

Principy digitální techniky, logické obvody a kombinační logika

Petr Jedlička, Cyril Klimeš, Tomáš Pitner

Předmluva

HTML · Slides · PDF | Přehled předmětu

Cíle

- Poznat základní **logické obvody**
- Poznat principy kombinační logiky a umět je ve schématu propojit ve složitější logické funkce
- Poznat a umět použít logická hradla: *AND, OR, NOT, NAND, NOR*
- Poznat a umět použít *dekodéry* (demultiplexory), *multiplexory*
- Témata: Booleova algebra, Hradla, Sčítačka, Technologie CMOS, Multiplexer

Kombinační logické obvody

🔑 **Kombinační logické obvody**: jejich stav na výstupech závisí pouze na momentálním stavu na vstupech, tedy nemají vnitřní stav (paměť) — jsou to hradla jako AND, OR, NOT, NAND, NOR, multiplexory, dekodéry apod.

Principy

- Logické obvody fungují na principu booleovy algebry,
- dokážou vyhodnocovat výsledky jednoduchých či složitějších *booleovských výrazů* (**hradla a kombinace hradel**),
- vyhodnocovat výsledky *aritmetických výrazů*, když čísla reprezentujeme ve dvojkové soustavě (**aritmeticko-logické jednotky**),
- dokážou si údaje *zapamatovat* (**registry, paměti**),
- v případě **procesorů či kontrolérů** dokážou opakovaně provádět program zapsaný speciální sekvencí čísel - *instrukcí*.
- Podíváme se nejprve na jejich teoretický fundament: **dvouhodnotovou Booleovu algebru**.

Booleova algebra

🔑 **Booleova algebra** je počítání s jedničkami a nulami; tak počítá drtivá většina počítačů, které jsou na bázi *logických obvodů*.

- Základ digitální výpočetní techniky
- Čísla, následně znaky, obraz… jsou reprezentovány pomocí dvojkových číslic `0` a `1`.
- místo `0` a `1` můžeme říkat `true` a `false` (pravda, nepravda)

Booleovské operace

- operace nad logickými hodnotami `0`, `1`, kdy výsledkem je zase `0` nebo `1`
- definováno nad jednotlivými booleovskými hodnotami (bity), ale lze rozšířit na n-tice bitů
- běžně používanými booleovskými operacemi jsou:
- **AND**, **OR**, **NOT**, pak **XOR** a dále jejich kombinace **NAND**, **NOR**

Operace AND, OR, NOT, XOR

Mějme dvě booleovské hodnoty X_1 a X_2, pak jejich:

AND

logický **součin** (**konjunkce**), značíme X_1 `AND` X_2 nebo `AND(`X_1`,` X_2`)` nebo $X_1 \wedge X_2$

OR

logický **součet** (**disjunkce**), značíme X_1 `OR` X_2 nebo `OR(`X_1`,` X_2`)` nebo $X_1 \vee X_2$

NOT

logická **negace**, značíme `NOT(`X_1`)` nebo $\neg X_1$ nebo i `-`X_1

XOR

logický **výlučný součet**, značíme X_1 `XOR` X_2 nebo `XOR(`X_1`,` X_2`)` nebo $X_1 \veebar X_2$



Lze definovat i pro více vstupů, například *třívstupový AND*: `AND(`X_1`,` X_2`,` X_3`)=1` právě když všechny vstupy X_n`=1`

Pravdivostní tabulka AND, OR, NOT, XOR

A	B	A AND B	A OR B	NOT A	A XOR B
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Odvozené operace

- Negací operace `XOR` získáme operaci ekvivalence, rovnosti (`EQ`)
- Negací operace `AND` získáme operaci `NAND`
- Negací operace `OR` získáme operaci `NOR`

Pravdivostní tabulka EQ, NAND, NOR

A	B	A EQ B	A NAND B	A NOR B
0	0	1	1	1
0	1	0	1	0
1	0	0	1	0
1	1	1	0	0

Booleovské výrazy

Booleovský výraz může být:*

- Primitivní hodnota (=*term*) `0`, `1` (`true`, `false`)
- Proměnné nabývající booleovských hodnot, značíme obvykle X_n, Y_n, `A`...
- Výše uvedené operace `AND`, `OR`, `NOT`, `XOR`, ...
- Závorky obsahující booleovský výraz `(` booleovský výraz `)`



Nejsou-li použity závorky, obvykle bereme, že konjunkce (AND) má přednost před disjunkcí (OR) podobně jako má násobení přednost před sčítáním.

Příklad booleovského výrazu s proměnnými

Table 1. Pravdivostní tabulka pro `Y = (X1 AND X2) OR (X1 AND X3)`

X_1	X_2	X_3	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0

Table 2. Pravdivostní tabulka pro `Y = (X1 AND X2) OR (X1 AND X3)`

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Vyhodnocení výrazu

- Pro konkrétní přiřazení hodnot do vstupů X_1, X_2, X_3 spočteme postupně Y

1. mějme na vstupech X_1=1, X_2=0 a X_3=0
2. výraz `Y = (X1 AND X2) OR (X1 AND X3)` se vyhodnocuje "zevnitř ven" podobně jako třeba aritmetický `(A + B) * C`
3. vypočteme obsah první závorky `(X1 AND X2)` ⇒ `(1 AND 0)` ⇒ `0`
4. nahradíme obsah první závorky `Y = (0) OR (X1 AND X3)`
5. vypočteme obsah druhé závorky `(X1 AND X3)` ⇒ `(1 AND 0)` ⇒ `0`
6. nahradíme obsah druhé závorky `Y = (0) OR (0)` ⇒ `Y = 0`
7. výsledek `Y = 0`

Příklady booleovských výrazů

Table 3. Pravdivostní tabulka pro `Y = NOT(NOT(X1) AND NOT(X2))` ≡ `Z = X1 OR X2`

X_1	X_2	NOT(NOT(X_1) AND NOT(X_2))	X_1 OR X_2
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Ekvivalence výrazů

- Z logického hlediska jsou výrazy `Y` a `Z` ekvivalentní.
- `Z = X1 OR X2` je jednodušší, kratší, vyhodnotitelný v jednom kroku; obvodově realizovatelný jediným hradlem `OR`.
- Je již *minimalizovaný* na nejmenší počet logických operací.
- Existuje **obecný postup** minimalizace výrazu.

Zákony: Identitní, Nulový, Dvojitá negace

Neutrální prvek

- `A AND 1 = A`: jednička nemění výsledek konjunkce
- `A OR 0 = A`: nula nemění výsledek disjunkce

Absorpce (pohlcení)

- `A AND 0 = 0`: nula určuje výsledek konjunkce
- `A OR 1 = 1`: jednička určuje výsledek disjunkce

Dvojitá negace

`¬(¬(A)) = A`: negace negace dá původní hodnotu

Zákony: Komplementární, Komutativní, Asociativní

`A OR ¬A = 1`

vždy platí A nebo negace A (pravidlo vyloučeného třetího)

`A AND ¬A = 0`

ale nikdy obě současně

`A OR B = B OR A` **a** `A AND B = B AND A`

operace disjunkce a konjunkce jsou komunitativní, lze prohodit pořadí operandů

`A OR (B OR C) = (A OR B) OR C`

asociativita = u stejných operací nezáleží na závorkování; obdobně `A AND (B AND C) = (A AND B) AND C`

Zákony: Distributivní, Absorpční, De Morganův

`A AND (B OR C) = (A AND B) OR (A AND C)`, **rovněž** `A OR (B AND C) = (A OR B) AND (A OR C)`

"roznásobení" závorky, funguje pro obě operace `AND` i `OR`!

`A OR (A AND B) = A` **a také** `A AND (A OR B) = A`

`A` rozhoduje o výsledku, tj. "pohltí" `B`

`¬(A OR B) = ¬A AND ¬B`, `¬(A AND B) = ¬A OR ¬B`

de Morganovy zákony (negace operace je totéž jako "opačná" operace nad negovanými vstupy)



Další pravidla už jsou aplikací předchozích, *so go ahead and use 'em* :)

Zjednodušování booleovských výrazů

- Zjednodušení booleovského výrazu nad proměnnými X_1 .. X_n vede k **ekvivalentní logické funkci** se stejnými pravdivostními hodnotami,
- ale v podobě, která je **kratší a efektivnější** (méně operací, méně proměnných, použití jen daných funkcí, třeba NAND).
- V praxi to znamená *méně logických hradel* při realizaci v hardware.

Příklad zjednodušení výrazu

1. `F(A,B,C) = (A AND B) OR (A AND NOT(B)) OR (NOT(A) AND B)`
2. lze rozdělit na `(A AND B) OR (A AND NOT(B))` OR `(NOT(A) AND B)`, pro celkový výsledek 1 stačí jeden z nich 1
3. hodnota prvního výrazu `(A AND B) OR (A AND NOT(B))` je vždy `= A` (protože v obou `AND` závorkách vždy jedno `B` nebo `NOT(B)` platí a druhé neplatí, ale pro výsledek=1 musí být `A=1`), jestliže `A=1`, je tedy jistě `F=1`.
4. Když `A=0`, pak druhý výraz `(NOT(A) AND B)` = `(NOT(0) AND B)` = `(1 AND B)` = `B`. Když `A=0`, pak druhý výraz lze zjednodušit na `B`.
5. celkově tedy `F(A,B,C) = A OR B`

Příklad

Mějme booleovskou funkci: `F(A,B,C) = (A AND B) OR (A AND NOT(B)) OR (NOT(A) AND B)`

Table 4. Krok 1: Karnaughova mapa

A\BC	BC=00	BC=01	BC=10	BC=11
A=0	0	0	1	1
A=1	1	1	1	1

Krok 2: Seskupování, kde jsou 1

- Řádek `A=0`, skupina (`B=1`, libovolné `C`) → výsledek `=B`
- Řádek `A=1` (libovolné `B` i `C`) → dává `A`

Řešení

Výsledek

```
F(A,B,C) = A OR B
```

Popis

- Zjednodušená podoba `A OR B` je *ekvivalentní* původnímu výrazu, ale používá *méně operací*,

- už nepotřebuje negaci, jen jedno OR a dvě proměnné.

Online nástroje na minimalizaci

- https://www.dcode.fr/boolean-expressions-simplifier
- https://boolean-algebra.com/
- Nyní přejděme k technické realizaci Booleovy algebry: k **logickým obvodům**.

Logické obvody

🔑 **Logické obvody** umějí *počítat* v Booleově algebře a tedy i s čísly a texty a dalšími daty + ty složitější si něco *pamatují* a dají se *programovat*.

K čemu je logický obvod?

- Logické obvody jsou technickou realizací teoretického konceptu dvouhodnotové Booleovy algebry.
- Dnes téměř výhradně na bázi polovodičových spínacích prvků - různých typů tranzistorů.
- Logické hodnoty jsou obvykle reprezentovány napěťovými úrovněmi.
- V případě komunikačních rozhraní se často místo napěťových úrovní používá na rozlišení 0 a 1 směr toku proudu nebo jiné metody.

Logické úrovně

Logická hodnota	Pozitivní logika (active high)	Negativní logika (active low)
0	L (nízká úroveň)	H (vysoká úroveň)
1	H (vysoká úroveň)	L (nízká úroveň)

Napěťové úrovně

- Doposud jsme rozlišovali "nižší" a "vyšší" napěťové úrovně. *Kolik voltů to je?*
- Záleží na použité technologii:

  TTL

  0..0.8 V =nízká úroveň (L), 2..5 V =vysoká úroveň (H), mezi tím zakázané pásmo 0.8..2 V

  CMOS

  různě - některé jsou TTL-kompatibilní, jiné pracují na celkově nižších nebo vyšších napěťových úrovních než TTL

- Obecně nahrubo platí, že pro logické obvody čím nižší napětí dané technologii stačí, tím lépe: nižší spotřeba.

Třístavový výstup (tri-state output)

- Doposud jsme měli za to, že na výstupu obvodu je jen `0` nebo `1`.
- Třístavový výstup přidává třetí stav: **vysoká impedance** (`Z`, high impedance)
- Výstup ve stavu `Z` se chová efektivně jako *odpojený*, jako by vůbec nebyl.
- Jde o fundamentální techniku umožňující zapojit více výstupů na společný vodič — obvykle sběrnici.

Stavy logického obvodu

Stav obvodu je kombinací:

- Hodnot na *vstupech* obvodu
- Hodnot na *výstupech* obvodu
- *Vnitřního stavu*, který neexistuje u *kombinační logiky*, viz Kombinační logika

Kombinační obvody

- také *logika nezávislá na času* zahrnuje obvody, jejich výstupní stav (stav na výstupech) závisí pouze na momentálním stavu na vstupech
- například *logická negace* vstup *X*, výstup *Y*:
  - `X=0` ⇒ `Y=1`
  - `X=1` ⇒ `Y=0`
- na ničem jiném než na *X* výstup *Y* nezávisí
- rovněž je úplně jedno, jakou hodnotu měl v přechozí době



Kombinační logické obvody tedy nemají paměť.

Hradla

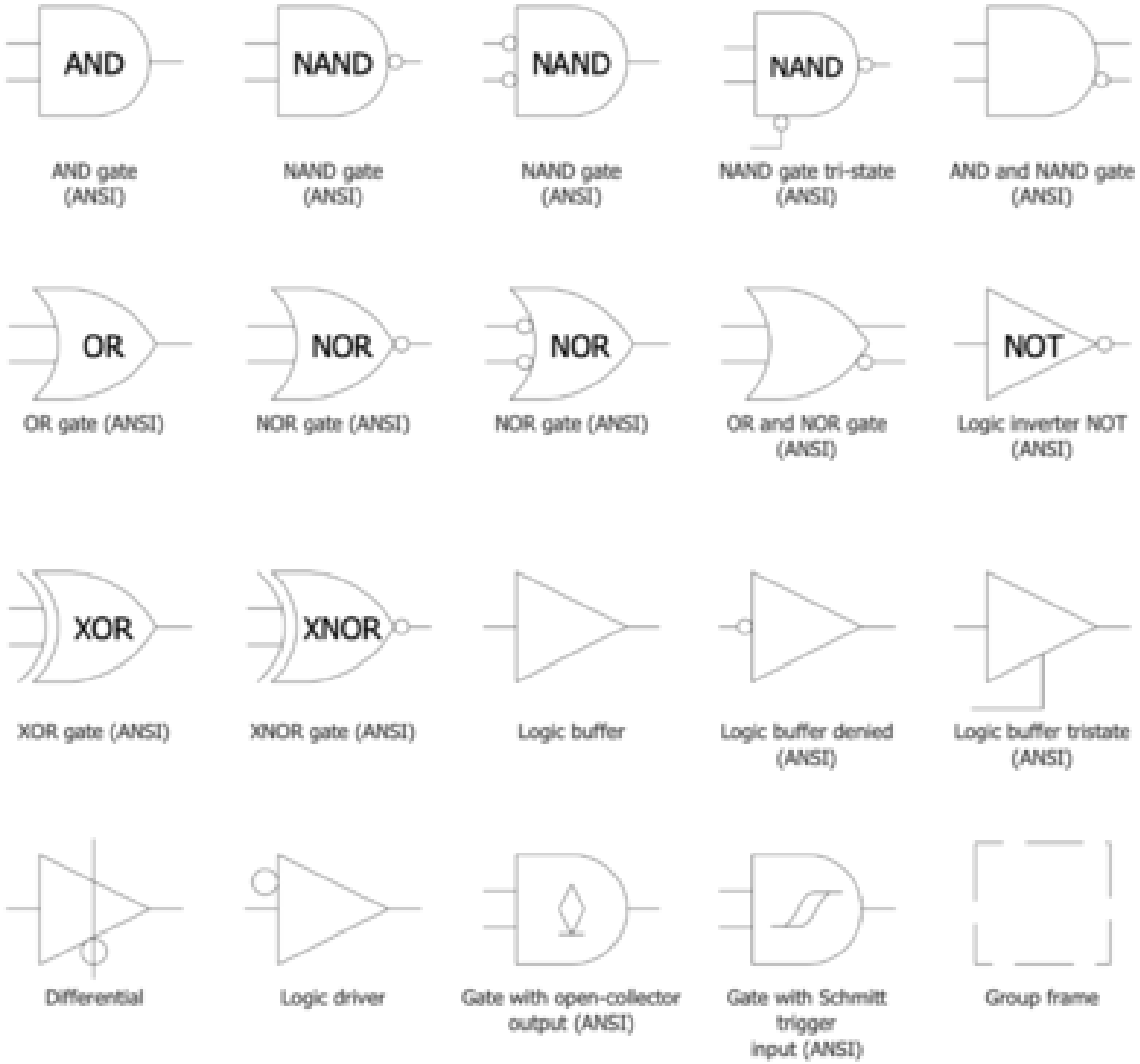
- Hradla (gates) jsou jednoduché *logické obvody* s jednotkami až desítkami tranzistorů, kde stačí nízký stupeň integrace.
- Realizují *základní logické (booleovské) operace*:
- AND,OR,NOT,
- ale často i `NAND` ≡ NOT(AND),`NOR` ≡ NOT(OR),`XOR` a`XNOR`.

Technické provedení

- Často je více hradel v jednom pouzdře, třeba 4 NAND hradla.
- Každé hradlo může mít 1 (NOT) nebo 2, případně více vstupů — u AND, OR, NAND, NOR.
- Někdy bývá na jednom čipu, tedy i v jednom pouzdře, i kombinace vícero typů hradel zapojených třeba jako "AND-OR-INVERT".

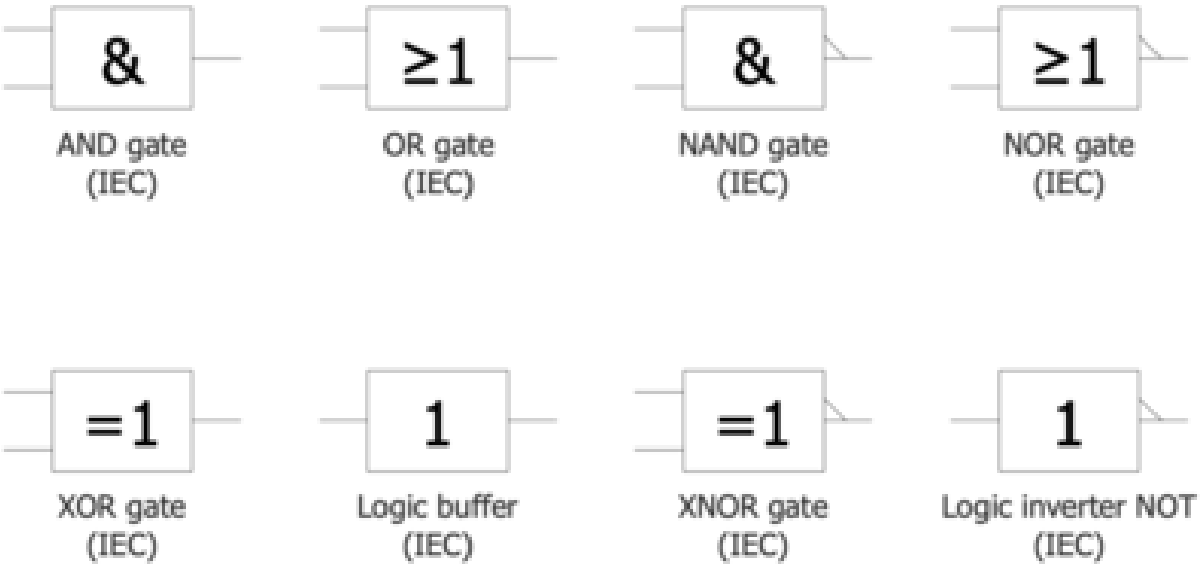
Standardy schematického značení

- Pro schematické znázornění hradel se používají symboly, které jsou standardizované, ale standardů je více, v ČR se používají nejméně dva:
- ČSN/EN/IEC 60617
- ANSI/IEEE Std 91-1984

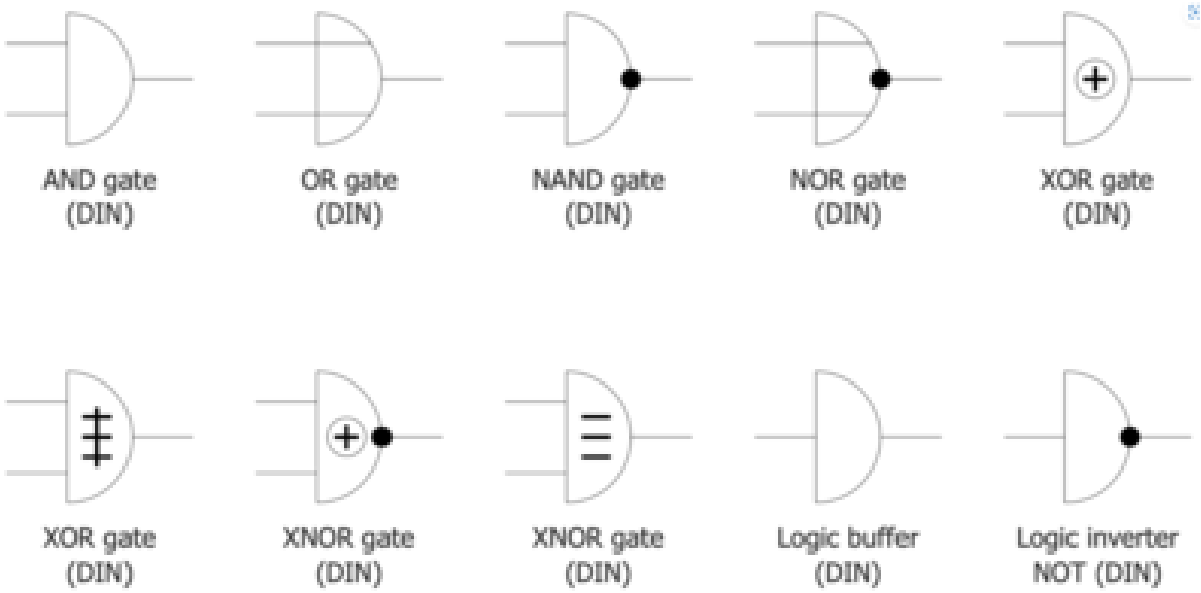


Obr. 1. ANSI = US standard, nejpoužívanější v SW nástrojích

IEC a DIN



Obr. 2. IEC = Světově nejuniverzálnější, jednoznačné

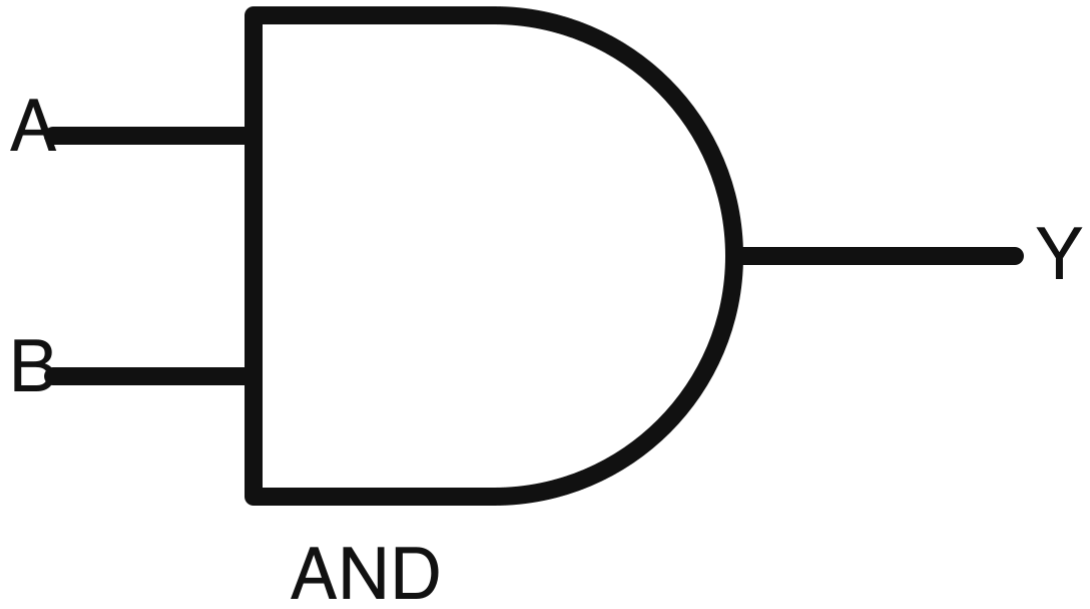


Obr. 3. DIN = Používané u nás ojediněle, v německých schématech

AND

- Hradlo **AND** dává na výstupu logickou 1 jen tehdy, když jsou všechny jeho vstupy v logické 1.
- U dvouvstupové varianty tedy platí: `Y = A AND B`.

Schematická značka AND

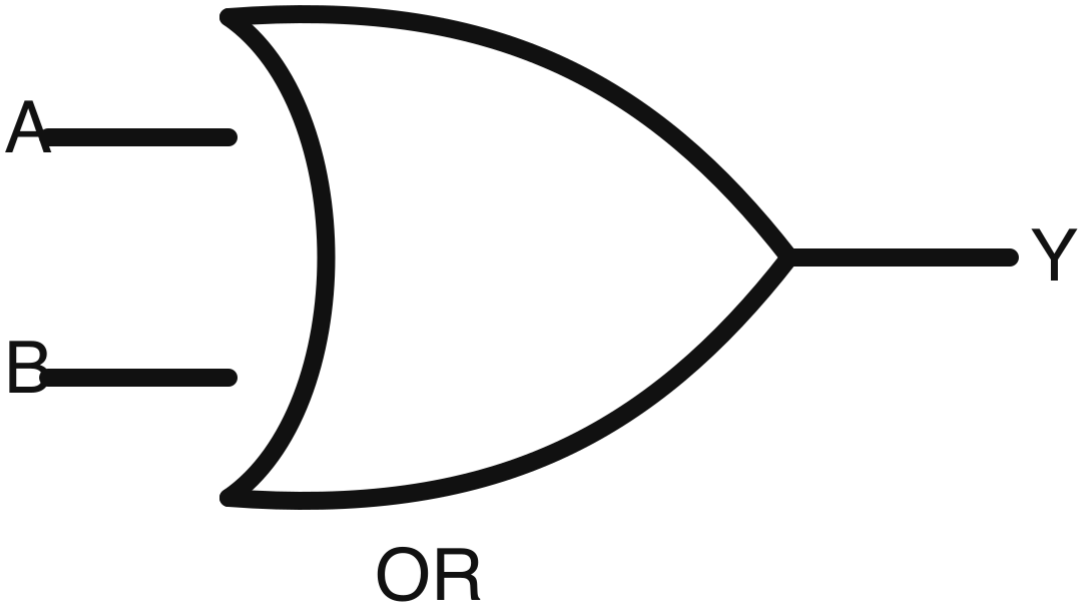


OR

- Hradlo **OR** dává na výstupu logickou 1, když je v logické 1 alespoň jeden vstup.

- U dvouvstupové varianty tedy platí: `Y = A OR B`.

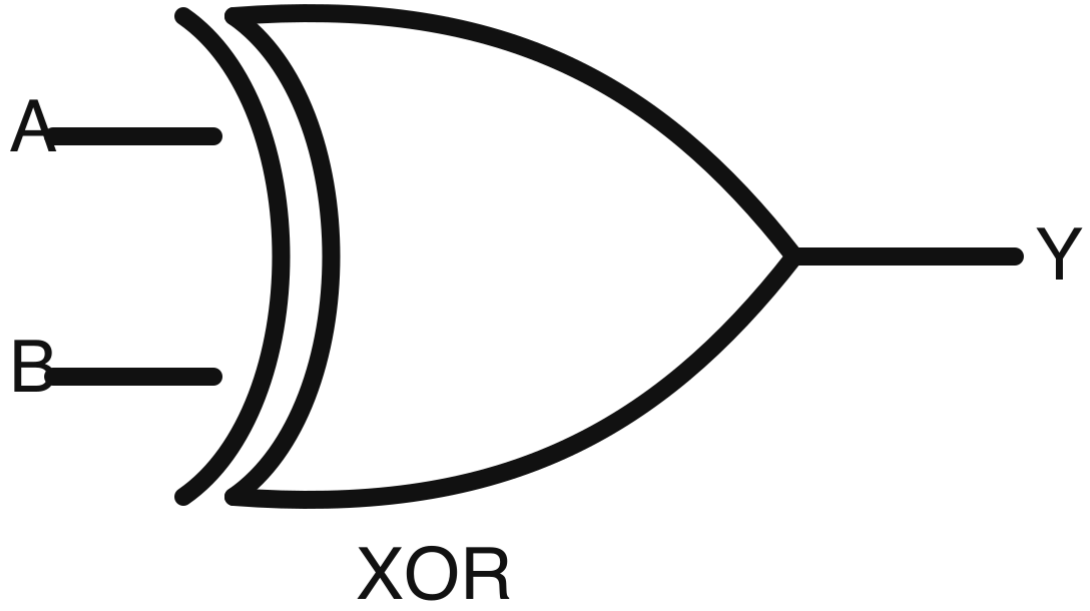
Schematická značka OR



XOR

- Hradlo **XOR** dává na výstupu logickou 1 právě tehdy, když se vstupy liší.
- U dvou vstupů tedy platí: `Y = 1`, pokud `A != B`.

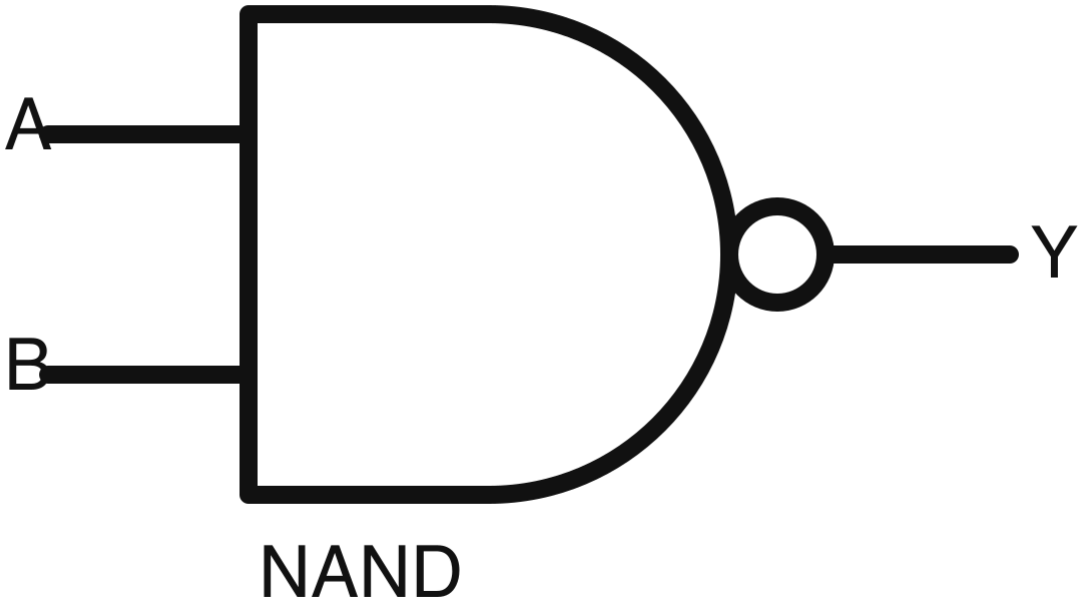
Schematická značka XOR



NAND

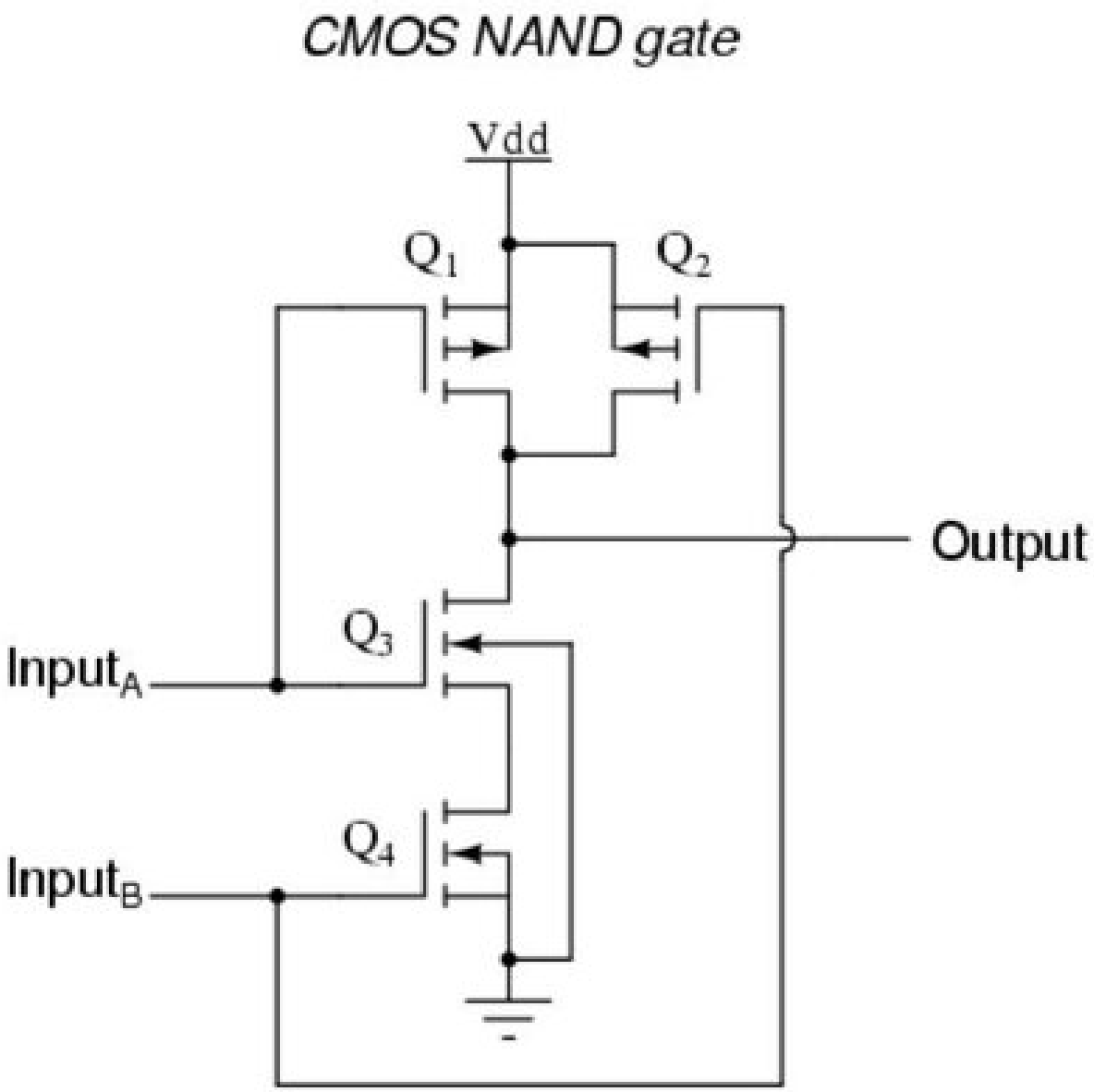
- Obvodová realizace booleovské operace `NAND` (NOT AND) nad nejméně dvěma vstupy.
- NAND(X,Y) = NOT(AND(X,Y)) = NOT(X AND Y) …
- Zjednodušeně řečeno:
  - na výstupu NAND hradla je *logická 1*, pokud *alespoň jeden z jeho vstupů je logická 0*,
  - v opačném případě, pokud jsou oba vstupy logické 1, je na výstupu logická 0.

Schematická značka NAND



NAND v CMOS

- Nejčastěji se NAND hradla realizují čtveřicí MOSFET tranzistorů opačných polarit
- dva jsou NMOS, dva PMOS
- To dovoluje optimální, tzn. zcela minimální, klidovou spotřebu: obvodem neteče v ustáleném stavu L nebo H proud.

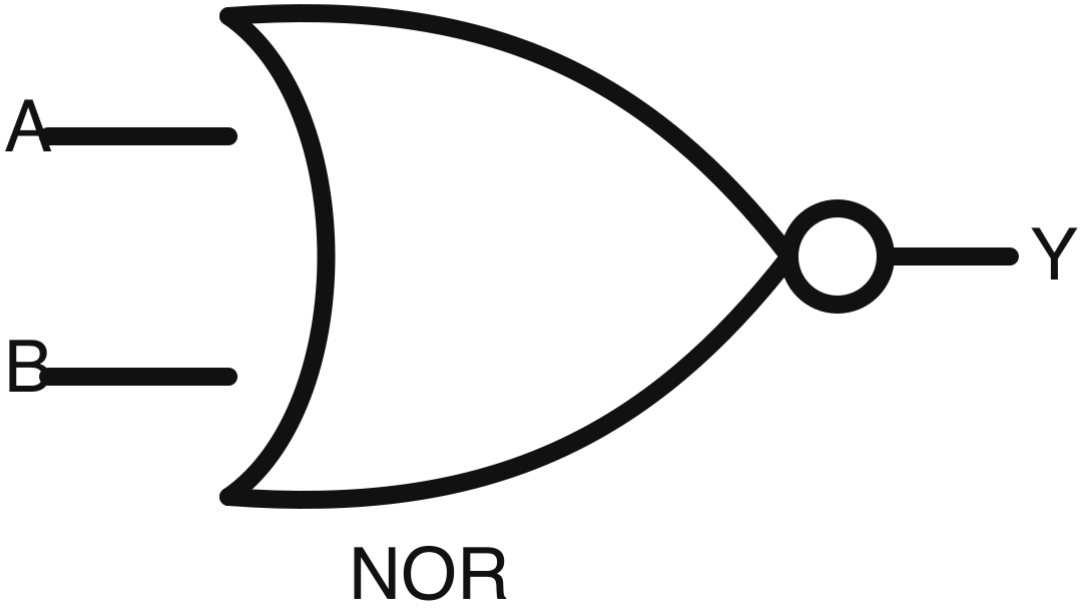


Obr. 4. NAND v provedení CMOS

NOR

- Hradlo **NOR** je negací hradla **OR**.
- U dvou vstupů tedy platí: `Y = NOT(A OR B)`.

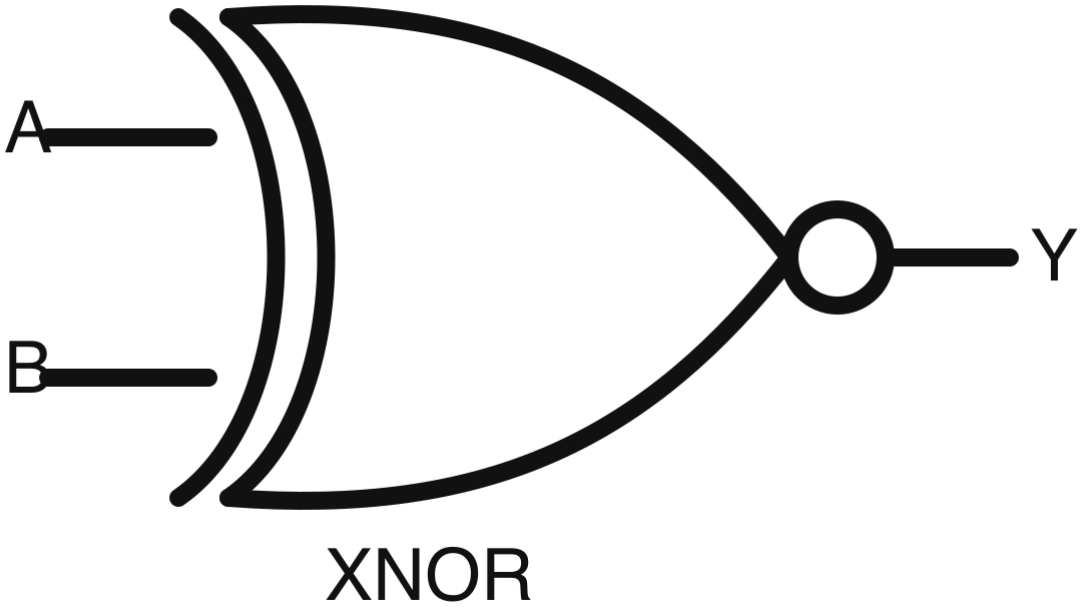
Schematická značka NOR



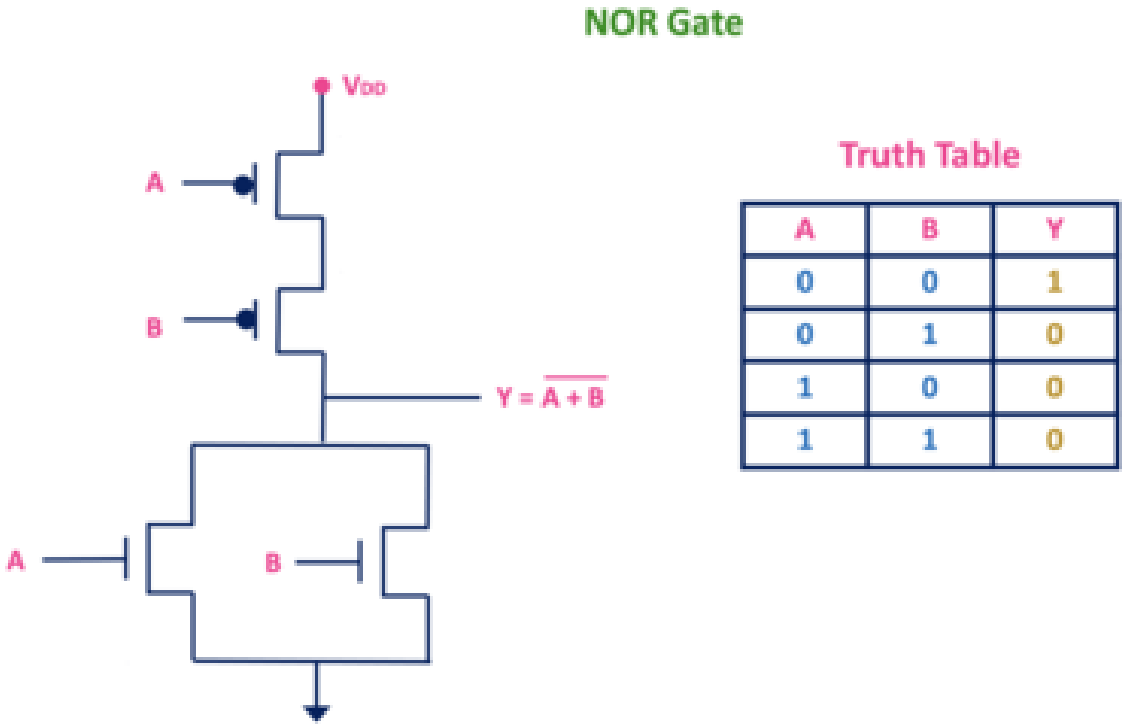
XNOR

- Hradlo **XNOR** je negací hradla **XOR**.
- Dává na výstupu logickou 1 právě tehdy, když jsou vstupy stejné.

Schematická značka XNOR



- Nejčastěji se NOR hradla rovněž realizují čtveřicí MOSFET tranzistorů opačných polarit
- dva jsou NMOS, dva PMOS
- To dovoluje optimální, tzn. zcela minimální, klidovou spotřebu: obvodem neteče v ustáleném stavu L nebo H proud.

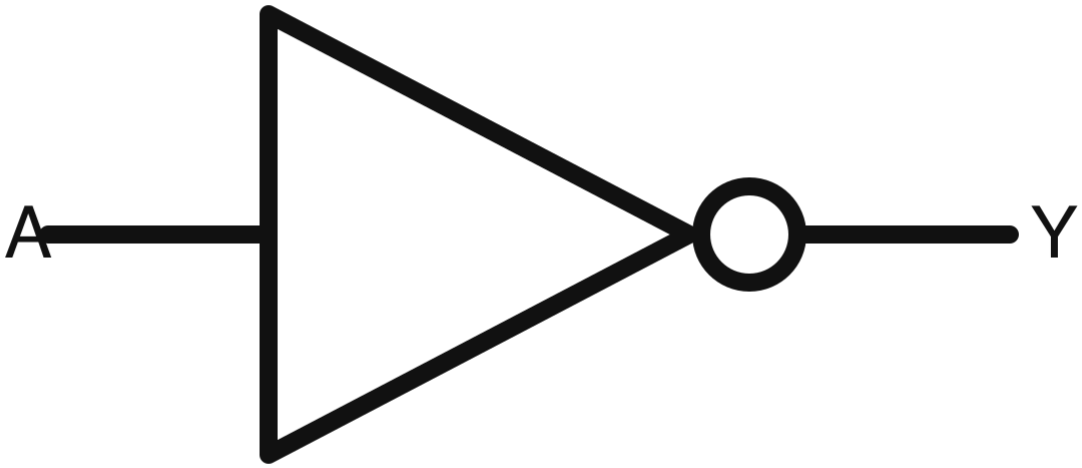


Obr. 5. NOR v provedení CMOS

NOT (Invertor)

- Z povahy fungování logických prvků na bázi tranzistorů je hradlo NOT (invertor) nejjednodušší, s minimem součástek.
- Nejčastěji se NOT hradla realizují dvojící MOSFET tranzistorů opačných polarit — po jednom NMOS a PMOS

Schematická značka NOT



NOT

Sčítačka

- **Sčítačka** (adder) je typickým příkladem (stále ještě) jednoduchého kombinačního obvodu.
- Úkolem je *sečíst* v nejjednodušším případě dva bity (tzn. dvě jednobitová čísla) a dostat jeden bit jako výsledek.

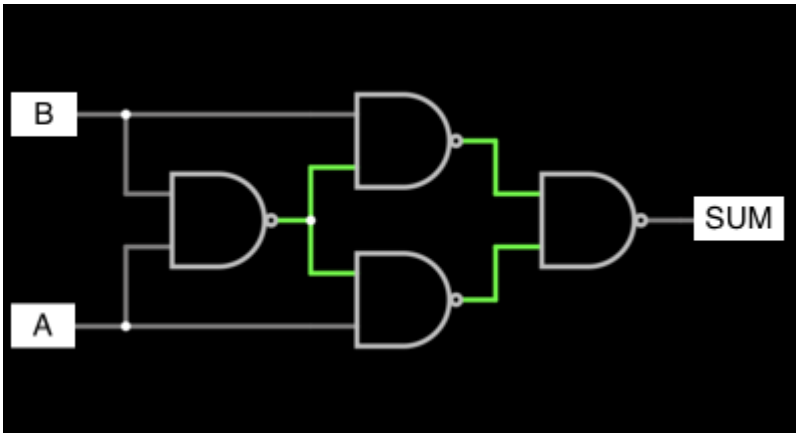
Reálně používané sčítačky

- V reálu je potřeba:
  - **sčítat delší čísla** (vícebitová) než 1-bitová (např. `01 + 10 ⇒ 11`)
  - kromě bitového výsledku zachytit a poslat ven i případné **přetečení**, tzn. situaci, kdy výsledek překročí bitový rozsah výstupu (`1 + 1 ⇒ 0`, ale přetečení `1`)
  - do sčítání ke vstupům **přičítat i přenos** (tj. přetečení) z nižších řádů, z předchozích sčítaček: `1` (přetečení z přechozího stupně) `1 + 1 + 0 ⇒ 0` a přetečení `1`

Nejjednodušší sčítačka bez přenosu

- Exclusive-OR jako úplně primitivní sčítačka
- sečte dva bity,
- ale neindikuje přetečení

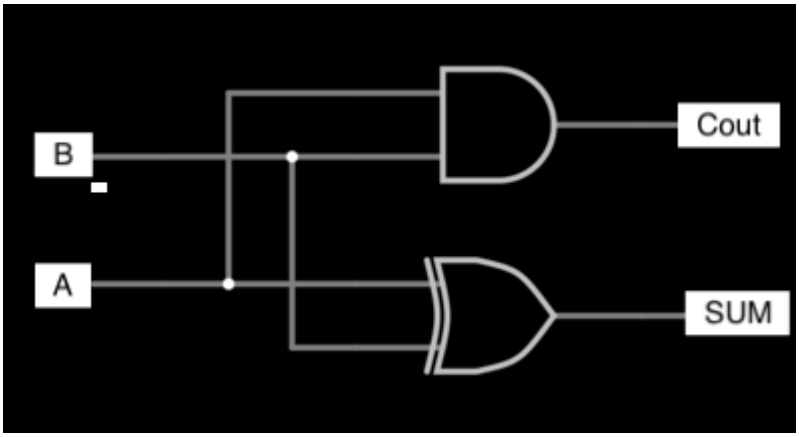
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



Polosčítačka (half-adder)

- Simulace jednobitové polosčítačky:
  - na vstupu jsou dvě jednobitová čísla

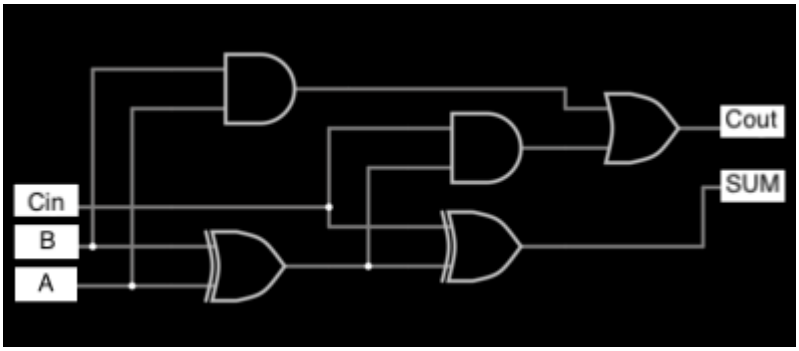
- produkuje jeden bit součtu + přetečení



A	B	SUM (A XOR B)	C_{out} (A AND B)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Úplná sčítačka (full-adder)

- Simulace jednobitové úplné sčítačky:
  - na vstupu jsou dvě jednobitová čísla
  - produkuje jeden bit součtu + přetečení



Úplná sčítačka (full-adder)

A	B	C_{in}	SUM	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0

A	B	C_{in}	SUM	C_{out}
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Multiplexer

- **Multiplexer (MUX)** je *kombinační logický obvod*, který z několika vstupů (např. 2, 4, 8…) vybírá jeden jediný a ten přeposílá na výstup.
- Který vstup bude zvolen, určují *řídicí (adresové) vstupy*.
- Též se jim říká *selektory dat*, protože vybírají vstup, který se posílá na výstup.
- Základní označení: *n:1* MUX znamená, že má *n* datových vstupů a 1 výstup.
- Počet adresových vstupů je $\log_2(n)$.
- Multiplexor může data propouštět přímo nebo invertovaná

Příklad: DM74LS157

- DM74LS157 je výrobek spol. National Semiconductor Corporation.
- *Quad 2-Line to 1-Line Data Selectors/Multiplexers*
- Z názvu plyne, že vybírá *jeden ze dvou 4bitových vstupů* a propojuje je na výstup.
- Existuje také varianta s invertovaným výstupem *DM74LS158*.

Inputs				Output Y	
Strobe	Select	A	B	LS157	LS158
H	X	X	X	L	H
L	L	L	X	L	H
L	L	H	X	H	L
L	H	X	L	L	H
L	H	X	H	H	L

H = High Level, L = Low Level, X = Don't Care

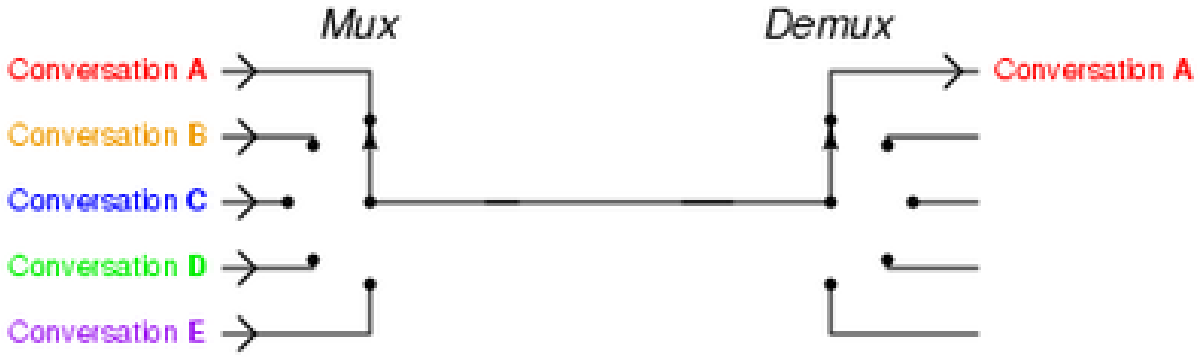
Obr. 6. Funkční tabulka

Demultiplexer

- **Demultiplexer (DEMUX)** je *kombinační logický obvod*, který z jednoho vstupu (byť vícebitového) posílá data na jeden z výstupů vybraný z několika (např. 2, 4, 8…).
- Který výstup bude zvolen, určují *řídicí (adresové) vstupy*.
- Též se jim říká *dekodér*, protože dekóduje adresu na *1 z n* výstupů, kde $n = 2^a$ (a = počet adresových bitů) .
- Základní označení: *1:n* MUX znamená, že má 1 vstup a *n* datových výstupů.
- Počet adresových vstupů je $\log_2(n)$.

- Demultiplexer je "protikus" multiplexoru.

Systém: Multiplexer a demultiplexer



Obr. 7. Nejdříve multiplexujeme (sloučíme), pak demultiplexujeme (rozdělíme)

SN74LS138

- SN74LS138
- 3 adresové bity vybírá jeden z 8 výstupů
- 3-LINE TO 8-LINE DECODER/DEMUX
- Má navíc tři povolovací vstupy G1, G2A, G2B
- ⇒ dokud nejsou aktivní, není aktivní žádný výstup

Pravdivostní tabulka

A2	A1	A0	G1	G2A	G2B	Y0	Y1	…	Y7	pozn.
X	X	X	0	X	X	1	1	…	1	nepovolený výstup kvůli `G1`
X	X	X	X	1	X	1	1	…	1	nepovolený výstup kvůli `G2A`
X	X	X	X	X	1	1	1	…	1	nepovolený výstup kvůli `G2B`
0	0	0	1	0	0	0	1	…	1	vybrán výstup `Y0`
0	0	1	1	0	0	1	0	…	1	vybrán výstup `Y1`
0	1	0	1	0	0	1	1	…	1	
0	1	1	1	0	0	1	1	…	1	
1	0	0	1	0	0	1	1	…	1	
1	0	1	1	0	0	1	1	…	1	
1	1	0	1	0	0	1	1	…	1	
1	1	1	1	0	0	1	1	…	0	vybrán výstup `Y7`

`X` znamená „don’t care“ (libovolná hodnota). Výstupy jsou *active-low*.

Užití MUX/DEMUX

- Široké užití: přenos více signálů po jednom vodiči
- Sloučení n vstupů do jednoho ⇒ přenos ⇒ rozdělení zase na n signálů
- Typická aplikace: v autě jsou takto sloučeny kabely (nejen) k osvětlení.
- Šetří se za kabeláž.

Shefferova algebra

- Používá na všechno jen hradla **NAND**
- Vhodné pro návrh složitých obvodů na čipu, protože hradla NAND se dobře vyrábí technologií **CMOS** (*Complementary Metal-Oxide-Semiconductor*)

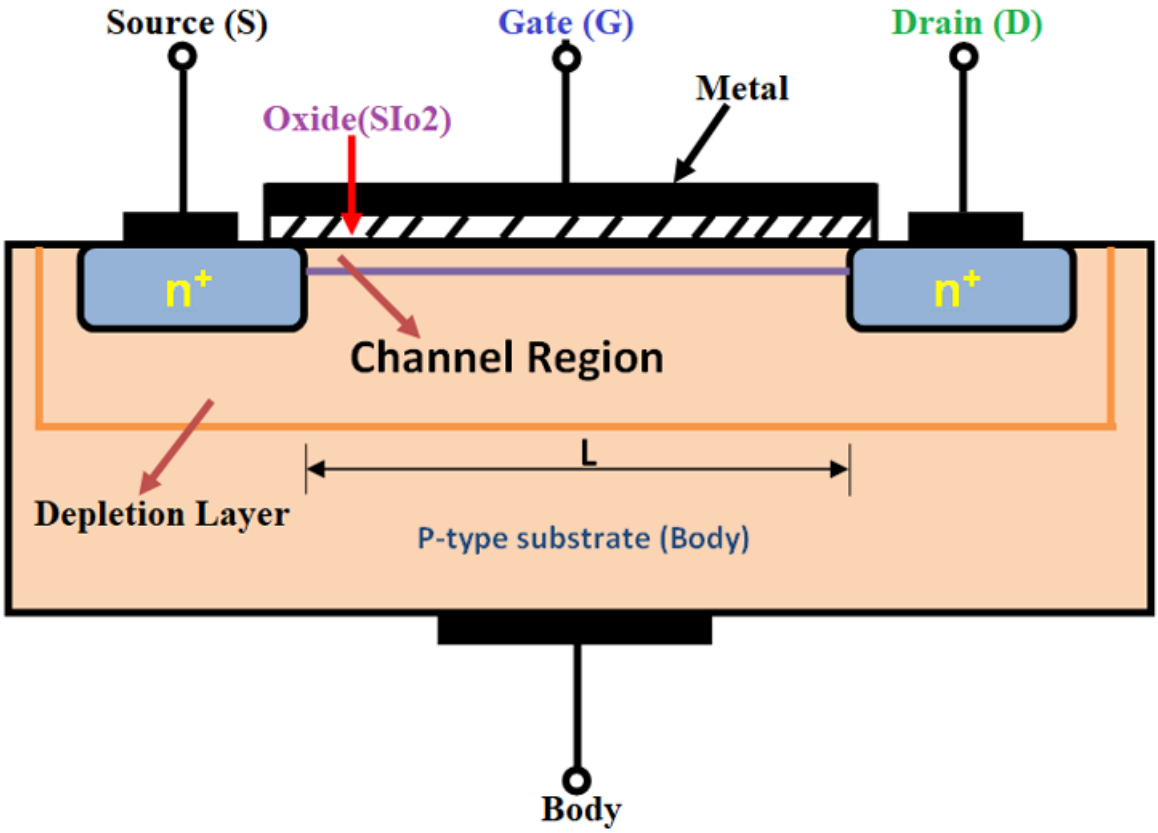
Peirceova algebra

- Používá na všechno jen hradla **NOR**

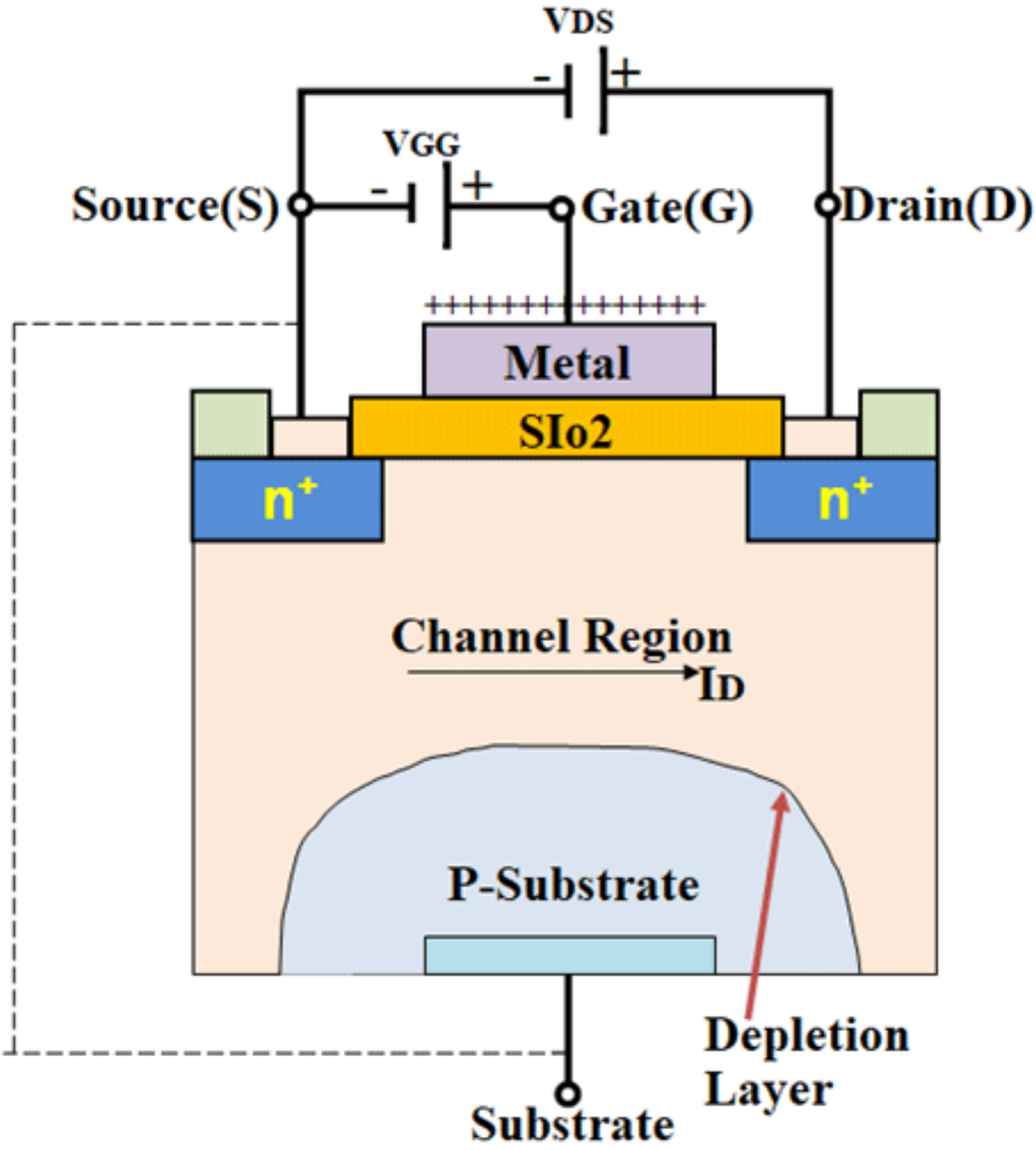
Technologie MOSFET

- **MOSFET** (Metal Oxide Semiconductor Field Effect Transistor)
- tranzistory *řízené polem* (FET = Field-effect Transistor)
- také označované jako *unipolární*
- mají tři elektrody S(ource), G(ate), D(rain): Gate je řídicí
- přítomnost napětí na G umožní průtok proudu mezi S a D - tranzistor se otevře

Technologie MOSFET



Obr. 8. Struktura MOSFET tranzistoru



Obr. 9. Práce MOSFET tranzistoru

Fundamentals of MOSFETs and Their Working

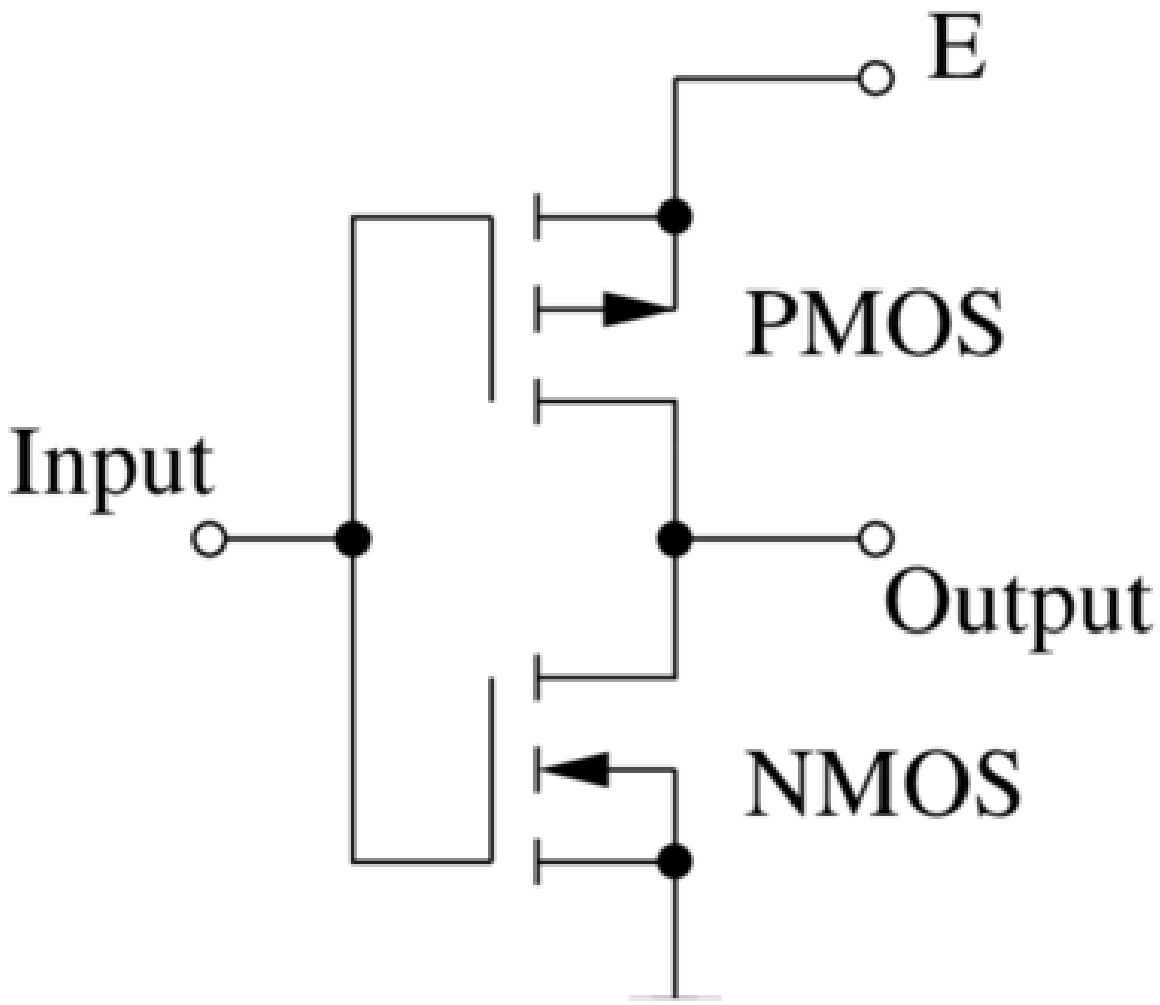
Technologie CMOS

> CMOS (anglicky Complementary Metal–Oxide–Semiconductor) je způsob vytváření logických členů vynalezený v 60. letech 20. století a zároveň technologie, kterou se po roce 1985 vyrábí naprostá většina logických integrovaných obvodů, včetně mikroprocesorů, jednočipových počítačů a elektronických pamětí, ale také analogových obvodů, jako jsou snímače obrazu, datové konvertory, zesilovače a transceivery používané v telekomunikacích.
>
> — Wikipedia, https://cs.wikipedia.org/wiki/CMOS

Co je CMOS?

- CMOS není technologií *jednoho* tranzistoru.
- CMOS je kombinací *obou polarit MOSFET*, tzn. **PMOS** a **NMOS** na jednom čipu.
- Dnes standardem pro výrobu logických čipů.
- Mají nízkou spotřebu energie díky tomu, že v klidovém stavu nevyžadují průtok proudu.
- Spotřeba nastává prakticky jen při přepínání, tedy než se stav obvodu ustálí.

Invertor s technologií CMOS

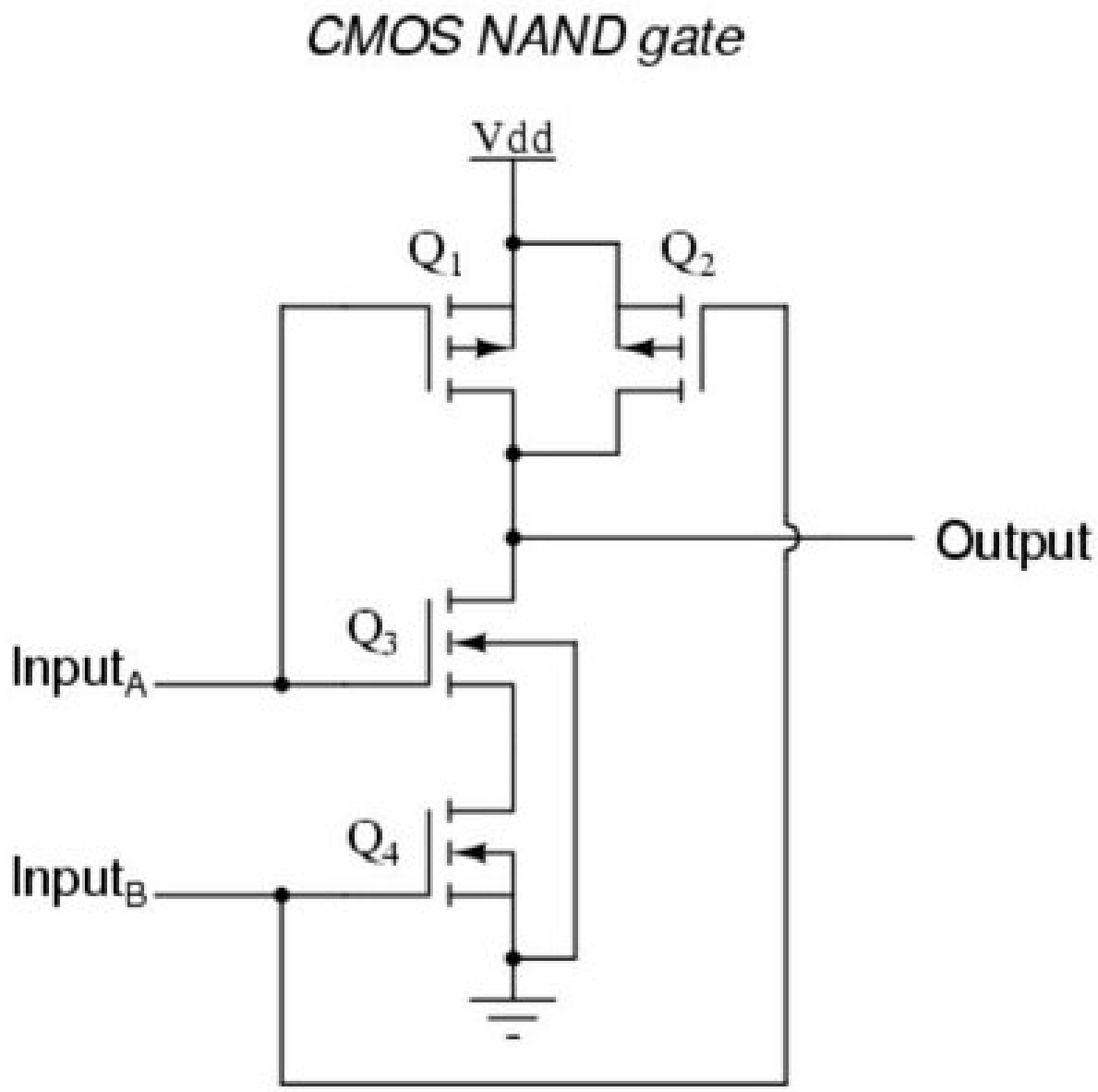


Obr. 10. Podle napětí na vstupu vede (je otevřen) buďto horní nebo spodní tranzistor

Fungování CMOS invertoru

- na G (Input) vyšší kladné napětí (= úroveň H) ⇒ vede spodní NMOS tranzistor a na jeho horní elektrodě (Output) je napětí téměř rovné 0 (= úroveň L) ⇒ invertuje vstupní napětí
- na G (Input) nulové nebo záporné napětí ⇒ vede horní PMOS tranzistor a na jeho spodní elektrodě (Output) je vyšší napětí téměř rovné napájecímu (E) ⇒ opět invertuje vstupní napětí

NAND ze 4 tranzistorů

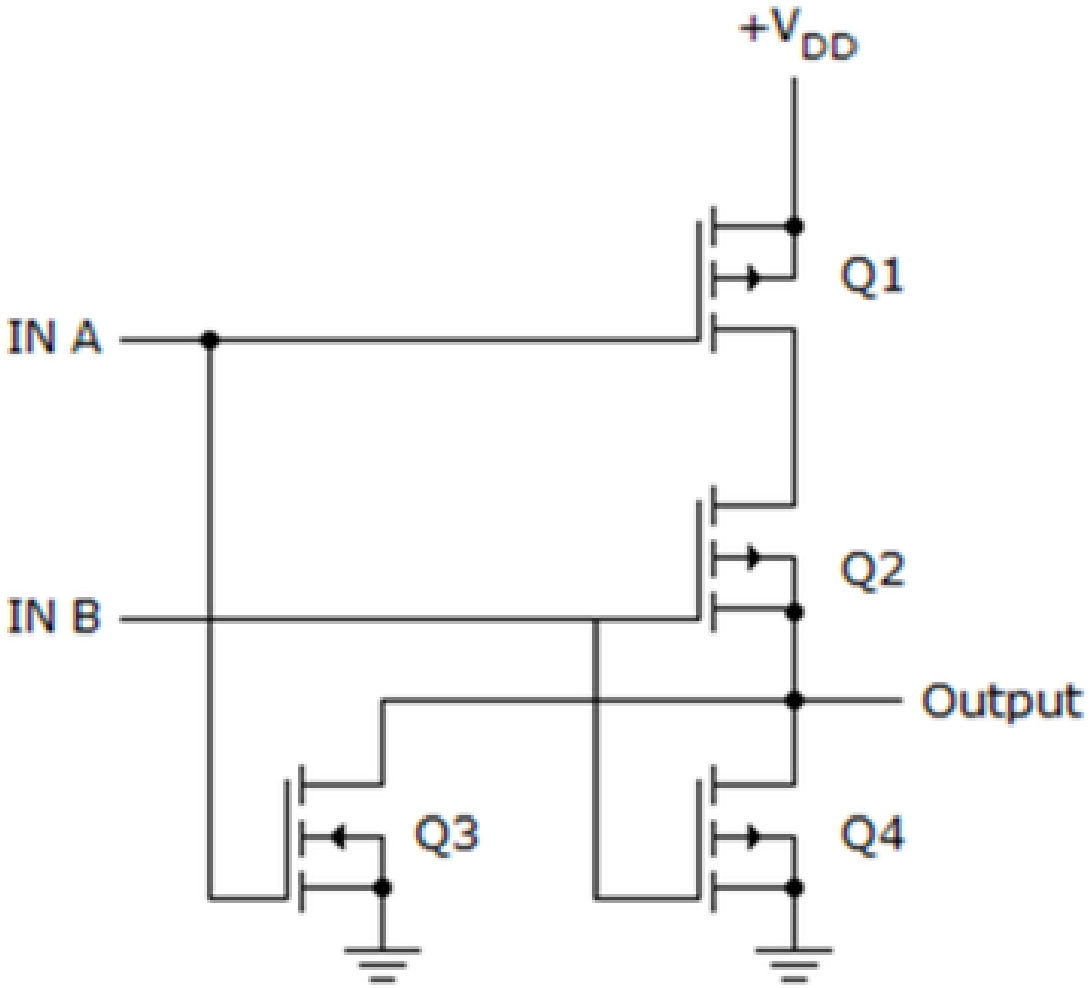


Obr. 11. NAND ze dvou NMOS (dole) a dvou PMOS tranzistorů (nahoře)

Chování

1. Přivedením nízkého (L) napětí na kterýkoli ze vstupů
2. sepne příslušný horní PMOS tranzistor a na výstupu se objeví úroveň 1 (H).
    - Jinak je tam stabilně 0 (L).

NOR ze 4 tranzistorů



Obr. 12. NOR ze dvou PMOS (dole) a dvou NMOS tranzistorů (nahoře)

Chování

1. Přivedením vysoké (H) úrovně napětí na kterýkoli ze vstupů
2. sepne příslušný dolní PMOS tranzistor a na výstupu se objeví úroveň 0 (L).
    - Jinak je tam stabilně 1 (H).

Stupně integrace

- V zásadě vyjadřuje, kolik jednotlivých prvků (typicky tranzistorů) se na jeden kus křemíku (čip) vejde.
- Čím vyšší stupeň integrace, tím více prvků a tím větší výkon/schopnosti daného čipu.
- Většinou také roste spotřeba proudu a tím generování tepla, ale musí se konstruovat tak, aby spotřeba rostla pomaleji, než roste výkon.

Stupně integrace: nižší

Zkratka	Význam	Počet součástek v pouzdře	Příklady obvodů
SSI	Small Scale Integration	do 30	Hradla NAND, NOR, invertory (např. 7400, 7404)
MSI	Medium Scale Integration	do 1 000	Sčítače, multiplexory, registry (např. 74161, 74138)
LSI	Large Scale Integration	do 100 000	Procesor Z80, řadič disku, čipová sada 8255

Stupně integrace: vyšší

Zkratka	Význam	Počet součástek v pouzdře	Příklady obvodů
VLSI	Very Large Scale Integration	do 10 000 000	Mikroprocesory jako 80386, grafické čipy, paměti DRAM
ULSI	Ultra Large Scale Integration	do 1 000 000 000	Moderní CPU (Intel Core, AMD Ryzen), GPU (NVIDIA RTX)
GSI	Gigantic Scale Integration	více než 1 miliarda	Nejnovější AI čipy (např. Apple M1/M2, NVIDIA H100, Cerebras WSE)

Na další kapitolu: Sekvenční logika

HTML · Slides · PDF | Přehled předmětu