistoren.



4 (INVs) + 12 (NANDs) Transistoren

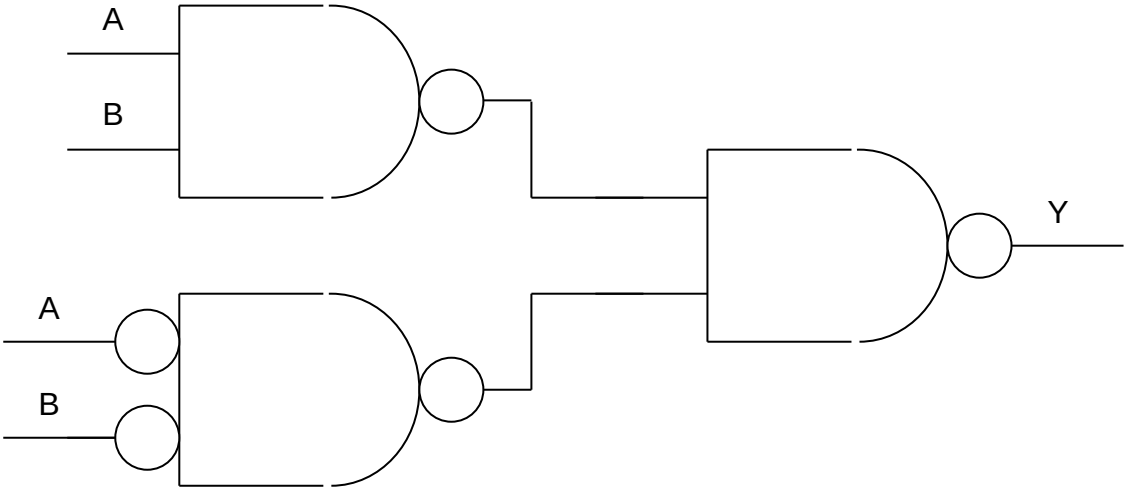


Abbildung 26: EXNOR als CMOS Gate (Variante 2)

Abbildung 27 zeigt eine weitere EXNOR-Variante mit nur 12 Transistoren.

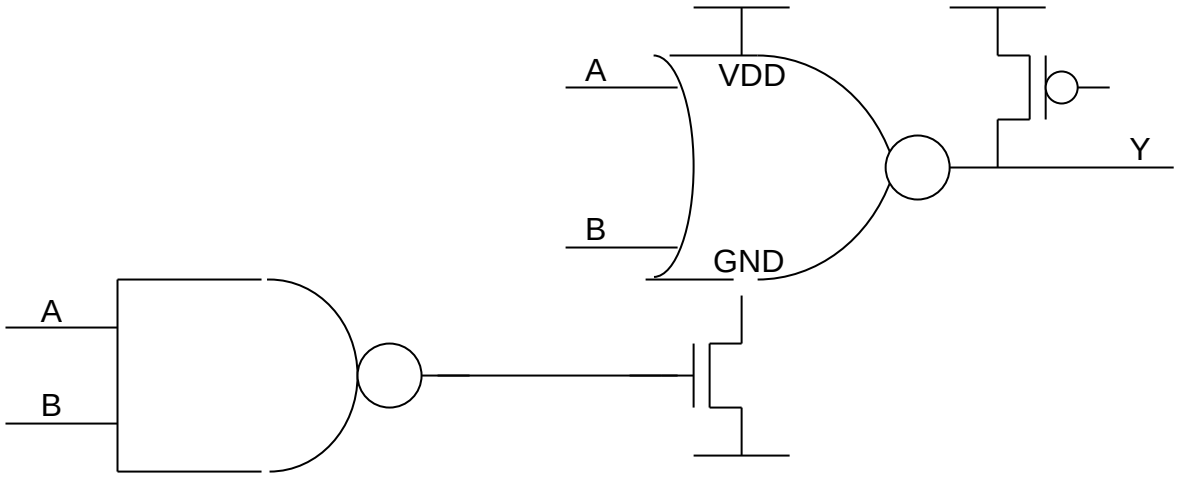


Abbildung 27: EXNOR (Variante 3)

Komplexere Gates mit drei Eingängen

Multiplexer

Vielleicht der wichtigste und vielseitigste Bauteil in Digitaltechnik ist der Multiplexer (Abbildung 28). Je nachdem ob der Select-Eingang null oder eins ist, ist der Ausgang X0 oder X1.

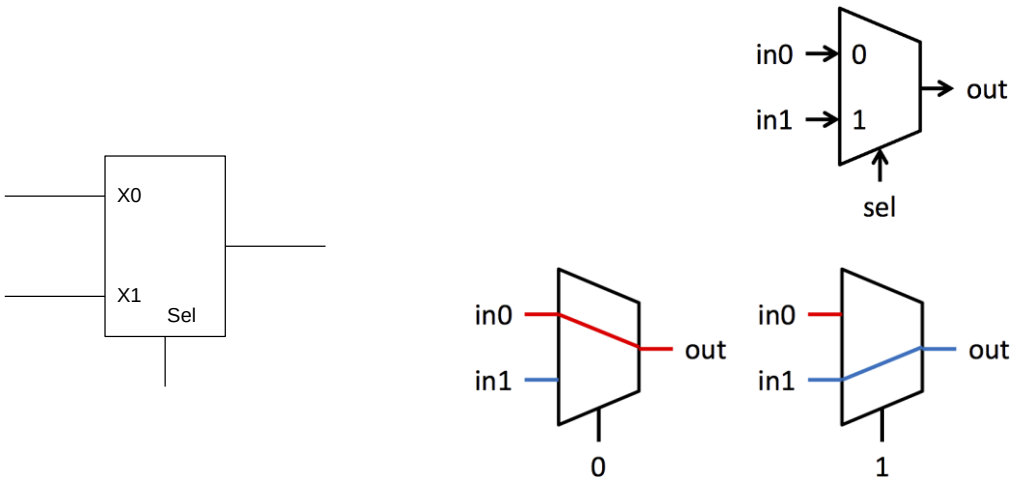


Abbildung 28: Multiplexer

Multiplexer hat auch eigene Notation im Verilog Code

Y = sel ? X1 : X0.

Multiplexer kann wie folgend dargestellt werden: (disjunktive Normalform):

Y = !sel & X0 | sel & X1

Der entsprechende Schaltplan ist in Abbildung 30 gezeigt.

Wir brauchen 3 NAND Gattern und einen Inverter. Das sind 3 × 4 + 2 = 14 Transistoren.

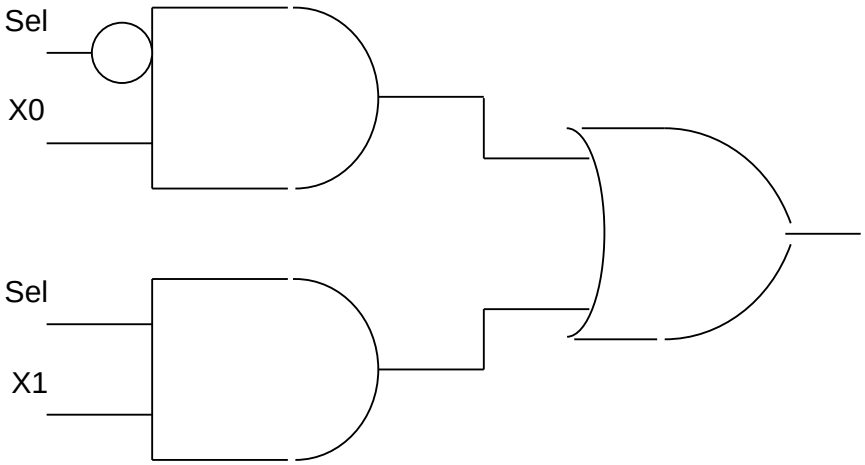


Abbildung 29: Multiplexer als DNF

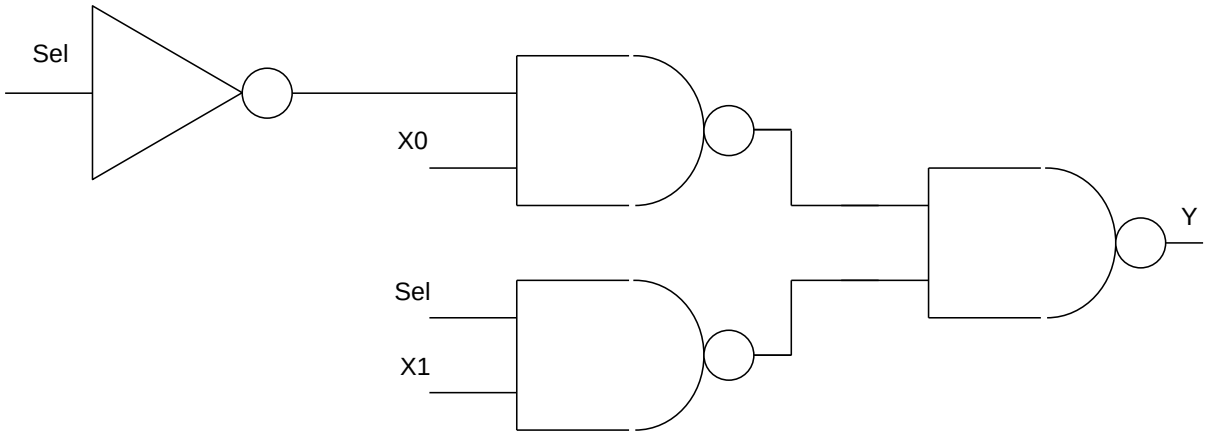


Abbildung 30: Multiplexer als DNF, AND und OR in NAND umgewandelt

Warum ist ein Multiplexer so wichtig? Jede logische Funktion kann mit Multiplexern, Invertern und logischen Konstanten realisiert werden.

Nehmen wir als Beispiel AND.

AND ist null, wenn die Variable A null ist, unabhängig von B. Wir können, also, einen Multiplexer verwenden und A an Select anschließen. An Eingang X0 schließen wir die logische 0. Wenn A eins ist (Select = 1), hängt das Ergebnis von Variable B ab.

AND = A ? B : 0

Variable B wird an Eingang X1 angeschlossen.

Dementsprechend, kann man AND mithilfe eines Multiplexer realisieren. Das ist in Abbildung 31 dargestellt.

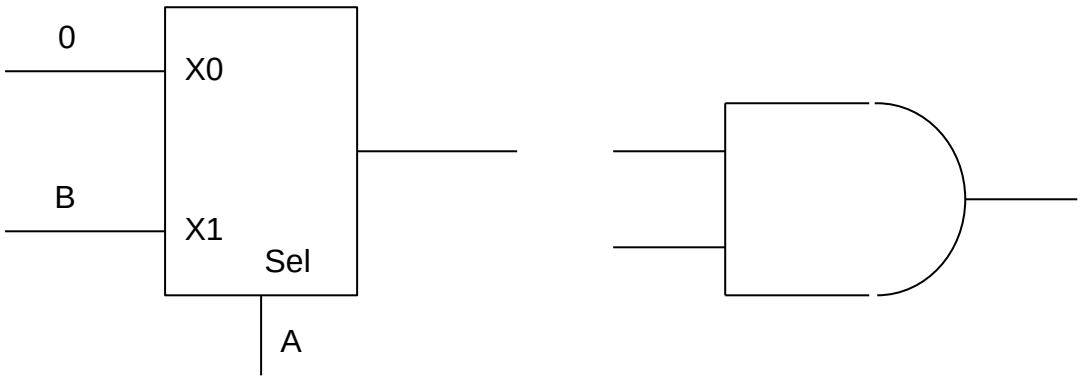


Abbildung 31: AND realisiert mit einem MUX

Ähnlich kann man auch ENXOR (Äquivalenz) realisieren (Abbildung 32).

EXNOR = A ? B : ~B

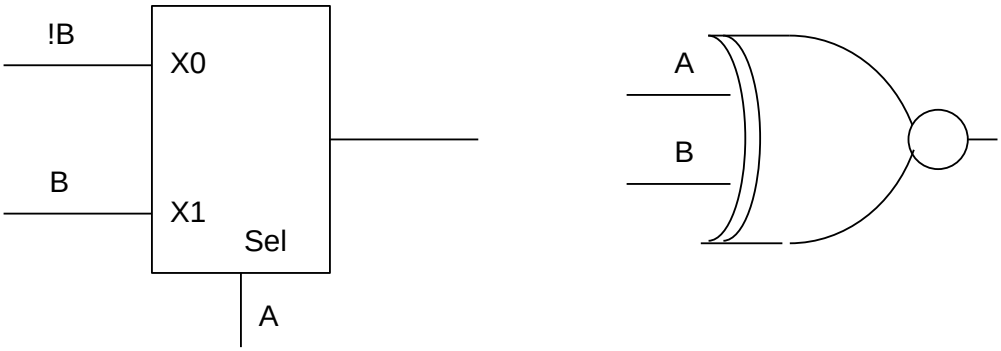


Abbildung 32: EXNOR realisiert mit einem MUX

Multiplexer kann auch noch einfacher realisiert werden.

Gated Inverter

Beachten wir, dass wir bis jetzt nie zwei Gate-Ausgänge kurzgeschlossen haben. (Wir haben auch erwähnt, dass sich, im Fall von Standardgattern, ein Ausgang nie in einem Hochohmigen Zustand befinden darf.) Wenn wir zwei Ausgänge kurzschließen und wenn sie verschiede logische Niveaus haben, wirken die PMOS und NMOS Transistoren gegeneinander. Der PMOS-Teil schließt den Ausgangsknoten mit VDD kurz und NMOS-Teil mit GND. Wir haben einen Querstrom (Abbildung 33).

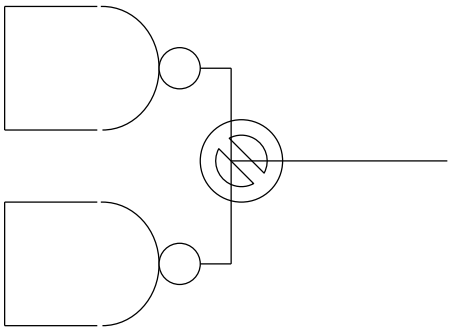


Abbildung 33

Wir können aber ein Gate so erweitern, dass es sich auch in einem hochohmigen Zustand befinden kann. Die Ausgänge solcher Gates können auch kurzgeschlossen werden. Die Bedingung ist es, dass sich in einem Moment nur ein Gate im niederohmigen Zustand befindet.

Das einfachste Beispiel eines Bauteils mit dem zusätzlichen hochohmigen Zustand ist ein so genannter Gated-Inverter.

Wenn der „Enable“ Eingang eins ist, funktioniert der Inverter wie ein ganz gewöhnlicher Inverter. Mit Enable gleich null, ist der Ausgang von VDD und GND „abgeklemmt“ - er schwebt („floatet“) im hochohmigen (high impedance) Zustand.

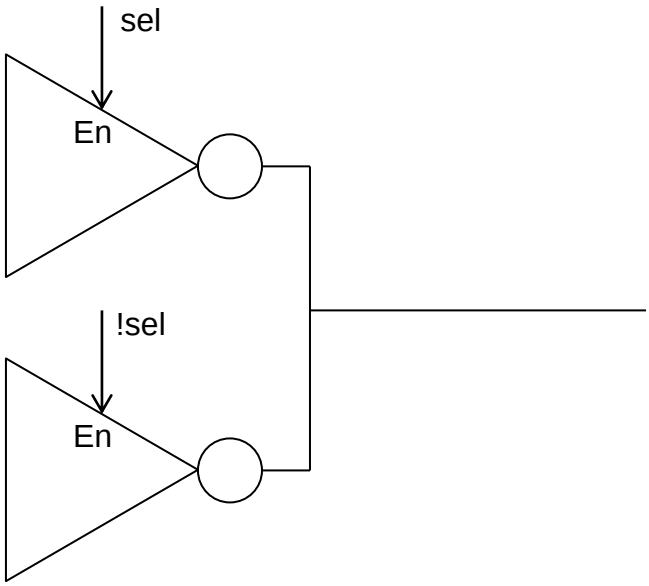


Abbildung 34: Ausgänge von Gated-Invertern dürfen kurzgeschlossen werden.

Abbildung 35 - Abbildung 38 zeigen die Schaltpläne vom Gated-Inverter von der Grundidee bis zur endgültigen Schaltung.

Die Gated-Inverter werden oft benutzt. Zum Beispiel, man kann mit zwei Gated-Invertern, einen Multiplexer realisieren (Abbildung 39).

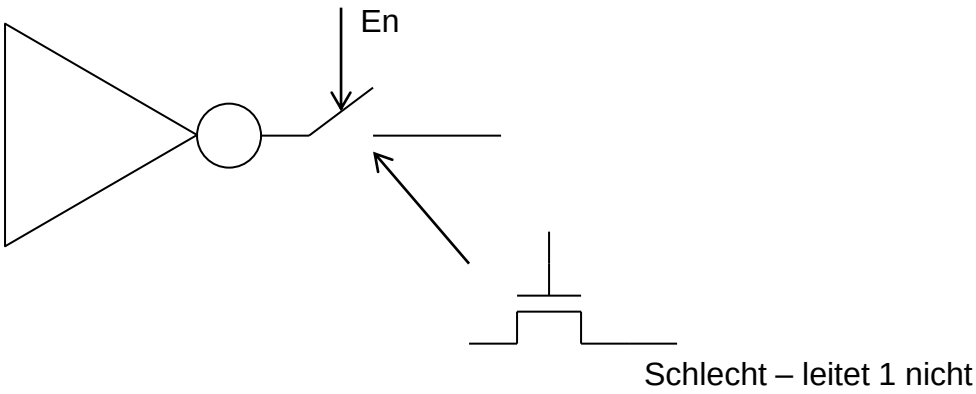


Abbildung 35: Gated Inverter

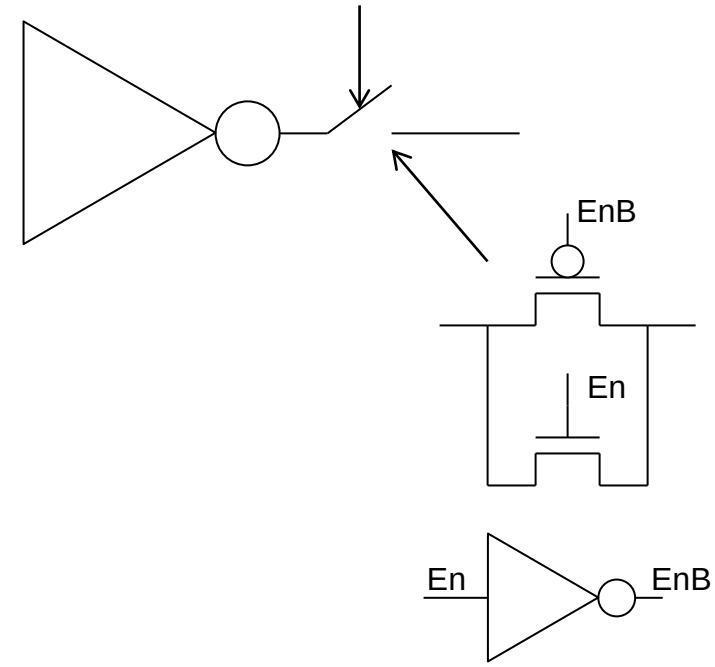


Abbildung 36: Gated Inverter

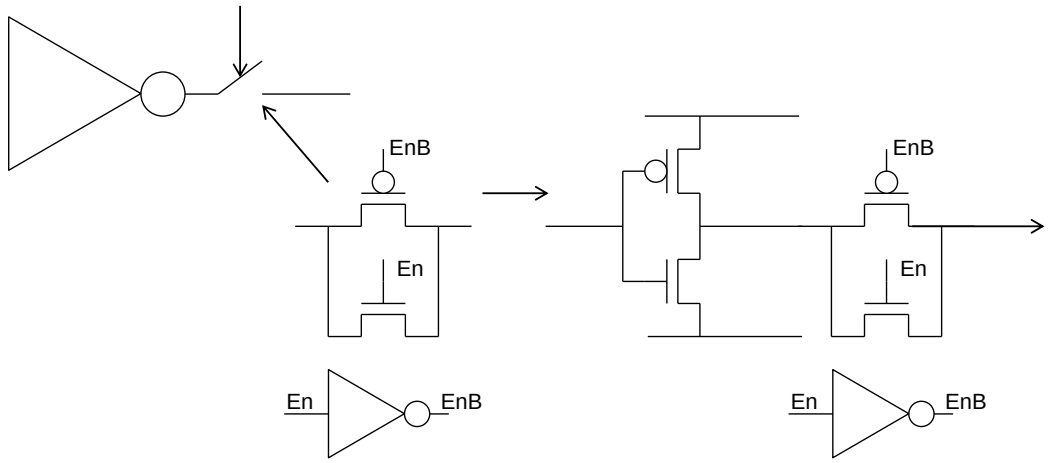


Abbildung 37: Gated Inverter

Leitung wird aufgespalten um Geschwindigkeit zu optimieren

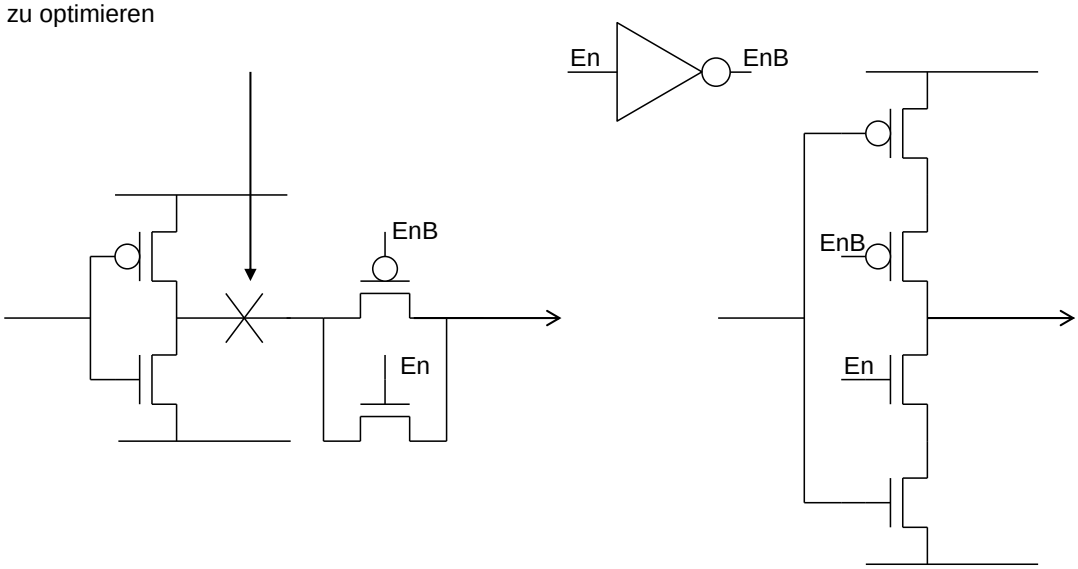


Abbildung 38: Gated Inverter

Die Schaltung des Multiplexers ist in Abbildung 39 gezeigt. Wir brauchen zwei normale- und zwei Gated-Invertern – es sind insgesamt $2 \times 2 + 2 \times 4 = 12$ Transistoren, zwei weniger als in der Abbildung 30.

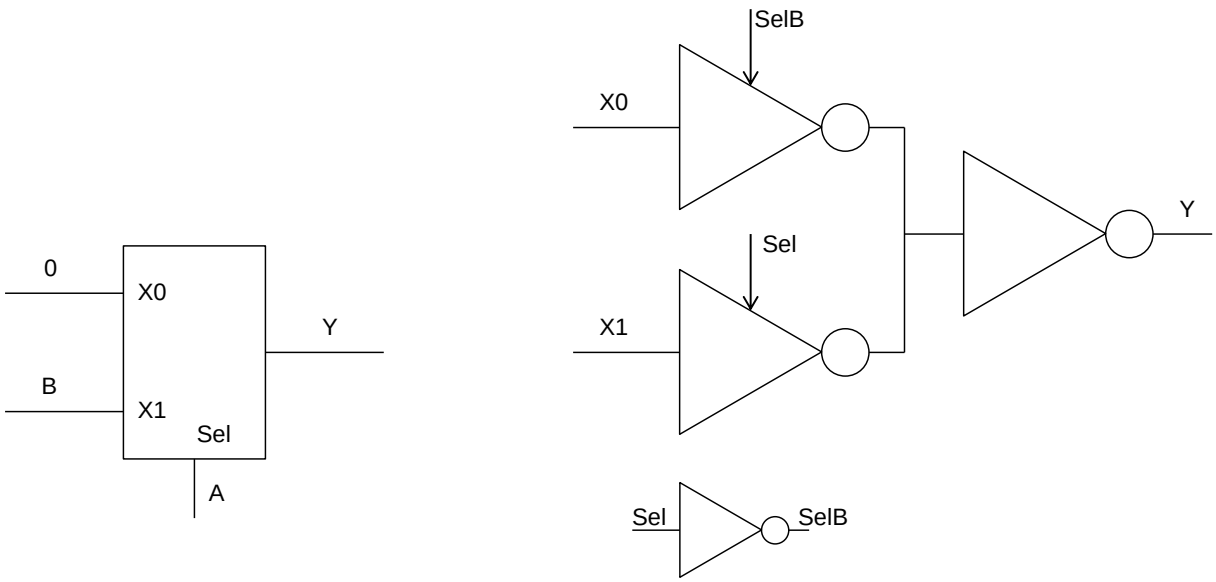


Abbildung 39: MUX realisiert mit den Gated Invertern

Oft werden auch mehrfache Multiplexer verwendet (Abbildung *40*). Ein Anwendungsbeispiel ist es, wenn man die digitalen Signale von mehreren Quellen über eine Leitung übertragen möchte.

Hier könnten wir mehrere Gated-Inverter verwenden (Abbildung *40*). Deren Ausgänge sind kurzgeschlossen. Als Select haben wir normalerweise eine binäre Zahl die der Kanalnummer (der Eingangsnummer) entspricht. Wir betrachten in diesem Fall Select als Bus: Sel = Sel(1:0). Z.B. für Sel(1:0) = 11 wird der Inverter X3 (bzw. Kanal X3) an den Ausgang geschlossen und alle andere Inverter sind im high-impedance Zustand.

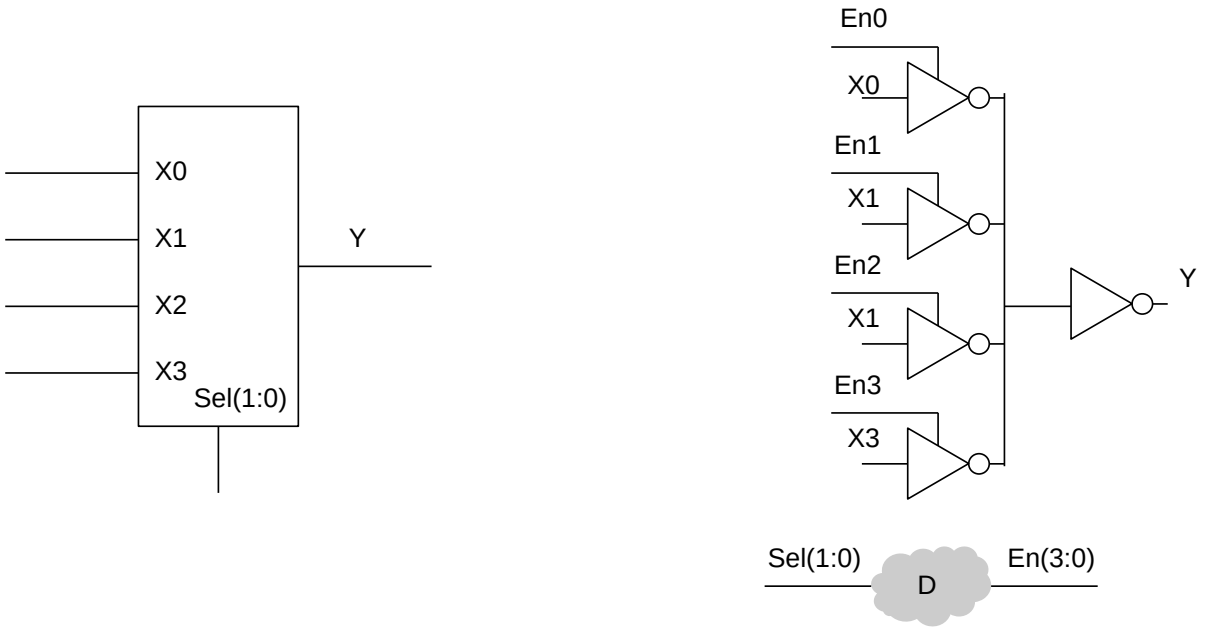


Abbildung 40: Multiplexer mit mehreren Eingängen

Decoder

Um Multiplexer zu realisieren, brauchen wir einen Decoder (Schaltung D in Abbildung *40*). Hier haben wir Z.B. einen 8-bit Eingang D(7:0) und 2^8 Ausgänge. Der Eingang ist eine binäre Zahl. Wenn der Eingang m ist (binär Kodiert), ist der m-te Ausgang 1. Alle anderen Ausgänge sind null.

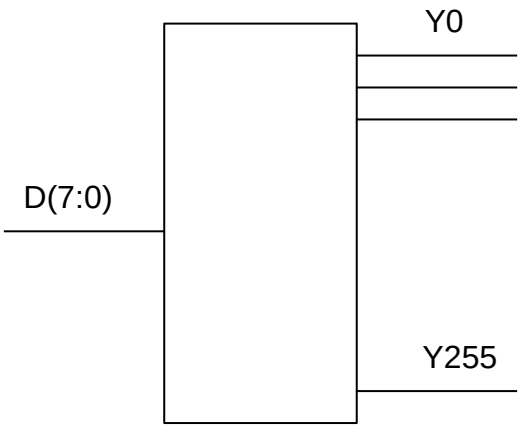


Abbildung 41: Decoder

Den Decoder können wir auf mindestens zwei Weisen realisieren, die erste ist in Abbildung 42 gezeigt.

Wir verwenden 2^8 AND Gattern mit n Eingängen. Wenn z.B. das AND-Gate dem Ausgang 5 gehört, sollte es logisch 1 für die binäre Zahl D(7:0) = 0000_1001 erzeugen.

Daraus folgt:

Y5 = !D7 & !D6 & !D5 & !D4 & D3 & !D2 & !D1 & D0

Alle Variablen, die null sind, werden negiert.

In solcher Realisierung brauchen wir 256 ANDs mit 8 Eingängen und 8 Invertern. Das sind insgesamt 256 × 18 + 8 × 2 ~ 4600 Transistoren.

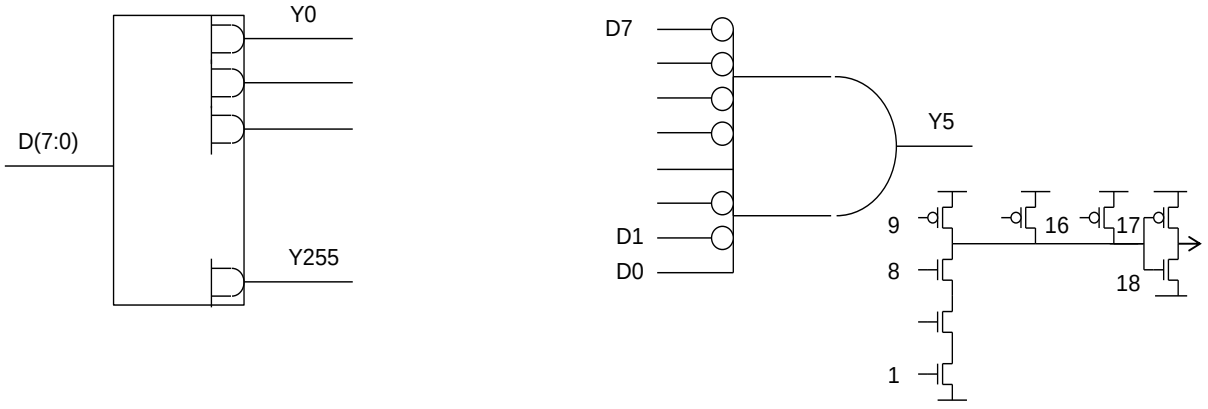


Abbildung 42: Decoder – erste Realisierung

Decoder kann als Teil eines Multiplexers verwendet werden, um die Gated-Invertern ein- und auszuschalten. Abbildung 43 zeigt den 256 -> 1 Multiplexer.

Für einen solchen Multiplexer werden etwa 6100 Transistoren benötigt.

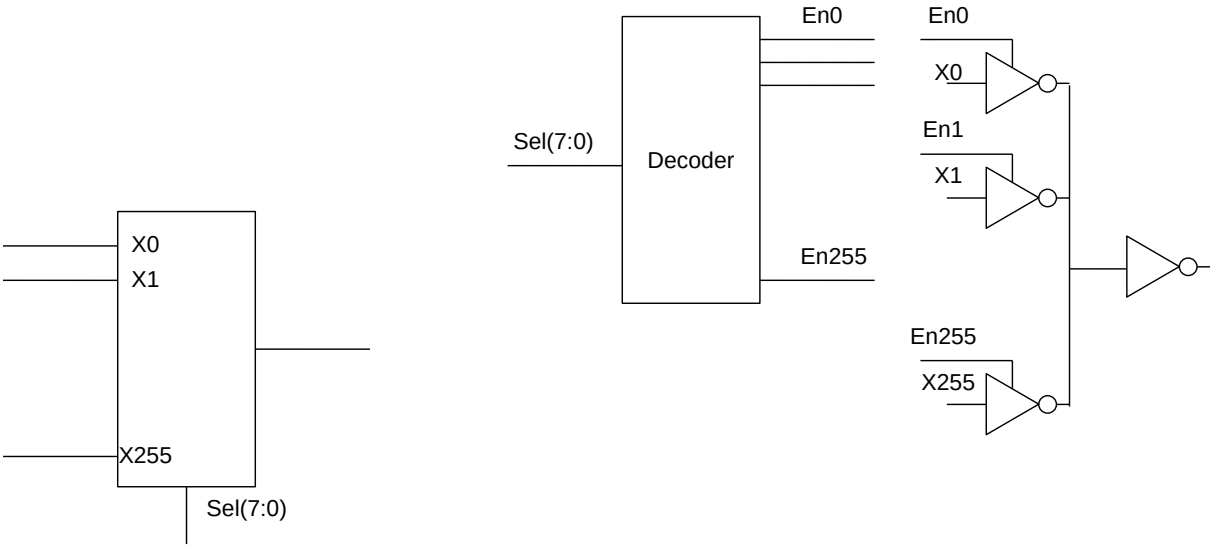


Abbildung 43: Multiplexer

Demultiplexer

Abbildung 44 zeigt einen Analogmultiplexer. Hier werden statt Gated-Invertern, die Schaltern (Transmission-Gates) benutzt. Ein Analogmultiplexer kann verschiedene analoge Signale über eine Leitung übertragen. Im Unterschied zum Digitalmultiplexer, ist der analoge Multiplexer bidirektional. Er kann also die Signale in beide Richtungen leiten. Das wird in Abbildung 44 gezeigt. Wenn man in einem Analogmultiplexer die Eingänge und Ausgänge „vertauscht“ bekommt man einen analogen Demultiplexer (Abbildung 45).

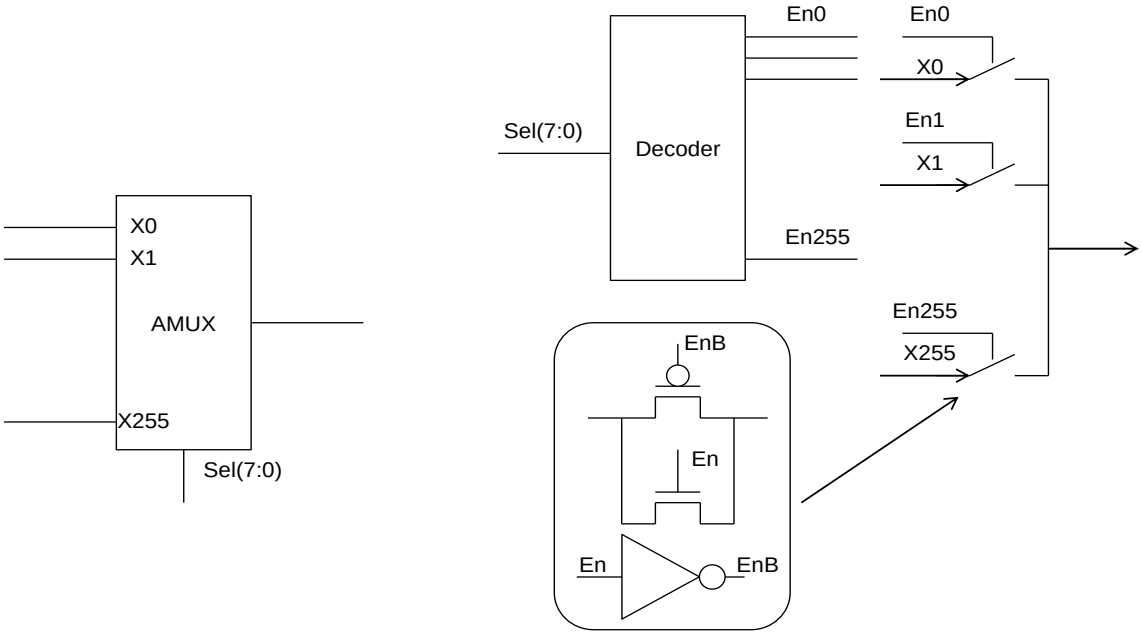


Abbildung 44: Analogmultiplexer

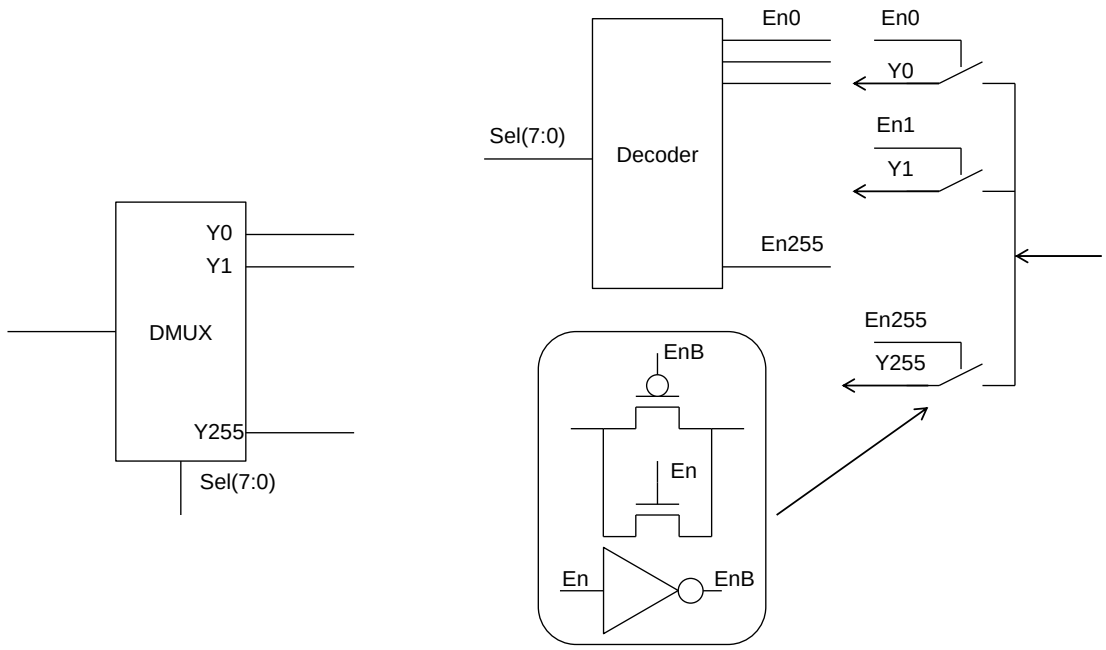


Abbildung 45: Demultiplexer - Prinzipschaltplan

Abbildung 46 zeigt den Aufbau des digitalen Demultiplexers mit 256 Ausgängen. Diese Schaltung soll ein Eingangssignal an einen von vielen Ausgängen verteilen. Die unbenutzten Ausgänge sollen null sein.

Für die Realisierung dieser Schaltung werden etwa 5136 Transistoren gebraucht. Zuerst brauchen wir 256 9 Fach ANDs mit jeweils 20 Transistoren. Wir brauchen auch, für jenen Select Eingang, einen Inverter (2 Transistoren) um die Negation zu erzeugen.

Bemerken wir, dass man aus einem Demultiplexer einen Decoder bekommt, wenn man Eingangssignal an logische Eins anschließt.

Abbildung 46: Demultiplexer mit AND-Gates

Zusammenfassung

Ein n-Fach Demultiplexer verteilt ein Digitalsignal an 2^n Ausgänge (Abbildung 47 - Mitte). Wenn man den Eingang des Demultiplexers an logisch 1 anschließt, bekommt man einen Decoder (Abbildung 47 - rechts). Den Decoder kann man als Teils eines n -> 2^n Multiplexers verwenden (Abbildung 47 - Links).

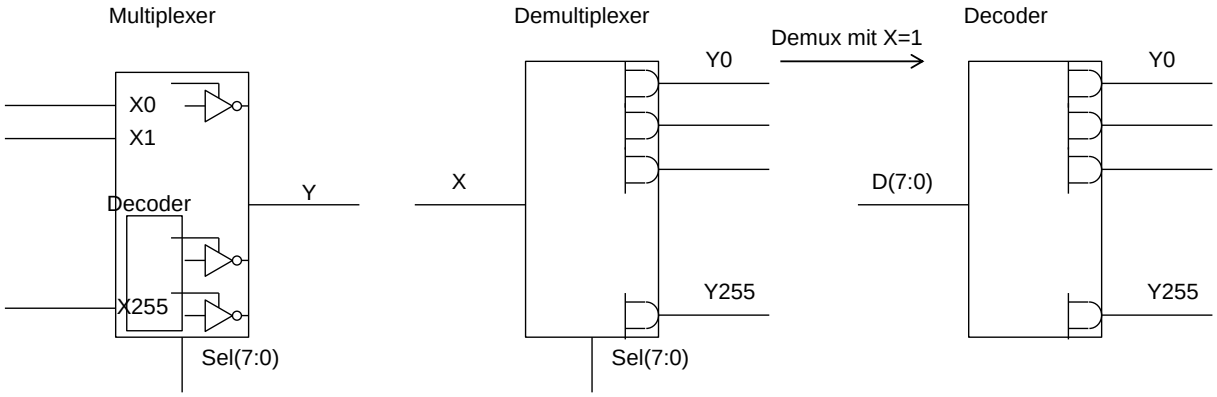


Abbildung 47: Multiplexer, Demultiplexer, Decoder

Ein Multiplexer kann auch mit weniger Transistoren realisiert werden, wenn eine Baumstruktur verwendet wird (Abbildung 48). In jedem Knoten verwenden wir jeweils einen (2->1) Multiplexer. Der Select-Eingang von allen Multiplexern in der ersten Stufe wird an Sel0 angeschlossen, von der zweiten an Sel1, usw. Diese Schaltung ist etwas langsamer als die Standardschaltung, benötigt aber um Faktor 3 weniger Transistoren.

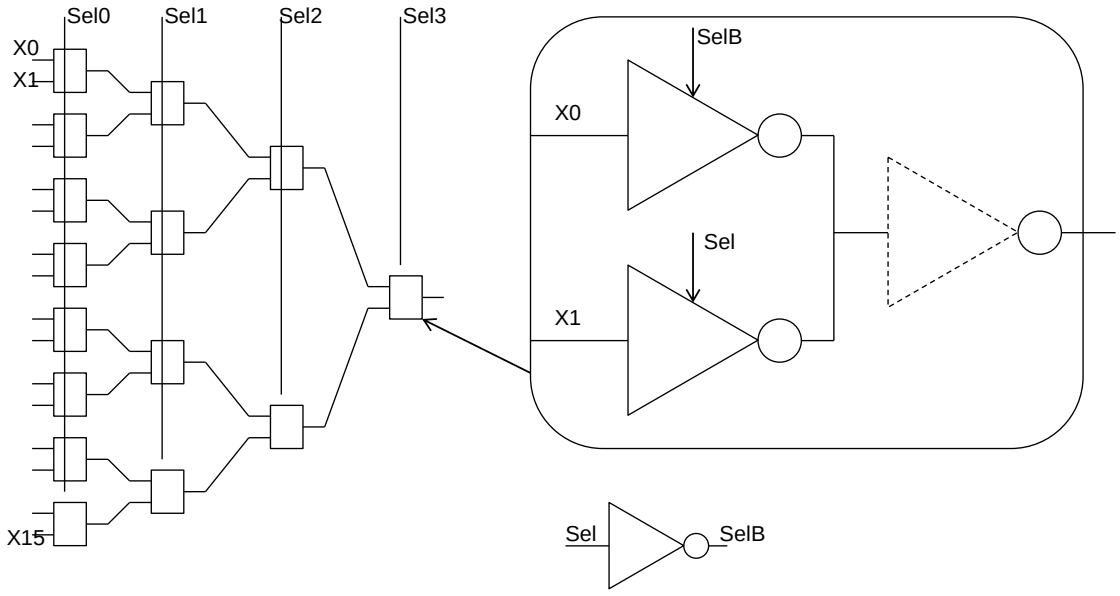


Abbildung 48: MUX Baum

Auch ein Demultiplexer (und Decoder) können als Baumstruktur realisiert werden (Abbildung 49).

In jedem Knoten verwenden wir jeweils einen (1->2) Demultiplexer. Der Select Eingang der ersten Stufe wird an Sel3 angeschlossen, der nächsten an Sel2 usw. Die Zahl von Transistoren im Fall eines 1->256 Demultiplexers ist:

(128 + 64 + … + 1) × 12 (2 × AND in jedem 1->2 DEMUX) + 8 × 2T (Negation für jeden Select Eingang) ~ 3076 Transistoren.

Trick: Man kann statt AND-Gattern in den Demultiplexern, stufenweise abwechselnd NANDs und NORs verwenden, um die Schaltung weiter zu vereinfachen.

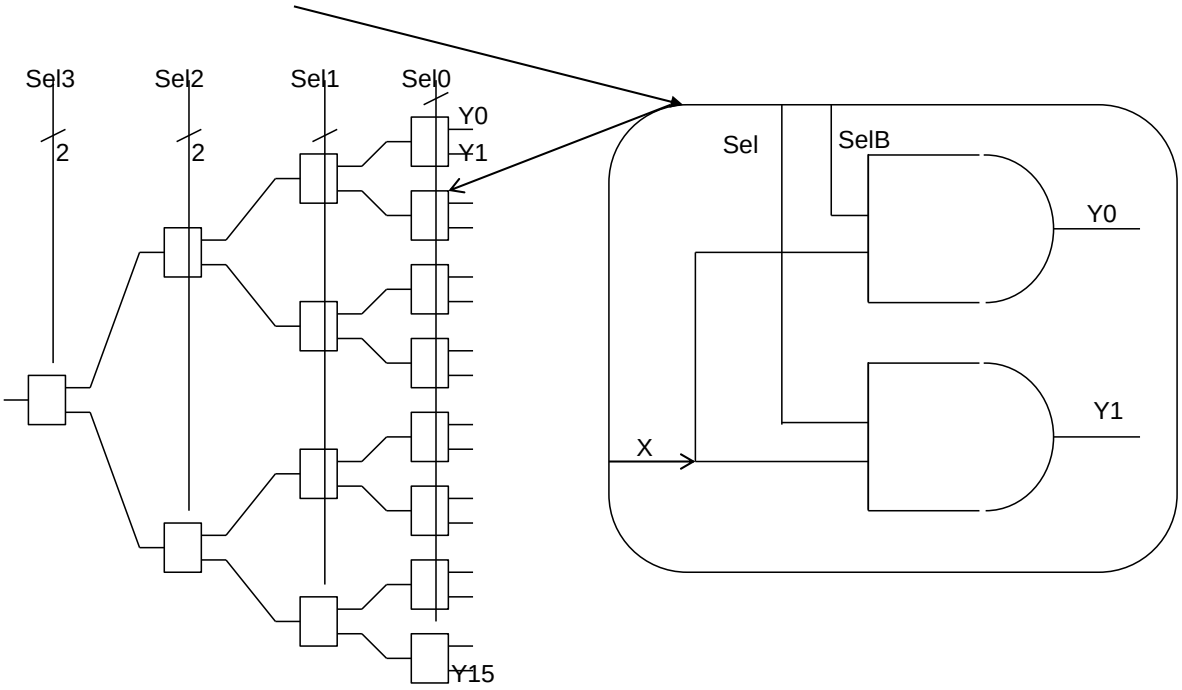


Abbildung 49: Demultiplexer - Baum