Přehledové Texty sebrané z různých zdrojů ---

http://programujte.com/clanek/2006012803-ze-sesitu-cislicove-techniky-br-0000-0011-dil-pocitani-v-binarni-soustave/

http://programujte.com/clanek/2006053002-ze-sesitu-cislicove-techniky-br-0001-0011-dil-7400-potreti-a-ne-naposledy/

LOGICKÉ OBVODY

Logický obvod je fyzikální systém (např. integrovaný obvod), jehož každá veličina na výstupu závisí na velikosti vstupních veličin, při čemž tyto mohou mít jen konečný počet stavů (hodnot) - nejčastěji dvě. Chování těchto obvodů lze vyjádřit matematicky pomocí tzv. logické algebry.

Logická algebra (algebra logiky)

Je to dvouhodnotová algebra. Posuzuje pravdivost nebo nepravdivost určitého výrazu na základě pravdivosti a nepravdivosti dílčích výroků.

Výrok - je to logická veličina *nezávisle proměnná*. Je to každé *tvrzení*, o kterém má smysl prohlásit, zda je *pravdivé nebo nepravdivé*.

- výrok pravdivý: pravda, log „1“, H, T
- výrok nepravdivý: nepravda, log „0“, L, F

Výraz - je to logická veličina *závisle proměnná*. *Pravdivost* celého výrazu není *určena* množstvím dílčích výroků, ale *podmínkami, kterými jsou výroky vzájemně vázány* (∪, ∩, negace).

Příklad 1:

Důvodem k zastavení vozidla mohou být tyto tři příčiny*: 1. překážka, 2. červené světlo, 3. dopravní značka.* Každá z těchto příčin k zastavení buď je anebo není přítomna. Je-li přítomna jedna nebo více z těchto příčin, řidič se rozhodne zastavit.

1 překážka je v cestě	2 červené světlo svítí	3 dopravní značka je postavena	Rozhodnutí zastavit	Rozhodnutí jet dále
NE	NE	NE	NE	ANO
NE	NE	ANO	ANO	NE
NE	ANO	NE	ANO	NE
NE	ANO	ANO	ANO	NE
ANO	NE	NE	ANO	NE
ANO	NE	ANO	ANO	NE
ANO	ANO	NE	ANO	NE
ANO	ANO	ANO	ANO	NE

Příklad 2:

K přejetí vozidla přes železniční přejezd jsou nutné tyto tři podmínky: *1. vytažení závor, 2. zhasnutí červeného světla, 3. nepřítomnost výstražného signálu.*

Přes přejezd lze přejet jen tehdy budou-li splněny všechny tři podmínky.

1 závory vytaženy	2 červené světlo nesvítí	3 výstražný signál nepřítomen	Rozhodnutí jet dál	Rozhodnutí zastavit
NE	NE	NE	NE	ANO
NE	NE	ANO	NE	ANO
NE	ANO	NE	NE	ANO
NE	ANO	ANO	NE	ANO
ANO	NE	NE	NE	ANO

ANO	NE	ANO	NE	ANO
ANO	ANO	NE	NE	ANO
ANO	ANO	ANO	ANO	NE

Boolova algebra

Je to dvouhodnotová logická algebra, která pracuje se třemi základními Boolovými funkcemi: logický součin, logický součet, negace. Každou funkci můžeme vyjádřit algebraicky a pro různé hodnoty vstupních proměnných můžeme napsat hodnotu výstupní veličiny pomocí pravdivostní tabulky.

Pravdivostní tabulka

Udává stav na výstupu logického obvodu při daných hodnotách vstupních signálů.

Základní logické funkce

1. Negace

Je to funkce jedné proměnné, u které má závisle proměnná vždy opačnou hodnotu než nezávisle proměnná. V technické praxi to znamená, že činnost jednoho prvku je dána nečinností prvku druhého. Člen, kterým se realizuje logická negace se nazývá **invertor** (NE) nebo **negátor**. Má jeden vstup a jeden výstup. Na výstupu je vždy opačná hodnota logické proměnné přivedené na vstup.

$$Y = \overline{a}$$

a	Y
0	1
1	0

2. Logický součet (disjunkce, sjednocení)

Výrok je pravdivý, jestliže je pravdivý alespoň jeden elementární výrok, tzn. že je hodnota této fce Y = 1, jestliže alespoň jedna vstupní veličina má hodnotu 1. Tato fce je realizována logickým členem **OR** (NEBO). Musí mít alespoň dva vstupy. Pokud je spojíme paralelně, bude stav výstupu sledovat stav vstupu - získáme tzv. **sledovač.**

$$Y = a + b$$

a	b	Y
0	0	0
0	1	1
1	1	1
1	0	1

3. Logický součin (konjunkce, průnik)

Výrok je pravdivý jen tehdy, jestliže jsou pravdivé elementární výroky, tzn. že hodnota této fce Y = 1, pouze v případě, že všechny vstupní proměnné mají hodnotu 1. Funkce je realizována logickým členem **AND** (A). Při spojení vstupů získáváme opět **sledovač.**

$$Y = a \cdot b$$

a	b	Y
0	0	0
0	1	0
1	1	1
1	0	0

Pomocí těchto tří zákl. funkcí můžeme vyjádřit libovolně složitou funkci se 2 nebo více nezávisle proměnnými

Další logické funkce

4. Negovaný logický součet (Pierceova fce)

Tato fce nabývá hodnoty Y = 1 pouze tehdy, mají-li současně oba vstupy hodnotu logická 0. Funkce je realizována logickým členem **NOR**. Spojením vstupů jím lze realizovat **negaci**.

$$Y = \overline{a} + \overline{b}$$

a	b	Y
0	0	1
0	1	0
1	1	0
1	0	0

5. Negovaný logický součin (Shefferova fce)

Tato fce nabývá hodnotu Y = 0 pouze tehdy, mají-li současně oba vstupy hodnotu logická 1. Ve všech ostatních případech je Y = 1. Funkce je realizována logickým členem NAND. Spojením vstupů jím lze realizovat též negaci.

a	b	Y
0	0	1
0	1	1
1	1	0
1	0	1

$$Y = \bar{a} \cdot \bar{b}$$

6. Ekvivalence (totožnost, shodnost)

Tato fce nabývá hodnoty Y = 1 pouze tehdy a jen tehdy, mají-li současně oba vstupy shodné hodnoty. Tento člen je komparátorem shodnosti proměnných. Funkce je realizována log. členem **AND - NOR.**

a	b	Y
0	0	1
0	1	0
1	1	1
1	0	0

$$Y = a \cdot b + \bar{a} \cdot \bar{b}$$

7. Antivalence (nonekvivalence, neekvivalence, různoznačnost)

Tato fce nabývá hodnoty Y = 1 pouze tehdy, mají-li současně oba vstupy různé hodnoty. Funkce je realizována logickým členem **EXCLUSIVE-OR**.

a	b	Y
0	0	0
0	1	1
1	1	0
1	0	1

$$Y = a \cdot \bar{b} + \bar{a} \cdot b$$

Logické fce se dají vyjádřit několika způsoby:

- algebraickým výrazem
- pravdivostní tabulkou
- řádkovým schématem
- mapou (Karnaughova, Svobodova)

Všechny tyto způsoby jdou mezi sebou **vzájemně převádět**.

ZJEDNODUŠOVÁNÍ LOGICKÝCH FUNKCÍ

Logické funkce jsou často zbytečně složité. Opakují se v nich mnohokrát stejné proměnné, které často nemají na výslednou hodnotu logické funkce vliv. Při realizaci potom musíme použít zbytečně mnoho logických členů nebo kontaktů a zapojení se stává nepřehledné. Stejného výsledku lze dosáhnout použitím daleko menšího počtu prvků - někdy stačí méně než polovina. Původní logickou funkci, která bývá *ve tvaru součtu součinů*, musíme nejprve zjednodušit. Používáme přitom některé zákony Booleovy algebry. Pro zjednodušování se používají algebraické a grafické metody.

a) algebraické metody

- Z jednotlivých součinových členů můžeme většinou *vytknout jednu nebo více proměnných*. Vybíráme takové členy, aby *po vytknutí* zbyla z jednoho členu *jedna proměnná v přímém tvaru a z druhého stejná proměnná ve tvaru negovaném*. Jejich součet je roven jedné a získáváme tak ze dvou členů člen jeden, který obsahuje o jednu proměnnou méně.
- Po vynásobení součinových členů se v některých objevuje *jedna nebo více proměnných v přímém a negovaném tvaru*. Jejich součin se rovná nule a celý takovýto *člen můžeme proto vypustit*, aniž by se hodnota logické funkce změnila.
- Vyskytne-li se v rovnici jedna proměnná osamoceně, můžeme zjednodušovat tak, že stejné proměnné v ostatních výrazech můžeme nahradit nulou nebo jedničkou dle těchto pravidel:
- *při součinu* se stejné proměnné ve stejném tvaru nahradí log. 1, v negovaném tvaru se nahradí log. 0

  $y = a \cdot f(a, \bar{a}, b, \bar{b}, c, \bar{c}, \ldots)$ $\quad y = \bar{a} \cdot f(a, \bar{a}, b, \bar{b}, c, \bar{c}, \ldots)$

  $y = a \cdot f(1, 0, b, \bar{b}, c, \bar{c}, \ldots)$ $\quad y = \bar{a} \cdot f(0, 1, b, \bar{b}, c, \bar{c}, \ldots)$

  součin s a: $a \rightarrow 1$ $\quad a \rightarrow 0$
- při součtu se stejné proměnné ve stejném tvaru nahradí log. 0, v negovaném tvaru se nahradí log. 1

  $y = a + f(a, \bar{a}, b, \bar{b}, c, \bar{c}, \ldots)$ $\quad y = \bar{a} + f(a, \bar{a}, b, \bar{b}, c, \bar{c}, \ldots)$

  $y = a + f(0, 1, b, \bar{b}, c, \bar{c}, \ldots)$ $\quad y = \bar{a} + f(1, 0, b, \bar{b}, c, \bar{c}, \ldots)$

  součet s a: $a \rightarrow 0$ $\quad a \rightarrow 1$

b) grafické metody

Nejpoužívanější grafickou metodou pro zjednodušování logických funkcí je používání **Karnaughovy mapy**. Zjednodušování *je založeno na vztahu* $a + \bar{a} = 1$ (zákon vyloučení třetího - kontradikce).

Karnaughovu mapu sestrojujeme:

a) ***ze zadané logické funkce*** (nejčastěji v obecném tvaru) – musíme ji nejprve pomocí pravdivostní tabulky nebo použitím některých pravidel Boolovy algebry *převést na standardní součtový tvar*.

b) **z pravdivostní tabulky** tak, že každému řádku pravdivostní tabulky je přiřazeno určité políčko mapy.

Obecný tvar logické funkce

Funkce je zapsána ve formě součtu součinů, ve kterých se vyskytují proměnné v přímém nebo negovaném tvaru.

Standardní součtový tvar logické funkce

V součinových členech jsou všechny proměnné, vyskytující se v dané logické funkci.

Pokud v některém součinu některá proměnná chybí, vynásobíme tento člen součtem chybějící proměnné v přímém a negovaném tvaru.

Např. rovnici $Y = a\,b\,c + \bar{a}\,\bar{b}$ upravíme takto:

$$Y = a\,b\,c + \bar{a}\,\bar{b}\,(c + \bar{c}) = a\,b\,c + \bar{a}\,\bar{b}\,c + \bar{a}\,\bar{b}\,\bar{c}$$

Logickou funkci v tomto tvaru již můžeme zapsat do Karnaughovy mapy. Každému členu (součinu) je přiděleno odpovídající pole KM.

Z KM sestavujeme přímo zkrácené tvary rovnic. Řídíme se přitom tímto pravidlem:
Členy dvou sousedních polí se od sebe liší jen jednou proměnnou a tou je můžeme krátit.
Sousední pole - dvě vedle sebe ležící pole s vepsanými hodnotami 1, která *se* od sebe *liší jen v jedné proměnné* a také pole, která leží na obou krajích téhož řádku nebo téhož sloupce.

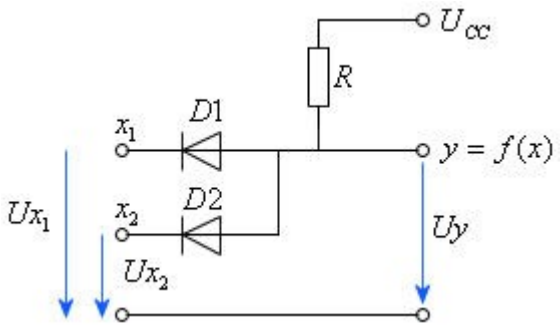
1. Dvě sousední pole obemkneme smyčkou - ve smyčce může být pouze sudý počet polí.
2. Sousedí-li spolu tři pole, obemkneme je dvěma smyčkami a pro každou z nich odečteme jeden zkrácený člen
3. Obemkneme-li smyčkou čtyři pole, můžeme zkrátit dvě proměnné a získáme tak jeden člen, ve kterém jsou obsaženy jen ty proměnné, které jsou obsaženy ve stejném tvaru ve všech polích.

Jedno pole může být obsaženo ve dvou smyčkách

Dnes nebo kdykoliv jindy si v tomto článku můžete přečíst o tom, co to je diodová logika, jak se zapojení s ní chová a jak vlastně funguje.

První něco o tom, co to je. Je to vlastně funkční zapojení, které se skládá z diod a rezistorů, a umožňuje nám z něj vytvářet dvě základní hradla AND a OR a z nich potom skládat složitější zapojení. Diodová logika je poměrně rychlá – až 200 MHz. Rychlosti může bránit pouze parazitní kapacita diod, ale tento problém řeší takzvané rychlé diody.

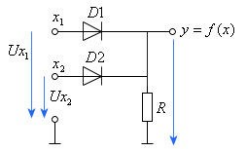
Teď se již podíváme na některá zapojení. Prvním z nich bude logický součin. Jak vypadá tabulka funkce, to už by vám mělo být jasné, my si ale tabulku uděláme, jen s jiným obsahem. Zajímat nás budou samozřejmě i logické jedničky a nuly, ale především to tentokráte budou napětí. Zbytek vysvětlím na obrázku.



Ux_2	Ux_1	Uy	f(x)
$0V$	$0V$	$0,7V$	0
$0V$	U_{cc}	$0,7V$	0
U_{cc}	$0V$	$0,7V$	0
U_{cc}	U_{cc}	U_{cc}	1

Na obrázku si můžete prohlédnout zapojení logického hradla AND složeného ze dvou diod a jednoho odporu. Nyní budeme postupovat podle tabulky: První připojíme oba dva vstupy x1 i x2 k zemi. Nyní poteče proud přes odpor a přes diody na zem, a tudíž na výstupu y nebude prakticky žádné výstupní napětí (jen 0,7 V), které se ztrácí na polovodičovém přechodu na diodě. I nadále budeme pokračovat dle tabulky a vstup x1 připojíme k Ucc a vstup x2 necháme na zemi. Dioda na vstupu x1 bude uzavřena a nepoteče jí žádný proud, ale přes diodu D2 poteče proud na zem, a tudíž se opakuje případ z minula, kdy byly obě diody připojeny k zemi. Totéž nastane, prohodíme-li výstupy. Ale pokud připojíme oba dva vstupy na Ucc, potom ani jednou diodou neprochází proud, a tudíž je celé napětí Ucc na výstupu y. A poprvé získáváme logickou 1.

Nyní jsem si řekli to podstatné o logickém součinu a přejdeme na logický součet. Budeme potřebovat tytéž součástky. Jeden odpor a dvě diody. Zapojení se ale bude celkem lišit.



Ux_1	Ux_2	Uy	$f(x)$
$0V$	$0V$	$0V$	0
$0V$	$10V$	$9,3V$	1
$10V$	$0V$	$9,3V$	1
$10V$	$10V$	$9,3V$	1

Opět bude nejlepší to vysvětlit na obrázku. Prvně spojíme oba dva vstupy se zemí. Jak vidíme, diodami neprochází žádný proud a na výstupu je tudíž 0 V. Pokud ale jakoukoliv z diod připojíme na napájecí napětí (v našem případě je to 10 V), bude diodou tento proud procházet na výstup, na diodě vznikne úbytek asi 0,7 V a na výstupu tudíž bude přibližně 9,3 V a tato hodnota je považována za logickou jedničku.

KOMBINAČNÍ LOGICKÉ OBVODY

Jsou to takové obvody, u nichž jsou hodnoty výstupních veličin jednoznačně určeny kombinací hodnot vstupních veličin - nezávisejí tedy na předchozím stavu. Je to např. sčítačka, nebo různé převodníky.

PŘEVODNÍKY

Převádějí čísla vyjádřená v jednom kódu do kódu jiného. Nejčastěji se realizuje převod čísla z desítkové soustavy do dvojkové soustavy v určitém kódu.

Převodník do kódu NBCD

Převodník můžeme realizovat čtyřmi logickými členy NAND tak, že výstup každého z nich odpovídá jedné dvojkové číslici. Vstupní obvod je tvořen rezistory, jimiž se jednotlivé vstupy log. členů udržují na úrovni logické hodnoty 1. Zapnutím kteréhokoliv spínače se příslušná úroveň 1 změní na úroveň 0. Logické členy NAND, jejíž vstupy se dostaly na úroveň log. hodnoty 0, mají na výstupu 1.

SEKVENČNÍ LOGICKÉ OBVODY

Hodnoty výstupních veličin jsou určeny nejen hodnotami veličin vstupních v daném časovém okamžiku, ale i hodnotami výstupních veličin z předcházejícího časového okamžiku. Sekvenční obvod se skládá z kombinačního logického obvodu a z paměťového obvodu. Na vstup kombinační části přicházejí vstupní signály a vnitřní proměnná z paměťového obvodu.

KLOPNÉ OBVODY

Jsou to sekvenční logické obvody, které se používají v obvodech číslicových zařízení, kde je třeba zachovat po určitou dobu signál s log. hodnotou 0 nebo 1. Jako elementární paměti pro jeden bit (jednu dvojkovou číslici) se používají bistabilní klopné obvody. Stav KO v určitém okamžiku je podmíněn stavem obvodu v okamžiku předchozím.

Klopné obvody mají jeden nebo dva řídící vstupy a jeden vstup pro hodinové impulsy (C - clock). Výstupem je signál Q a jeho negace Q non. Lze je realizovat pomocí logických členů typu NAND nebo NOR.

- **Asynchronní obvod** - výstupy obvodu se mění ihned po příchodu vstupních signálů.
- **Synchronní obvod** - výstupy obvodu se mění až po příchodu tzv. synchronizačního (hodinového) impulsu na zvláštní vstup označení C.

Pravdivostní tabulka

Udává stav na výstupu po příchodu hodinového (synchronizačního) impulsu při daných vstupních signálech.

Tabulka buzení

Vyjadřuje jaký musí být vstupní signál pro dosažení požadované změny na výstupu v následujícím čase.

Klopný obvod RS

Má dva vstupy - R (reset - nulování) a S (set - nastavení). Přijde-li signál 1 na vstup S, překlopí se obvod v následující době do stavu 1. Pokud v něm již byl, potom v něm zůstane. Přijde-li signál 1 na vstup R, překlopí se obvod do stavu 0. Pokud v něm byl, potom v něm zůstane. Přivedeme-li na oba vstupy signál 0, zůstane stav KO nezměněn. Přivedeme-li na oba vstupy signál 1, což je nepřípustné, nastaví se oba výstupy nejdříve do stavu 0 a konečný stav bude záležet na zpoždění použitých členů, nebo na tom, který ze signálů R a S dříve skončí. Tento stav označujeme jako hazardní stav KO (značíme X).

Klopný obvod D

Je vytvořen z klopného obvodu RS tak, že na vstup R je přes invertor připojen vstup S. Dosáhneme tím vyloučení neurčitého stavu. Vytváří zpoždění signálu přicházejícího na vstup D o jeden takt, tj. o časový interval mezi dvěma hodinovými impulsy. Příchodem hodinového impulsu se informace přítomna na vstupu D přesouvá na výstup Q.

Klopný obvod JK

Je to vylepšený synchronní klopný obvod RS. Získáme ho použitím třívstupových členů NAND na vstupu. Na jeden se kromě hodinových impulsů přivádí signál J a výstupní signál Q non. Jeden z členů je vždy ve

stavu 0, takže body R a S se současně nemohou dostat do stavu 1 ani když stav vstupů J a K je 1. V takovém případě se vždy původní stav změní na stav opačný. Tento KO se tudíž nemůže dostat do hazardního stavu.

Klopný obvod T

Má pouze jeden vstup. Stav na výstupu se při každém hodinovém impulsu na vstupu C změní na opačný, je-li na řídícím vstupu T log. hodnota 1. Přivedeme-li na vstup log. hodnotu 0, KO svůj stav nemění. Chová se tedy jako dvojkový dělič, tj. dělí počet vstupních impulsů dvěma.

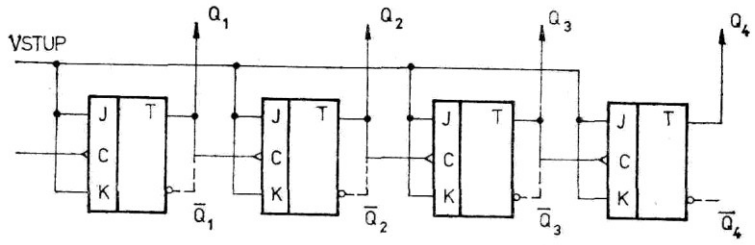
ČÍTAČE

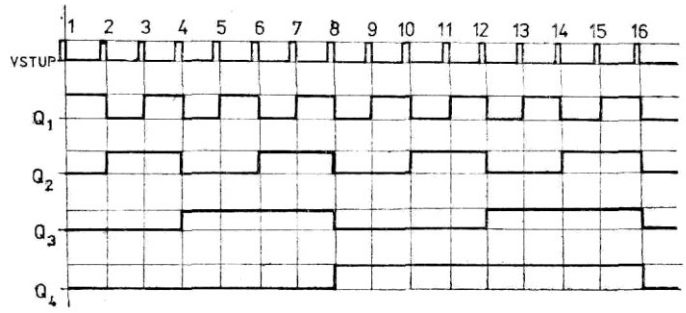
Čítač je obvod, který počítá (čítá) impulsy. Jejich počet určíme ze stavu klopných obvodů, z nichž jsou sestaveny (JK, D). Po vyčerpání všech dovolených stavů jednotlivých KO se čítač vrátí do výchozího stavu. Mohou pracovat v různých kódech - nejčastěji dvojkový a BCD. Podle způsobu čítání je dělíme na čítače vpřed, vzad a vratné čítače. Z hlediska řízení je rozdělujeme na asynchronní a synchronní čítače.

Asynchronní čítače

Každý KO je spuštěn předcházejícím KO, takže poslední KO nemůže změnit stav, dokud předcházející KO své stavy nezmění. Zpoždění jednotlivých KO se sčítá a tím je omezena max. rychlost čítání. Tato se zmenšuje se vzrůstajícím počtem KO.
Výhody: jednoduchost a *minimální zatěžování zdroje čítaných impulsů*.

Příklad asynchronního čítače





Časový diagram

DESÍTKOVÝ EKVIVALENT	VÝSTUPY Q_4	Q_3	Q_2	Q_1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Pravdivostní tabulka

Synchronní čítače

Vstupy hodinových impulsů jsou vzájemně propojeny. Všechny stupně čítače, které mají při čítání měnit svůj stav ho změní najednou. Maximální dosažitelná frekvence je určena zpožděním v jednom stupni čítače. Hlavní nevýhodou je podstatně *větší zatěžování zdroje* čítaných impulsů.

POSUVNÉ REGISTRY

Jsou sestaveny z klopných obvodů JK nebo D a lze do nich vložit informaci a vnějšími řídícími impulsy ji posouvat, popř. ponechat v registru obíhat po dobu, kdy ji není třeba zpracovávat. Počet KO udává délku posuvného registru, a tím i počet řádů dvojkového čísla, které můžeme do registru uložit. Posuvné registry rozdělujeme:

Podle směru posuvu vložené informace:

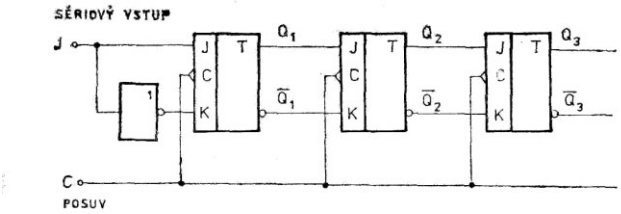
- **registry s posuvem vpřed** – informace se posouvá zleva doprava
- **vratné registry** – lze měnit směr posuvu vnějším řídícím signálem a posouvat informaci vlevo nebo vpravo
- **kruhové registry** – informace obíhá ve směru vstup – registr – výstup – vstup, popř. opačným směrem

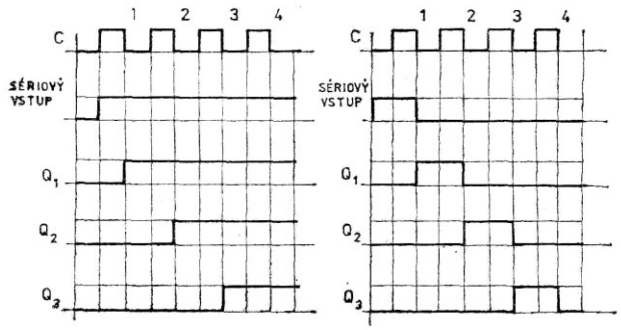
Podle uspořádání výstupu:

- **registry se sériovým výstupem** – posouvaná informace se objevuje postupně po jednom bitu na jediném výstupu posledního KO obvodu.
- **registry s paralelním výstupem** – posouvanou informaci lze odebírat současně z výstupů jednotl. KO.

Posuvný registr se dá použít jako:

- **paměť** pro nejrůznější použití, např. pro ukládání mezivýsledků
- **převaděč ze sériového způsobu činnosti na paralelní** - informace se sériově posune do posuvného registru, načež se z jednotlivých KO současně odebere
- **převaděč z paralelního způsobu činnosti na sériový** - informace se současně vloží do všech KO, načež se postupně (sériově) na výstupu registru snímá
- **zpožďovací člen** - výstup z posledního KO je zpožděn proti vstupu prvního KO o počet taktů rovnající se kapacitě registru (počtu KO)
- **kruhový čítač** - výstup čítače vyvedeme zpět na jeho vstup. Zapíšeme-li jedničku do prvního KO na počátku čítání, pak její poloha v posuvném registru určuje počet hodinových impulsů přivedených na vstup.





C	VST. INF.	STAVY KLOP. OBVODU Q_1	Q_2	Q_3
—	—	0	0	0
1	1	1	0	0
2	1	1	1	0
3	1	1	1	1
4	0	0	1	1
5	0	0	0	1
6	0	0	0	0

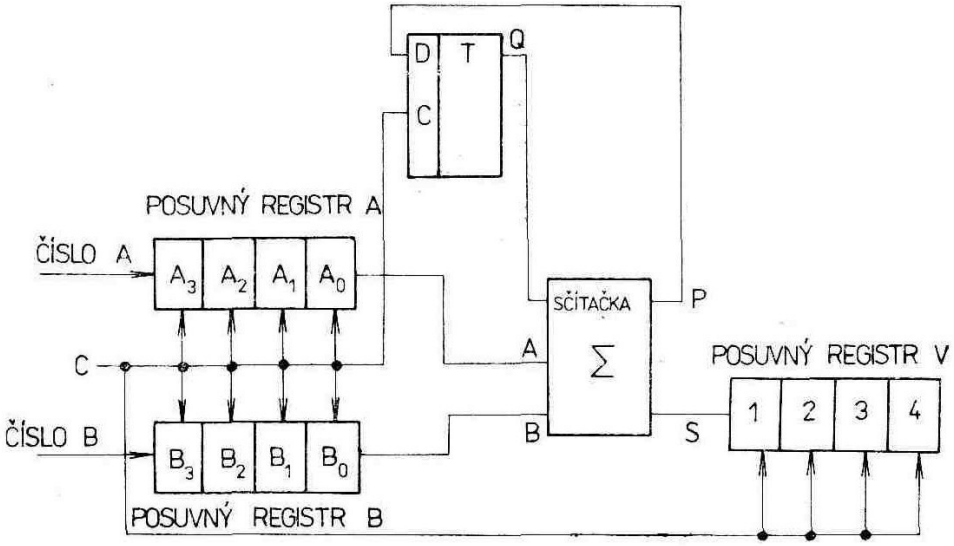
Použití posuvných registrů k sériovému sčítání

Čísla *A* a *B,* která chceme sčítat v dvojkovém kódu, jsou zapsána (vložena) do posuvných registrů *A a B,* jejichž délka je dána počtem bitů čísel, která se mají sčítat. Zápis čísel do registrů může být sériový nebo paralelní. Čísla jsou v registrech uspořádána tak, že nejnižší řád (bit) je nejblíže k výstupu.

Při prvním synchronizačním impulsu sčítá sčítačka nejnižší bity obou sčítanců (A_0, B_0) a přenos *P* se uloží do klopného obvodu *D,* který pracuje jako paměť. Součet *S* se uloží do posuvného registru *V*.

Při dalším synchronizačním impulsu se sečtou číslice druhého řádu (A_1,B_1) a k nim se přičte případný přenos z nejnižšího řádu. Současně se součet přesouvá z prvního místa na druhé místo v posuvném registru *V*. Po ukončení sčítání je celý výsledek sčítání uložen v posuvném registru *V*.

Sčítačka připojená na výstup registrů *A* a *B* je kombinační logický obvod sestavený ze základních logických členů.


D
T
Q
C
POSUVNÝ REGISTR A
ČÍSLO A
A3
A2
A1
A0
SČÍTAČKA
P
C
A
Σ
POSUVNÝ REGISTR V
ČÍSLO B
B3
B2
B1
B0
B
S
1
2
3
4
POSUVNÝ REGISTR B

3. Booleova algebra, logické funkce a binární číslice (bity)

V předchozí kapitole jsme si řekli, že pomocí dvojkové soustavy je možné zaznamenat stavy různých objektů, které jsou převedeny na posloupnost bitů. S těmito posloupnostmi lze provádět různé operace. Pokud se například jedná o přirozená, čísla, je možné provést jejich součet, porovnání apod. (tím se budeme podrobněji zabývat v další části tohoto seriálu). Zajímavé a pro další použití binárních hodnot (bitů) v počítačích velmi důležité však je, že existuje celý matematický aparát určený pro zpracování logických výroků, které jsou založeny na dvou stavech – *pravda* či *nepravda*. Vzhledem k tomu, že se jedná pouze o dva stavy, je jejich mapování na binární číslice 0 a I zcela jednoznačné: 0 značí nepravdu a I pravdu.

Logické výroky mohou mít tvar výrazů, pro jejichž zpracování se používá matematický aparát nazvaný *Booleova algebra*, který je vybaven potřebnými pravidly a operátory. Jednotlivé operátory Booleovy algebry je dokonce možné poměrně jednoduchým způsobem implementovat pomocí elektrického obvodu, který pracuje s hodnotami *pravda* (I) a *nepravda* (0), reprezentovanými buď různými úrovněmi napětí nebo proudu.

Dosáhli jsme tedy následující shody s dalekosáhlými důsledky: jak pro informace (převedené na dvojková čísla), tak i pro způsob jejich zpracování (Booleova algebra, resp. logické funkce) je možné použít dvojkovou soustavu a tím pádem i elektrické obvody pracující pouze se dvěma různými stavy. To je velmi důležitý závěr, na jehož základě je postavena dnešní technologie počítačů. Mimo jiného to znamená, že části počítače určené pro zpracování informací i části počítače určené pro řízení a (logické) rozhodování mohou být postaveny na stejných typech obvodů.

4. Základní logické operátory

Základ Booleovy algebry spočívá v *logických proměnných*, které mohou nabývat pouze hodnot označených symboly 0 a I, a *logických funkcí n-proměnných*, které také mohou nabývat pouze hodnot 0 nebo I. Logické funkce jsou sestaveny pomocí *logických operátorů*. Jsou přitom definovány tři základní logické operátory, které mezi logickými proměnnými provádějí tři základní operace: **negaci**, **logický součin** a **logický součet**. Výsledky těchto základních operací lze vyjádřit buď vzorcem nebo tabulkou – tabulární formu je možné použít z toho důvodu, že jsou použity pouze dva stavy, na rozdíl od funkcí vracejících přirozené, reálné či dokonce komplexní hodnoty.

Nejjednodušší je funkce *negace*, což je funkce jedné proměnné vyjádřená vztahy:

```
!0=I
!I=0
```

Funkce (operace) logického součtu, neboli disjunkce (+) se také nazývá funkce **nebo** (**or**), což dává smysl, když si uvědomíme, že symbol 0 značí nepravdu a symbol I pravdu:

```
0+0=0
0+I=I
I+0=I
I+I=I
```

Všimněte si, že se v tomto případě nejedná o normální součet (v něm by výsledek poslední operace byl I0), ale o logický součet se dvěma symboly, nikoli čísly.

Funkce (operace) logického součinu, neboli konjunkce (^) se nazývá funkce **a** (**and**) a je vyjádřena následujícími vztahy:

```
0^0=0
0^I=0
I^0=0
I^I=I
```

S výše uvedenými operacemi a jejich vzájemnou kombinací je možné vyjádřit jakoukoli logickou funkci. Pro obě binární operace, tj. operaci logického součtu i součinu, platí běžné vlastnosti známé i z „klasické“ algebry: existence nulového prvku vzhledem k součtu, existence jednotky vzhledem

k součinu, asociativita, komutativita, distributivnost součinu vzhledem k součtu a navíc ještě distributivnost součtu vzhledem k součinu (ten se v „klasické“ algebře neobjevuje). Navíc ještě v Booleově algebře platí *de Morganovy zákony*, které se často používají pro zjednodušení logických funkcí:

```
existence nuly vzhledem k logickému součtu
x+0=x

uzavřenost operace logického součtu
x+I=I
x+x=x

existence nuly vzhledem k logickému součinu
x^0=0

existence jednotky vzhledem k logickému součinu
x^I=x

uzavřenost operace logického součinu
x^x=x

komutativnost logických operací
x+y=y+x
x^y=y^x

asociativnost logických operací
x+y+z=(x+y)+z=x+(y+z)
z^y^z=(x^y)^z=x^(y^z)

distributivnost obou! logických operací
x^(y+z)=x^y+x^z
x+(y^z)=(x+y)^(x+z)

de Morganovy zákony (všimněte si "prohození" součtu za součin při rozepsání
negace)
!(x+y)=!x ^ !y
!(x^y)=!x + !y
```

5. Úplné systémy logických funkcí a jejich význam

Vzhledem k tomu, že logické funkce jsou postavené pouze nad dvěma symboly, má smysl se ptát, kolik vlastně existuje navzájem rozdílných funkcí pro *n* proměnných. Tato otázka například postrádá praktický smysl v oboru reálných čísel, protože nám nijak nepomůže při dalším zjednodušování či úpravách algebraických výrazů, ovšem v případě Booleovy algebry smysl má, jak uvidíme dále.

Pro jednu logickou proměnnou existují 2^2=4 funkce: konstantní pravda, konstantní nepravda, identita a negace. Pro dvě proměnné existuje celkem 2^{2*2}=16 různých funkcí nabývajících všech možných hodnot pro všechny možné kombinace dvou proměnných. Pro tři proměnné je to již 2^{2*3}=256 různých funkcí a pro více proměnných tento počet neustále roste. Ovšem platí, že je vždy možné pomocí výše uvedených vztahů provést redukci těchto funkcí na tři základní funkce jedné a dvou proměnných popsané výše: negace, logický součin a logický součet.

Tato trojice funkcí tvoří *úplný logický systém*, tj. všechny možné kombinace lze dekomponovat na sadu právě těchto funkcí. To by ovšem mohlo značit, že při realizaci logických funkcí v počítači si vystačíme se třemi typy obvodů, neboli *logických členů*. Skutečně je tomu tak, negace, součin i součet dokonce dostaly vlastní značky použité v elektronických schématech. Poznamenejme, že se místo termínu *logický člen* také používá pojmenování **hradlo**.

6. Minimální úplné systémy logických funkcí

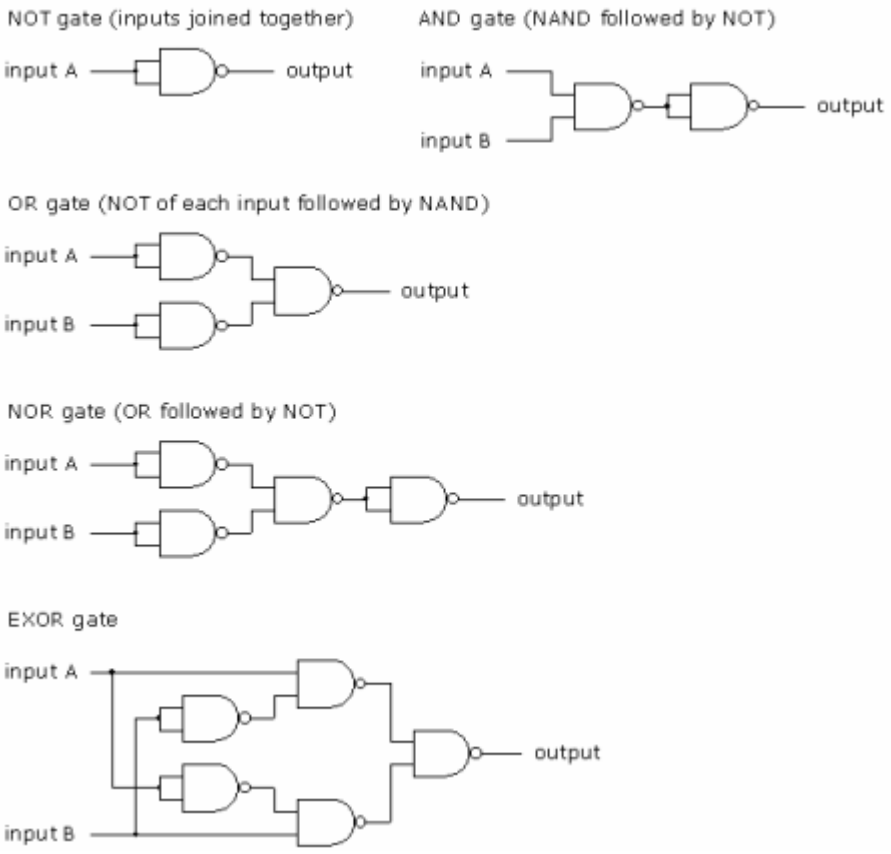
V praxi se většinou nesetkáme s přímým použitím logických členů, které by realizovaly logickou negaci, součet a součin. Namísto toho se (především v minulosti při použití integrovaných obvodů

s malou a střední integrací) těšily velké popularitě takzvané *minimální úplné systémy*, ve kterých se místo trojice rozdílných logických funkcí používaly buď funkce dvě (například **and** a **not**) nebo dokonce pouze jediná funkce. Jednou z variant minimálního úplného systému je systém používající funkci **not-or** neboli **NOR**, popř. funkci **not-and** neboli **NAND**. Důvodů, vedoucích ke snížení počtu funkcí, je více. Především dochází ke snížení různých typů součástek ve stavebnicích integrovaných obvodů, což se nepřímým způsobem projeví na výtěžnosti výroby i samotné ceně součástek.

Například do pouzdra integrovaného obvodu se 14 vývody lze integrovat celkem čtyři logické funkce se dvěma vstupy a jedním výstupem. Pokud návrh nějakého systému (třeba části řadiče či ALU) vede k použití dvou funkcí logického součinu a dvou funkcí logického součtu, znamenalo by to, že by musel buď existovat přesně tento integrovaný obvod, nebo by se musely použít integrované obvody dva, přičemž by polovina z jejich plochy zůstala nevyužita. V případě, že se logický součin i součet převede na jednu funkci, například **NAND**, použije se pouze jeden integrovaný obvod.

Druhou výhodou jsou naprosto shodné charakteristiky (elektrické – zatížitelnost, i časové) všech logických členů implementujících jedinou logickou funkci. Pokud by bylo použito více různých typů logických členů, pravděpodobně by při jejich realizaci docházelo k různým úpravám vedoucím například ke vzniku hazardů. Pro některé technologie (například TTL) je dokonce lepší implementovat **NAND** než přímou funkci **AND**.

Že je možné minimální logický systém vytvořit, dokazuje i následující obrázek, na němž jsou pomocí funkce **NAND** realizovány všechny tři základní logické funkce i další dvě funkce **NOR** a **XOR** (non-ekvivalence):



Pomocí hradla typu **NAND** lze realizovat i další logické funkce

U integrovaných obvodů s velkou a velmi velkou integrací se již setkáváme s jinými problémy při jejich návrhu, takže se v nich běžně používají i základní logické funkce či jejich „negované“ varianty – vše záleží na použité technologii, která určuje, které typy funkcí se snáze realizují a které musí být rozloženy na podfunkce.

1. Kombinační a sekvenční logické obvody v počítači

V předchozí části tohoto seriálu jsme si řekli, že činnost moderních počítačů je založená na Boolově algebře, logických funkcích a binární čili dvojkové číselné soustavě pracující s pouhými dvěma stavy, kterých mohou být reprezentovány pomocí binárních číslic (*bitů*). Základem Boolovy algebry jsou *logické proměnné*, které mohou nabývat pouze dvou hodnot 0 či I, a *logické funkce*, které taktéž mohou nabývat pouze hodnot 0 nebo I.

Logické proměnné jsou v počítači představovány většinou napěťovou úrovní, elektrickým proudem, úrovní magnetizace, polarizací paprsku atd. (samozřejmě v závislosti na použité technologii), zatímco logické funkce se implementují pomocí *logických členů* neboli *hradel*. Z logických členů se při návrhu počítače nebo jiného digitálního zařízení skládají větší celky, které jsou souhrnně nazývány *logické obvody* (ostatně jedna velmi dobrá kniha o této problematice má název „Od logických obvodů k mikroprocesorům“).

V praxi rozeznáváme dva typy logických obvodů. Jedná se o *kombinační logické obvody* a *sekvenční logické obvody*. Tyto obvody jsou sice na nižší úrovni sestaveny z těch samých členů, tj. hradel (a ještě níže pak většinou z tranzistorů), ale v jedné věci se od sebe zásadně odlišují. Výstupy kombinačních logických obvodů závisí pouze na vstupních kombinacích v daném časovém okamžiku (po ustálení případného přechodného děje). To znamená, že jedné vstupní kombinaci (vektoru bitů) odpovídá jediná a vždy stejná výstupní kombinace, tj. kombinační logický obvod nemá žádnou paměť, ve které by si mohl zapamatovat svůj předchozí stav.

Naproti tomu u obvodu sekvenčního závisí jeho výstupy nejen na kombinaci vstupů, ale i na jeho předchozím stavu. Z toho vyplývá, že sekvenční obvod musí obsahovat nějakou formu paměti, ve které má uložen svůj předchozí stav. Zajímavé je, že i tyto paměti jsou realizovány pomocí hradel, tj. obvodů, které samy o sobě paměťovou funkci nemají.

2. Tvorba složitějších logických funkcí pomocí kombinačních logických obvodů

Nejprve se budeme zabývat kombinačními logickými obvody, protože jsou jednodušší jak na návrh, tak i z hlediska jejich analýzy. Jak již bylo řečeno v úvodní kapitole, jsou kombinační logické obvody (ať již jsou jakkoli složité) složeny z hradel, které samy realizují jednoduché logické funkce s omezeným počtem vstupů. Typicky se jedná o funkce negace (NOT), logického součtu (OR), logického součinu (AND), negace logického součtu (NOR), negace logického součinu (NAND) a nonekvivalence (XOR), přičemž počet vstupů u typických hradel bývá od jednoho do čtyř, výjimečně i osm. Zkusme si nyní vypsat několik úkolů, které je možné s touto základní sadou hradel řešit, abychom viděli, že některé operace či instrukce probírané v dalších částech tohoto seriálu nejsou implementovány nějakými tajemnými postupy, ale „pouze“ sadou vhodně propojených hradel:

1. V mnoha případech stojíme před problémem vytvoření kombinačního obvodu, který by implementoval nějakou *základní logickou funkci o n-vstupních proměnných*, kde n>4. Se znalostí vlastností Boolovy algebry (asociativity, komutativity, distributivity součtu vzhledem k součinu i naopak, de Morganových zákonů atd.) je tento úkol lehce splnitelný. Většinou jsou hradla uspořádána do stromové struktury – na začátku (tj. v kořenech stromu) jsou vždy zpracovány dvě navzájem si odpovídající hodnoty, výsledek je předán do nadřazených uzlů stromu atd. až se dojde ke kořenu tvořeného jediným hradlem, na jehož výstupu je požadovaný výsledek.

2. Pomocí hradel typu logického součtu (OR) a logického součinu (AND) lze realizovat poměrně důležitou *funkci přepínání informace*. Princip je takový, že do vzniklého obvodu jsou přivedeny dva n-bitové vektory a na základě výběrového bitu (či signálu) se vybere vektor první či druhý. Podobný obvod je použit jak v operačních pamětech, tak i v samotných mikroprocesorech, například pro výběr registrů, jejichž obsah bude použit pro výpočet nějaké funkce v ALU.

3. Dalším snadno řešitelným úkolem může být *porovnání dvou n-bitových hodnot*. Zde je možné použít například sadu hradel nonekvivalence (XOR). Jak uvidíme v dalších částech seriálu, jedná se o operaci, pro kterou bývá v mikroprocesorech vyhrazen zvláštní operační kód, protože porovnání dvou hodnot představuje základ pro tvorbu rozeskoků a tím i řízení běhu programu.

4. Podobně lze, tentokrát ovšem pomocí hradel logického součtu (OR) realizovat test, zda je n-bitová hodnota nenulová, tj. zda je alespoň jeden bit nenulový či zda jsou všechny bity rovny nule. I tato operace je obsažena v instrukční sadě prakticky všech mikroprocesorů, z toho vyplývá, že funkce pro test nulovosti bývá součástí ALU (aritmeticko-logické jednotky).

5. I s další funkcí se můžeme v některých mikroprocesorech setkat. Jedná se o funkce pro výpočet sudé či liché parity, resp. paritního bitu. Paritní bit se přidává k užitečné informaci například pro kontrolu správnosti přenosu. Pro sudou paritu platí, že paritní bit je roven 0 v případě, že je počet jedniček ve zpracovávané informaci (bitovém vektoru) sudý nebo nulový. Lichá parita má přesně opačnou podmínku. Funkci pro výpočet parity lze realizovat sérioparalelním zapojením hradel realizujících nonekvivalenci (XOR).

3. Převody mezi různými kódy, multiplexory a demultiplexory

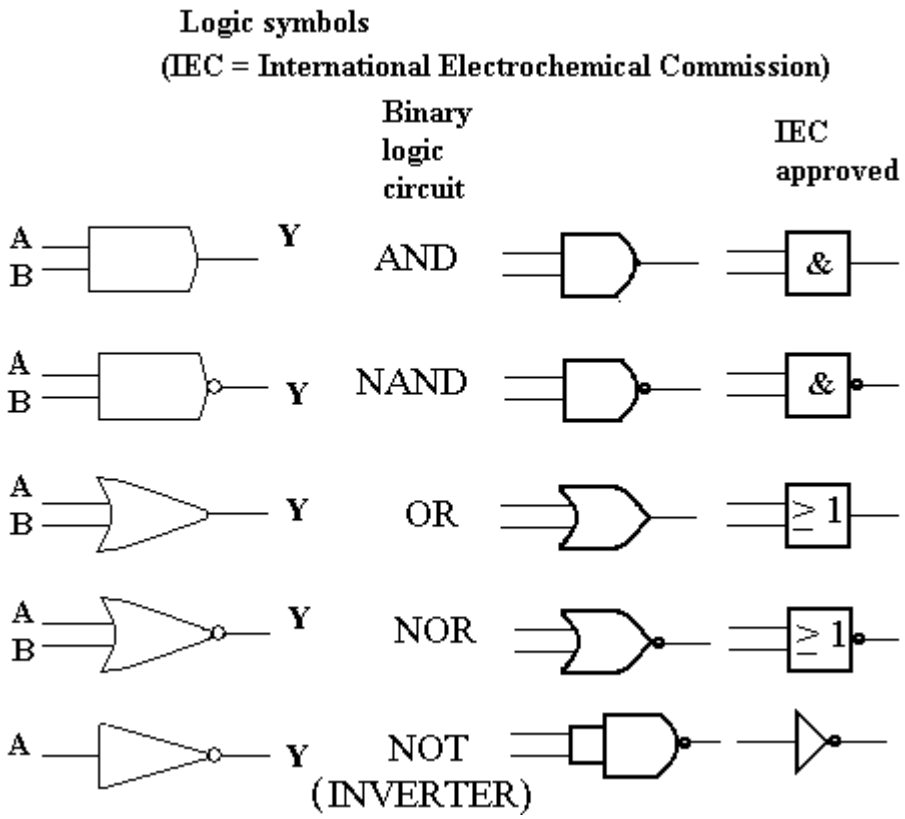
Poměrně často se v počítačích a dalších digitálních zařízeních používala taková zapojení hradel, která realizují *převody binárních hodnot* mezi různými kódy. Učebnicovým příkladem je převod jedné desítkové číslice reprezentované BCD kódem, tj. čtyřmi bity, na kód, který odpovídá zapojení sedmisegmentové zobrazovací jednotky, kterou všichni jistě znají z bezpočetného množství přístrojů – hodin, rádií, mikrovlnek, „takypočítače“ PMI-80 atd. Pro měření vzdálenosti či úhlu otočení se pro své výhodné vlastnosti používá takzvaný Grayův kód, který je před vlastním zpracováním měřené veličiny zapotřebí převést na poziční binární kód, který je již zpracovatelný v mikroprocesoru.

Příklad použití převodníků z nedávné minulosti: převod zvukových signálů z n-bitového pozičního binárního kódu na kód nelineární (μ-Law atd.), gamma korekce video signálu, pevná barevná paleta na některých osmibitových počítačích. Všechny tyto převodníky lze realizovat buď pomocí obvodů složených z hradel (tyto obvody se různými způsoby optimalizují, například na počet logických členů, maximální zpoždění atd.) nebo zapojením polovodičové paměti (typu ROM, PROM, EPROM, FLASH atd.): vstupní signály se zapojí na adresovou sběrnici a na sběrnici datové se po jisté době objeví přečtená informace.

Dalšími typy kombinačních obvodů jsou *multiplexory* a *demultiplexory*. Multiplexor (MUX) má na svém vstupu 2^n datových vodičů (binárních signálů či hodnot) a n adresových vstupů. Na základě zapsané adresy se na výstup, tj. pouhý jeden bit, pošle jeden ze vstupních datových binárních hodnot. Použití multiplexorů je poměrně široké. Již na první pohled nás napadne *serializace* původně paralelního tvaru informace do tvaru sériového (místo 2^n vodičů se použije pouze $n+1$ vodičů, protože se přenáší jak adresa, tak i vybraný bit).

To však není vše – multiplexor funguje jako jedna z forem funkce pro přepínání informace a ve své analogové podobě se například používá pro převod několika analogových signálů pomocí jednoho A/D převodníku. Digitální multiplexor se však dá také použít pro jednoduchou realizaci nějaké logické funkce, která by se jinak musela implementovat pomocí jednotlivých hradel, protože na adresové vodiče lze připojit vstupní signály dané funkce a datové vodiče jsou trvale připojeny buď na logickou jedničku nebo nulu (samozřejmě v závislosti na implementované funkci).

Demultiplexor (DMUX, DC) pracuje přesně opačným způsobem než multiplexor, tj. původní jednobitový stav převádí podle hodnoty n-bitové adresy na jeden z 2^n výstupních vodičů. Kromě *deserializace* lze i demultiplexory použít pro realizaci složitějších logických funkcí. Vstupem jsou v tomto případě jak adresové vstupy, tak i jediný datový vstup a na výstup demoltiplexoru se typicky zapojují hradla typu OR.



Značky hradel se do schémat kreslí různými způsoby

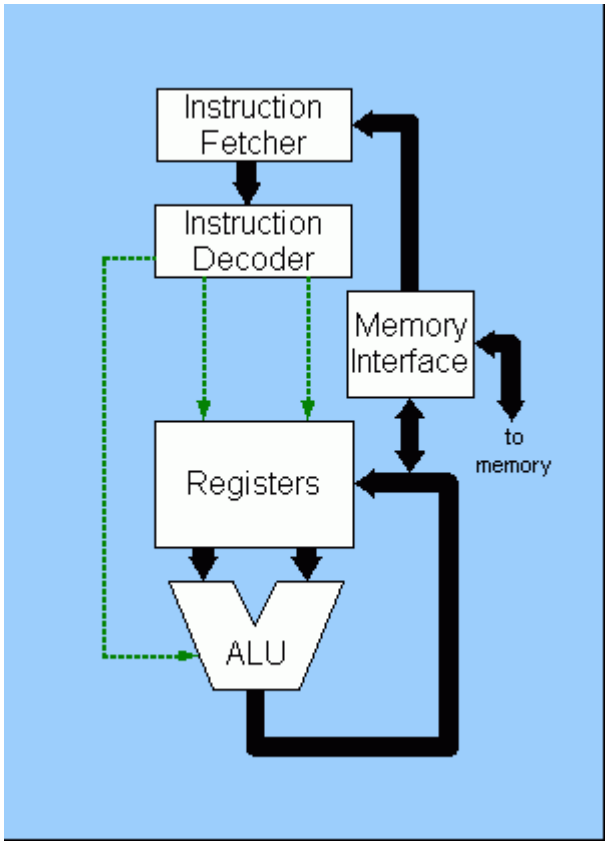
4. Realizace aritmetických operací – základ ALU

V předchozích dvou kapitolách jsme si uvedli některé typické použití kombinačních logických obvodů. Tyto typy sice můžeme v počítačích i přímo mikroprocesorech najít na různých místech, ovšem jeden poměrně složitý kombinační logický obvod se svojí důležitostí poněkud vymyká. Jedná se o aritmeticko-logickou jednotku (zkráceně ALU), která provádí, jak její název ostatně napovídá, výpočet základních aritmetických a logických funkcí.

Princip ALU je poměrně jednoduchý: na její vstup jsou v typické konfiguraci přivedena dvě *n*-bitová čísla a dále *i* vodičů, pomocí kterých se zvolí příslušná operace. Na výstup ALU je po určitém zpoždění posláno *m*-bitové číslo, které odpovídá výsledku zvolené operace. Kromě toho může být na výstupu ALU ještě několik dalších signálů, které se nazývají *příznaky* (*flags*) nastavované v závislosti na výsledku či průběhu výpočtu. Jedním z příznaků může být příznak nulovosti (ten jsme si už uvedli) nebo příznak přetečení.

Jednodušší ALU, které byly použity ve většině osmibitových mikroprocesorů (s výjimkou slavné Motoroly 6809), typicky pracovaly s osmibitovými operandy, tj. dvojicí osmibitových čísel. Z operací, které ALU prováděly, se jednalo o součet dvou čísel (s přenosem i bez přenosu), rozdíl dvou čísel (opět s přenosem i bez přenosu), výpočet dvojkového doplňku čísla, porovnání obou operandů, negace všech bitů prvního operandu, bitový logický součin, bitový logický součet, nonekvivalence a v některých případech i posuny doleva či doprava – mnohdy však poslední dvě funkce vykonával samostatný obvod někdy nazývaný *barrel shifter*.

ALU použité v současných počítačích jsou mnohem složitější, neboť umí například provést vynásobení či vydělení dvou celých čísel. Počet logických hradel a tím i počet tranzistorů nutných pro implementaci těchto funkcí je samozřejmě obrovský. Poznamenejme ještě, že většinou se za ALU nepovažuje jednotka pro provádění operací v pohyblivé řádové čárce, neboli FPU. I když se dnes zpravidla jedná o jednotky umístěné na jednom čipu, stále se považují za dva oddělené funkční bloky, i když rozdíl mezi nimi se pomalu zmenšuje.



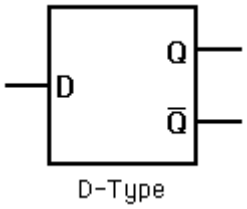
Příklad zapojení ALU v mikroprocesoru

5. Klopné obvody – základ registrů a řadiče

Aritmeticko-logická jednotka sice patří mezi základní stavební bloky mikroprocesorů, kromě ní se však v tomto klíčovém integrovaném obvodu musí nacházet i řadič a sada registrů. Tyto bloky mikroprocesoru se z velké části skládají ze sekvenčních logických obvodů, tj. obvodů, jejichž výstup závisí nejenom na stavu vstupů, ale také na jednom či více předchozích stavech obvodu – sekvenční logický obvod tedy obsahuje nějakou formu paměti. Mezi nejjednodušší sekvenční obvody patří *klopné obvody*, někdy také nazývané *paměťové členy*, protože jeden klopný obvod je možné považovat za jednobitovou paměťovou buňku.

Základním klopným obvodem, ze kterého se odvozují další typy klopných obvodů, je *klopný obvod typu RS*, jenž má dva vstupy označené symboly *R* (reset) a *S* (set) a jeden výstup označený symbolem *Q* (někdy je přítomný i negovaný výstup, to však není podstatné). Označení vstupů odpovídá jejich významu – přivedení logické jedničky na vstup *R* vede k přechodu do stavu 0, přivedení logické jedničky na vstup *S* naopak vede k přechodu obvodu do stavu I. Nastavený stav obvodu zůstane zachován i v případě, že jsou oba vstupy ve stavu logické nuly, tj. právě zde se projeví paměťová funkce obvodu. V případě, že jsou oba vstupy uvedeny do stavu logické jedničky, závisí chování obvodu na jeho vnitřním zapojení a obecně se jedná o zakázaný stav, kterému by se měl tvůrce zapojení vyhnout, což je naznačeno v pravdivostní tabulce klopného obvodu RS:

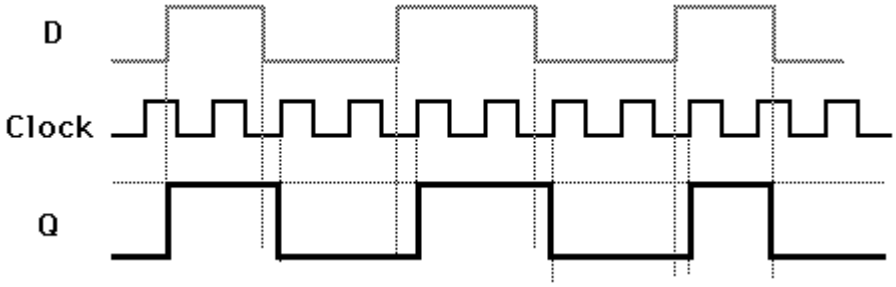
Vstup S	**Vstup R**	**Výstup Q_i**
0	0	Q_{i-1}
I	0	I
0	I	0
I	I	nedefinováno



Schématická značka klopného obvodu typu D bez hodinového vstupu

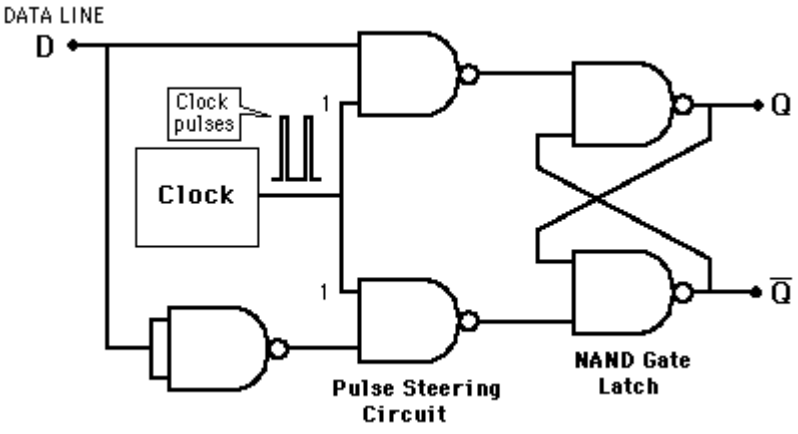
Z klopného obvodu RS lze jednoduchou modifikací, konkrétně přidáním jednoho logického členu negace a dvou členů NAND či NOR, vytvořit *paměťový člen D* neboli *klopný obvod D*. Název je odvozen od slova *data*. Místo signálů *R* a *S* jsou použity dva vstupní signály *D* (data) a *C* (clock). Činnost tohoto obvodu je taková, že při C=I dochází k zapamatování aktuální hodnoty vstupu D, přičemž tato hodnota je zaznamenána na výstup až do dalšího příchodu impulsu (logické jedničky) na vstup C. Jedná se tedy o skutečný paměťový člen, jehož chování lze popsat jak pravdivostní tabulkou, tak i jednoduchým časovým diagramem. Všimněte si, že zde již nedochází k žádnému přechodu obvodu do nedefinovaného stavu:

Vstup D	**Vstup C**	**Výstup Q_i**
0	0	Q_{i-1}
I	0	Q_{i-1}
0	I	0
I	I	I



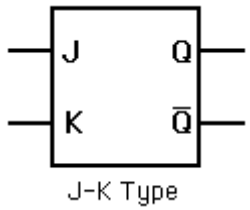
Časový diagram klopného obvodu D

Paralelním spojením několika klopných obvodů typu D (nebo i RS) vznikne *registr*. V něm je možné uschovat *n*-bitovou hodnotu. V mikroprocesorech je obecně umístěno větší množství registrů, typicky bývají minimálně dva registry na vstupech ALU a jeden registr pracuje ve funkci ukazatele na zpracovávanou instrukci. Starší mikroprocesory měly velmi omezenou sadu registrů, protože pro každý registr bylo zapotřebí použít poměrně velké množství hradel a tím i tranzistorů. U novějších procesorů, ve kterých počet tranzistorů jde do desítek milionů, toto omezení již neplatí, spíše jsme omezeni délkou instrukčního slova. Z těchto důvodů se současně používá cca 32 registrů, větší počet bývá rozdělen do takzvaného *registrového okna* (o této technologii si povíme příště).



Zapojení klopného obvodu typu D pomocí pěti dvouvstupových hradel NAND

Existují i další typy klopných obvodů, například *klopný obvod T* (trigger, toggle), nebo *klopný obvod JK*, které se od klopných obvodů RS a D liší především pravdivostní tabulkou a samozřejmě i interním zapojením. Klopný obvod T při hodinovém signálu změní, tj. neguje svůj stav. Klopný obvod JK vznikl vlastně spojením klopného obvodu RS a T – pro všechny stavy, kde platí R^S=0 se obvod chová jako typ RS, pro stav R^S=I dochází k negaci výstupu.

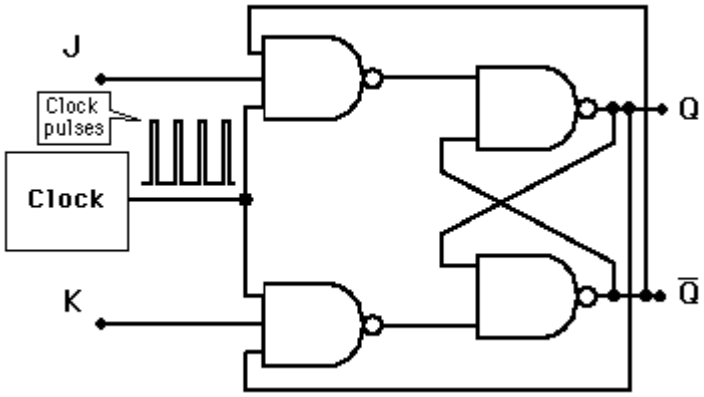


Schématická značka klopného obvodu typu JK bez hodinového vstupu

6. Čítače a posuvné registry

Dalším důležitým prvkem většiny mikroprocesorů je *čítač*. Ten je většinou použit na více místech, typicky pro postupné načítání mikroinstrukcí (v případě, že se jedná o mikroprocesor s mikroinstrukční sadou), přepínání jednotlivých stavů v mikroprocesoru, ale také ve funkci běžného čítače, což je doména jak mikrořadičů, které mají mnohdy více čítačů (counter) a časovačů (timer), ale také mnohých moderních mikroprocesorů. Čítače se také používají pro dělení kmitočtu, což je také důvod k tomu, že mnohé počítače mají své různé části taktované odlišným hodinovým signálem (sběrnice, paměť, procesor) a přitom používají jediný krystal. Čítače lze vytvořit vhodným zapojením několika klopných obvodů za sebe; většinou se jedná o klopné obvody typu T nebo JK. Vlastní přičtení jednotky je provedeno přivedením impulsu na hodinový vstup.

Posuvný registr pracující na podobném principu, který slouží k posunu *n*-bitové informace doprava či doleva, se taktéž v počítačích využívá na více místech. Může se jednat o převod paralelní informace (*n* bitů) na sériovou informaci či naopak, nebo se může jednat o běžný posun celého slova o jeden bit doprava či doleva, což vlastně realizuje operace dělení dvěma a násobení dvěma. S vhodně zvolenou zpětnou vazbou lze pomocí posuvného registru realizovat i jednoduché generátory pseudonáhodných čísel (takto se generoval například zvuk na osmibitových domácích počítačích Atari). Pro jeho realizaci se používají klopné obvody typu D, JK či RS.



Zapojení klopného obvodu typu JK pomocí čtyř hradel NAND

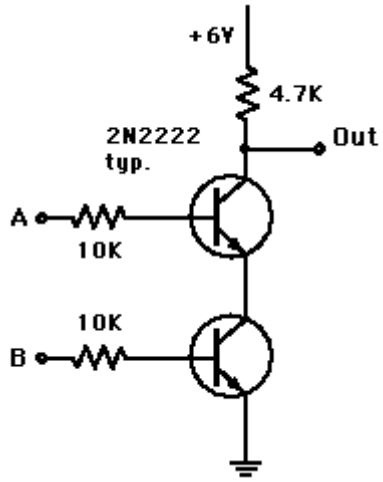
7. Role hodinových signálů (časování)

Se zavedením složitějších kombinačních a zejména pak sekvenčních logických obvodů se ukázala potřeba synchronizace jednotlivých částí celého systému. Důvod je ten, že každý obvod pracuje poněkud odlišnou rychlostí (přesněji řečeno má jiné zpoždění) a proto je synchronizace nutná pro ustálení všech signálů před jejich dalším zpracováním, čímž se do velké míry zamezí vznikům tzv. hazardů a metastabilních stavů. Role čítačů a posuvných registrů je na časování dokonce přímo založená. Z tohoto důvodu je většina mikroprocesorů vybavena vstupem pro hodinový signál nebo

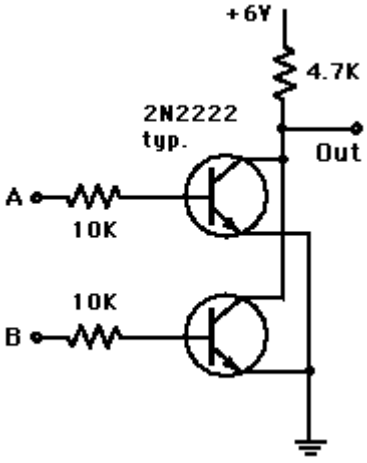
přímo obsahuje obvody pro generování hodinového signálu (potom postačuje připojit vhodný oscilátor, většinou na bázi krystalu). Tento signál je buď použit ve své nezměněné podobě, nebo je násoben popř. dělen tak, aby se pomocí něj daly řídit interní moduly mikroprocesoru. Podrobnosti si ukážeme v další části tohoto seriálu.

8. Technologie logických členů

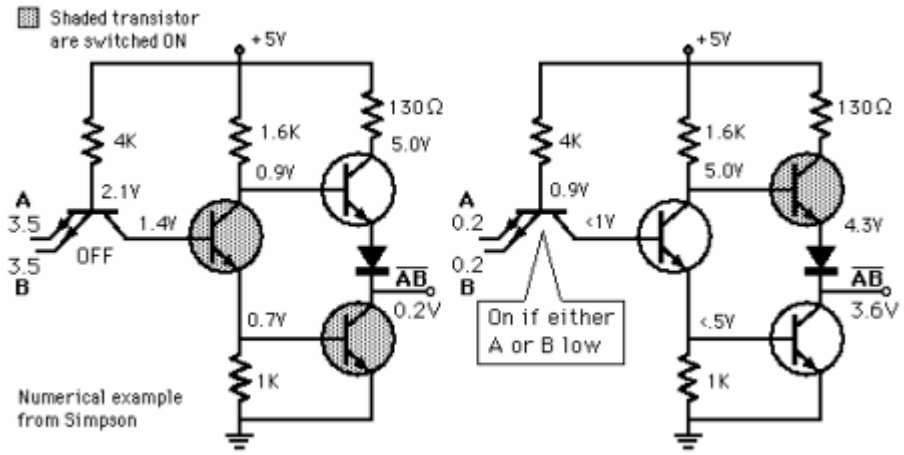
V současných učebnicích se automaticky předpokládá, že hradla a další logické obvody, tj. klopné obvody, čítače, aritmeticko-logické jednotky a další, jsou implementovány v integrovaných obvodech vyrobených nějakou starší či modernější technologií (PMOS, NMOS, CMOS, HMOS atd.). Ve skutečnosti je však možné tyto obvody pracující s binárními hodnotami implementovat například i pomocí relé, diskrétních tranzistorů či dokonce za pomoci optických tranzistorů (ty už byly vytvořeny i když prozatím nenašly při současném stavu technologie větší praktické uplatnění). Bez snahy o úplnost si některé technologie ukažme alespoň na obrázcích s krátkým vysvětlením.



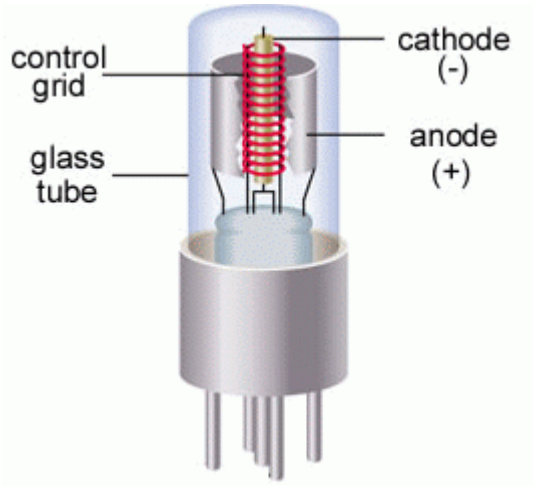
Hradla lze pro účely výuky vytvořit i z diskrétních součástek, i když v tomto případě nebudou mít takové statické a dynamické vlastnosti, jako integrované obvody. Na obrázku je zobrazeno hradlo typu NAND



Příklad implementace hradla typu NOR pomocí diskrétních součástek. Z obou zapojení lze jednoduše odvodit, jak by vypadal invertor.



Skutečná hradla mívají interně složitější podobu, aby se zlepšily jejich statické a dynamické vlastnosti, tj. například se zrychlilo přepínání, dosáhlo se strmější charakteristiky atd. Příklad hradla typu NAND v TTL logice, ve které jsou na vstupu použity tranzistory s více emitory a na výstupu je pro tuto technologii typický *totem*.



Pohled do minulosti – elektronka zvaná *trioda* je vlastně předchůdcem dnešních tranzistorů, ale i CRT obrazovek či aktivních diplejů použitých například ve starších kalkulačkách, videích nebo dalších domácích spotřebičích.

I z elektronek lze vytvořit klopné obvody, i když použité napěťové úrovně, spotřeba, velikost a zpoždění je mnohem vyšší, než v případě použití integrovaných obvodů.