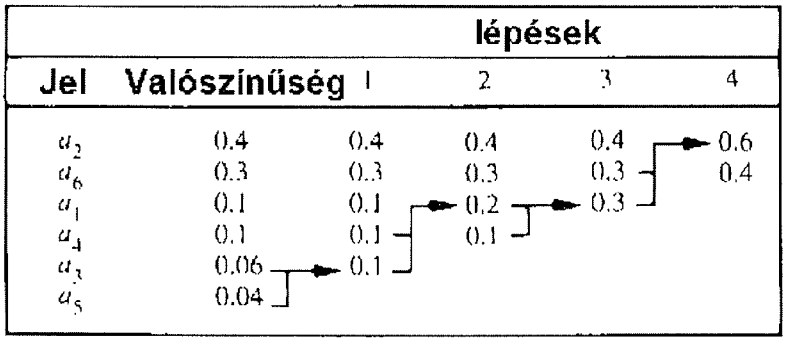
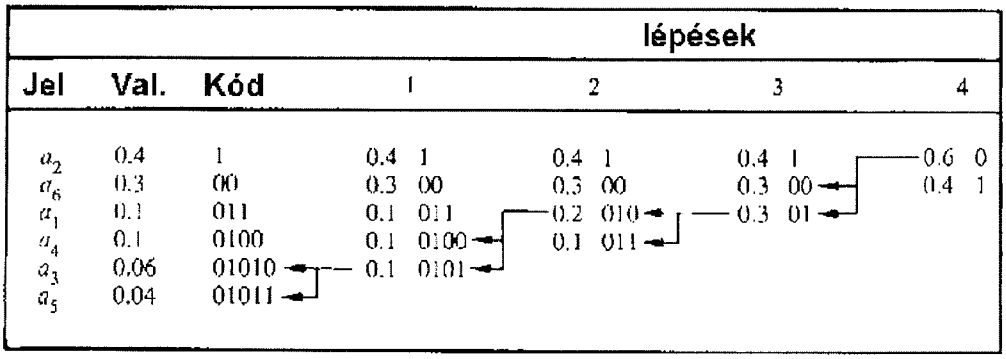



1.16. ábra. A Huffman-kódolás rendezési lépései.




1.17. ábra. A Huffman-kódolás kódolási lépései.

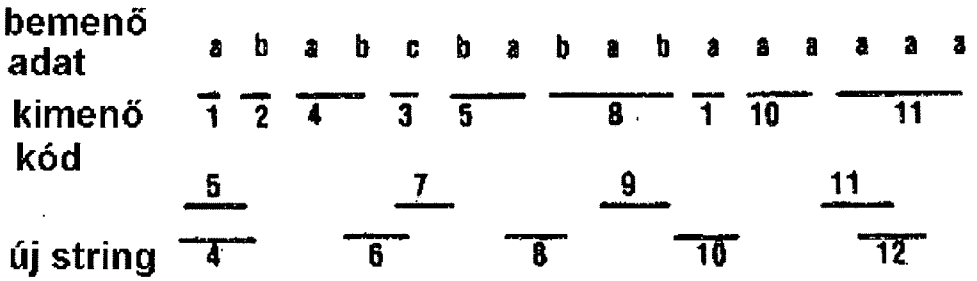
megjelölni az egyes kódok határait.

Az eljárás ugyan jól tud tömöríteni, de hátránya, hogy a bejövő jelek eloszlását ismerni kell (azaz folyamatos on-line tömörítést nem tudnk végezni vele). Ezen úgy segítenek, hogy megvizsgálják sok, hasonló jel eloszlását és ezek alapján felépítenek egy statikus kódtáblát (ennek használata gyorsabb, mivel nem kell elvégezni a rendező-kódoló lépéseket). Statikus kódtábla esetén ráadásul nem kell esetről-esetre rögzíteni (vagy a kommunikációs vonalon átküldeni) a kódtáblát.

Érdekes megemlíteni, hogy a Morse-kód is egyfajta Huffmann-kódolást alkalmaz: az angol ABC leggyakrabban előforduló betűit a legrövidebb Morse-kódok kódolják (nem véletlenül, a kiválasztás gyakorlati tapasztalat alapján történt).

| stringtábla | | szótár | |
|---|---|---|---|
| a | 1 | a | 1 |
| b | 2 | b | 2 |
| c | 3 | c | 3 |
| ab | 4 | 1b | 4 |
| ba | 5 | 2a | 5 |
| abc | 6- | 4c | 6 |
| cb | 7 | 3b | 7 |
| bab | 8 | 5b | 8 |
| baba | 9 | 8a | 9 |
| aa | 10 | 1a | 10 |
| aa | 11 | 10a | 11 |
| aaaa | 12 | 11a | 12 |

1.13. ábra. Példa a szótár felépítésére az LZW eljárásban: a szótár a szaggatott vonal feletti három a, b, c karakterrel lett inicializálva.




1.14. ábra. Az LZW kódok előállítása.

nagyságára $H = 1$-et kapunk. Ezt az információ egységet *bit*-nek (binary unit) hívják. Leggyakoribb többszöröse a *byte*: 1 byte=8 bit. A számítástechnikában használt *szó* (word) architektúra függő: általában a CPU regisztertméretével egyezik meg, ami általában 1-2-4-8 byte (8-16-32-64 bit) széles lehet.

Pl. $n = 8$ különböző, de ugyanannyira valószínű eset esetén $H = -1/8 \log_2 1/8 = 3$, azaz ha 8 esetből 1-et választunk az 3 bit információt hordoz.

Jól látszik a képletből, hogy amennyiben az események nem egyformán valószínűek (vagyis nem zajszerű jelet vizsgálunk), akkor az átlagos információmennyiség kisebb lesz. Például a betűk eloszlása ebben a jegyzetben nem egészen véletlenszerű, hiszen a magyar nyelv gyakoriságeloszlását követi, ráadásul szaknyelv. A redundancia és rendezettség miatt a szöveg entrópiája kisebb a véletlenszerű szövegénél: ezt az effektust pl. tömörítések esetén lehet felhasználni. Jól tömörített adathalmaz nem tartalmaz redundáns részt, az tömörített információ zajszerű. Ugyanígy a nem egyenletes eloszlás egy titkosított csatornán lehetőséget ad a statisztikai alapú visszafejtésre - nem véletlenül a jól titkosított információ entrópiája maximális, az adathalmaz ekkor is zajszerű.

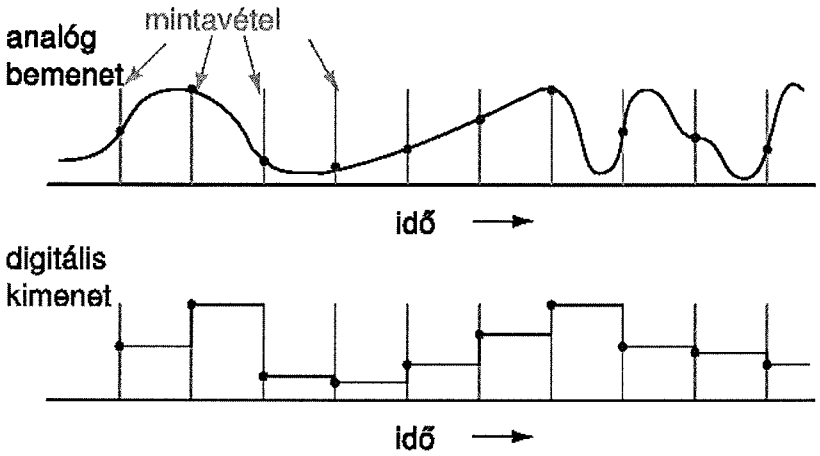
Az effektus jól érzékelhető ez a TV adások fejlődésén: a kezdeti, a világűrbe kisugárzott analóg TV jel galaktikus detektálása (elkülönítése a zajtól) még a földi technikával sem okozott volna gondot (voltak is hasonló földi megfigyelési/detektálási kísérletek, sőt sci-fi szerzők is felhasználták az ötletet...). A mai digitálisan kódolt adások már sokkal zajszerűbbek, a detektálás már sokkal bonyolulabb lenne, a csak előfizetéssel nézhető (titkosított) műholdas csatornákról már nem is beszélve. Egy tökéletesen tömörített TV adást járulékos információ nélkük nem tuduk megkülönböztetni a zajtól.

Pl. egy magyar szövegben a felhasznált betűk, írásjelek és számok száma általában 127-t nem haladja meg, azaz 7 bittel egy betűt kódolni tudunk (az utf-8 kódolás 2 byte-t használ, de ott sokkal több betűkészlet áll rendelkezésre). A szöveg/beszédbeli rövidítések célja is a tömörség, azaz a információfluxus növelése, pl. CMOS kaput mondunk Complementer Metall-Oxid Semiconductor kapuáramkör helyett.

Egy teljesen teleírt A4-es oldal kb. 2000 normál méretű betűt tartalmaz, tehát kb. 2kbyte információt. Hasonlóan egy karakteres 24x80 xterm terminál egy képernyője kb. 1680 byte információt tartalmaz. Egy 200 oldalas könyv (képek nélkül) néhány száz kilobyte információt hordoz (nem minden sor ill. oldal teljes).

Egy számítógép monitor grafikus képe 1024 oszlop x 800 sor x 3 byte (pixelenként 1-1-1 byte a vörös, zöld és kék pont intezitása) $\approx$ 2.45 Mbyte.

Az információmennyiség fogalma mellett gyakran van szükség az információfluxus fogalmára is. Ezzel a másodpercenkénti elküldött/fogadott információmennyiséget jellemezzük. Pl. egy valódi 1920x1080 felbontású, másodpercenként 50 képkockát tartalmazó FullHD videó átvitele 1920*1080 * 3 * 50$\approx$ 311 Myte/s fluxust jelent. Egy normál videó természetesen nem zajszerű, hiszen az egyes pixelek melletti pixelek sokszor hasonlóak (egyszínű területek), és általában nem sokat változik a kép az előzőhöz képest (ritkán van vágás a filmekben, a futballközvetítésekről nem is beszélve!). Ezeket a korrelációkat felhasználva csökkentik különböző tömörítési eljárásokkal kb. a századrészére az információfluxust és a szükséges




Belátható, hogy a bemeneten a jel legmagasabb frekvenciájú összetevőjéből is periódusonként legalább két mintát kell venni (pl. az ilyen frekvenciájú színusz és koszinusz jeleket is csak ekkor lehet szétválasztani!). Ez az ún. mintavételi törvény:

$$2f_{max} \leq f_m \tag{1.1}$$

A mintavételi törvény semmi mást nem mond ki, mint azt, hogy a lebegés (aliasing) jelensége miatt a gyors jelek visszaállitása nem lehetséges kevés pontból. A 1.12 ábrán láthatjuk, hogy a $d$ esetben sérül a mintavételi törvény, az AD átalakító kimenetén megjelenő pontok az $f_m - f_{be}$ különbségi jelet adják vissza.

A törvény megérthető úgy is, ha észrevesszük, hogy a mintavétel során a mérendő $f_{be}$ jelet egy $f_m$ frekvenciájú impulzussorozattal szorozzuk össze, azaz az impulzusokat amplitúdó-moduláljuk (l. XXX fejezet). Az amplitúdó-modulálás állítja elő a frekvenciák összegét és különbségét, amiből a kisebb abszolút értékű frekvencia jelenik meg látszólag a kimeneten.

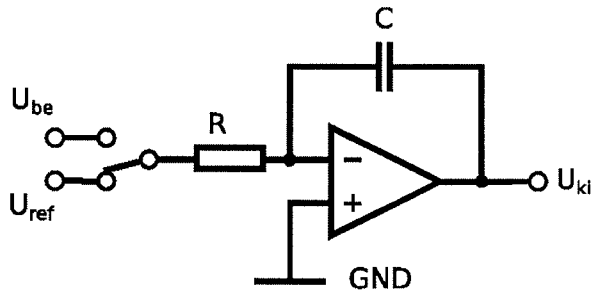
# 1.6. Információ mérése, tömörítés

## 1.6.1. Shannon-féle információmennyiség

Az információmennyiség fogalmát és definicióját Claude E. Shannon fogalmazta meg 1948-ban: e szerint egy adott $i$-ik eseményhez kapcsolódó információmennyiség (bitben mérve) szorosan kapcsolódik annak $p_i$ bekövetkezési valószínűségéhez:

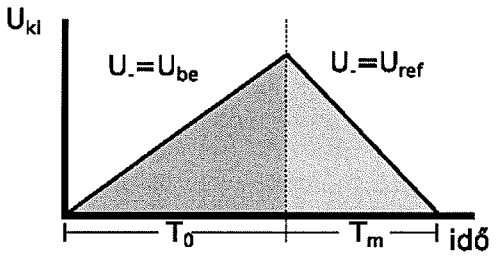
$$I = -\log_2 p_i \tag{1.2}$$

Az információ mértékszáma a "meglepetés" nagyságát tükrözi. Egy megfigyelés, vagy mérés eredménye akkor jelent jelentős információt, ha a kérdéses esemény bekövetkezésének



(a)

1.9. ábra. Kettős meredekségű / kétszeresen integráló / Dual slope AD átalakító integráló tagja. Az $U_{ref}$ feszültség negatív, ha pozitív $U_{in}$ bemeneti feszültséget akarunk mérni.



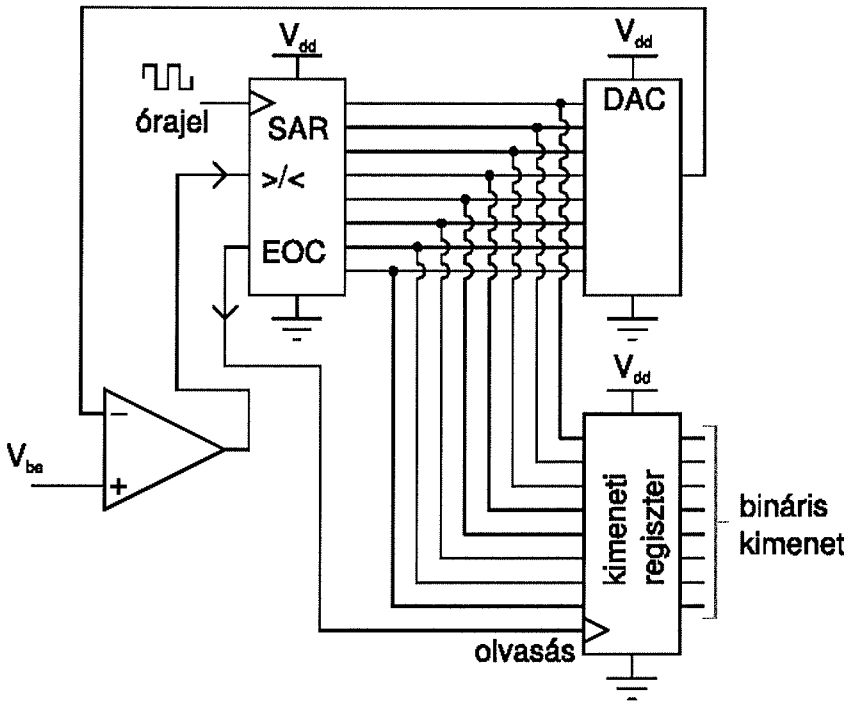
(a)

1.10. ábra. Kétszeresen integráló átalakító (dual slope) időbeli működése. $T_0$ a berendezés által rögzített fix idő, a $T_m$ idő alatt a görbe meredeksége állandó. A $T_0$ és $T_m$ időt általában egy fixfrekvenciájú (pl. kvarc) oszcillátor és egy számláló segítségével mérik.

(itt nem tárgyalt) nagysebességű, pipe-line szervezésű párhuzamos (flash) átalakítóknál (pl. digitális oszcilloszkóp) a mintavételek közti idő akár rövidebb is lehet a konverziós időnél.

Pl. a korábbiakban megismert szukcesszív approximációs ADC nagyon érzékeny a bemenet változására a konverzió alatt (nem lehet úgy találgatni, hogy néha más számra gondol az, akit kérdeznek). Mivel a konverzió legalább néhány órajelig tart, a tényleges konverter elé két további elemet kell illeszteni. A mintavevő áramkör a bemenő jelből nagyon rövid idő alatt mintát vesz, majd a jelnyújtó áramkör ezt az értéket tárolja, megnyújtja konverzió idejére.

A mintavevő áramkör általában diódás vagy tranzisztoros (FET) analóg kapcsolóáramkört jelent, míg a tartó/jelnyújtó áramkör általában valamilyen kondenzátor feltöltésével

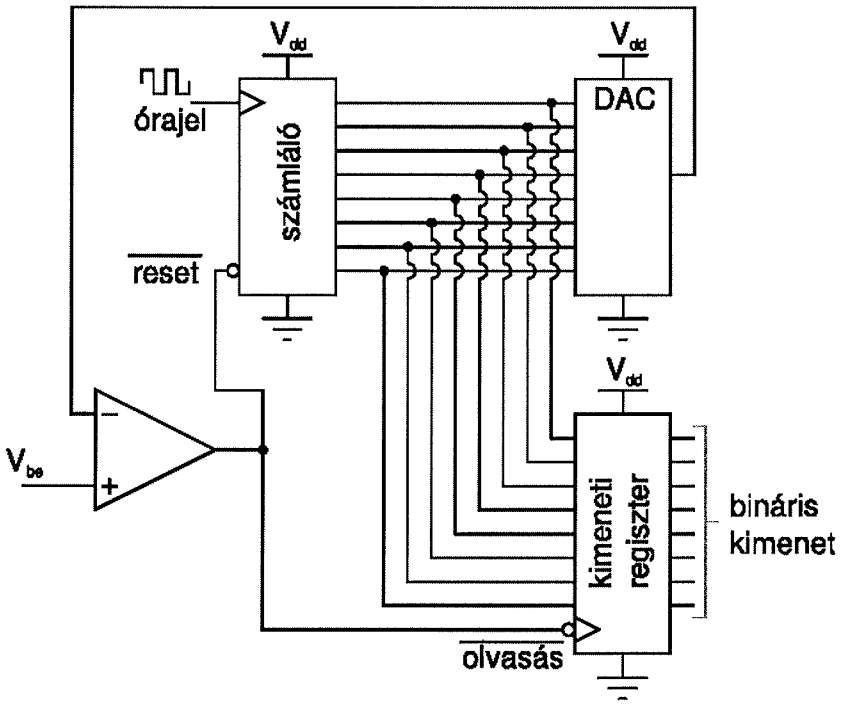



Az ADC nem lineárisan növekvő komparáló feszültséget állít elő, hanem $n$ bit esetén a méréstartományt $2^n$ részre felosztja, és lépésenként megfelezi az éppen vizsgált tartományt.

A számlálás helyett az átalakítás során a SAR (szukcesszív approximációs regiszter) az MSB-től (legnagyobb helyiértékű bittől) indul el, és első lépésként megvizsgálja, hogy a mérendő mennyiség kisebb, vagy nagyobb-e, mint a mérési tartomány fele. Úgy állítja be a bit értékét, hogy ha a minta nagyobb, mint az analóg jel, a vezérlést végző SAR a bitet törli, ha a minta kisebb az analóg jelnél, akkor a bit 1 marad.

A továbbiak során a SAR a bináris számrendszer csökkenő helyértékeinek megfelelő biteknél megismétli az eljárást. Minden lépésnél az előzőhöz hasonlóan a bit felezi a vizsgált tartományt. A komparátor megvizsgálja, hogy a mérendő mennyiség kisebb, vagy nagyobb-e, mint aktuális mérési tartomány fele: a bit értéke ennek megfelelően áll be. Az eljárás valójában egy optimális stratégiával barkochbázó automata.

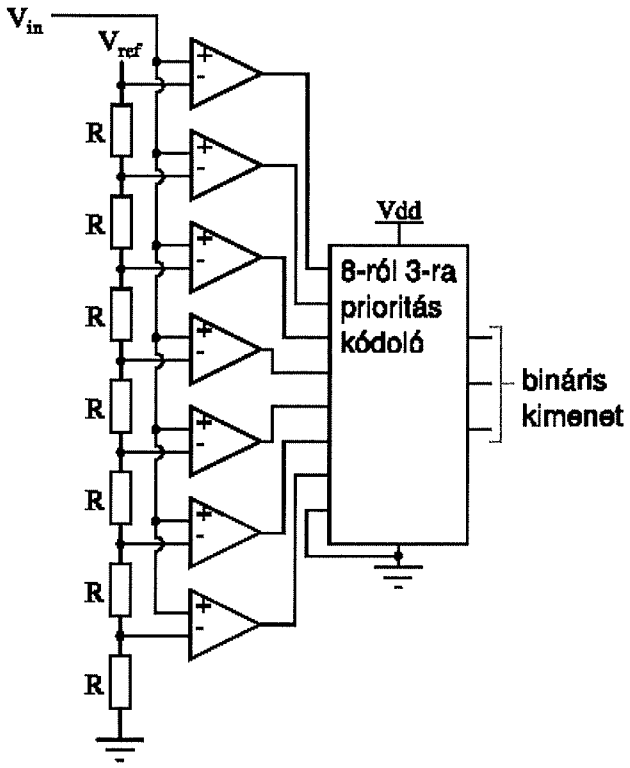
A mérési idő a SAR regiszter méretétől, azaz a mérési pontosságtól függ, mivel az összehasonlítások száma megegyezik a AD biteinek számával. Pl. 8 bit esetén 8 ciklus kell a teljes méréshez:




A számláló kimenetét egy DA konverterre vezetjük amely egy nagypontosságú $U_{ref}$ feszültségforrásból analóg feszültséget állít elő. Ezt egy komparátor áramkör segítségével összehasonlítjuk a bemenő, mérni kívánt ismeretlen analóg jellel $V_{in}$ jellel. Amikor a DAC kimenő feszültsége eléri a $V_{in}$-t, akkor a komparátor átbillen, és ez egyrészt beírja a számláló értékét a kimeneti regiszterbe (pl. D tárolókból építetünk ilyet), másrészt nullázza a számlálót.

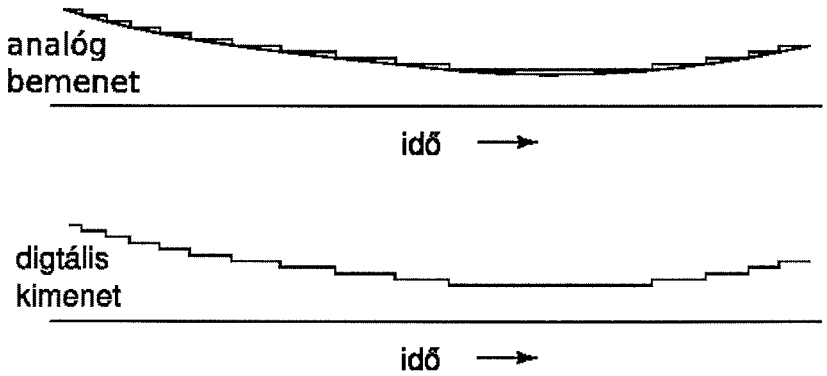
A mérés során tehát a mérendő értékkel arányos számú impulzus került a számlálóba, a mért érték digitális formában rendelkezésre áll.

A konverter nagy hibája, hogy a mérési (átalakítási) idő függ a mérendő feszültségtől (ez kellemetlen, mivel pl. nehezen kiértékelhetővé teszi az ezzel mért idősoros adatokat):

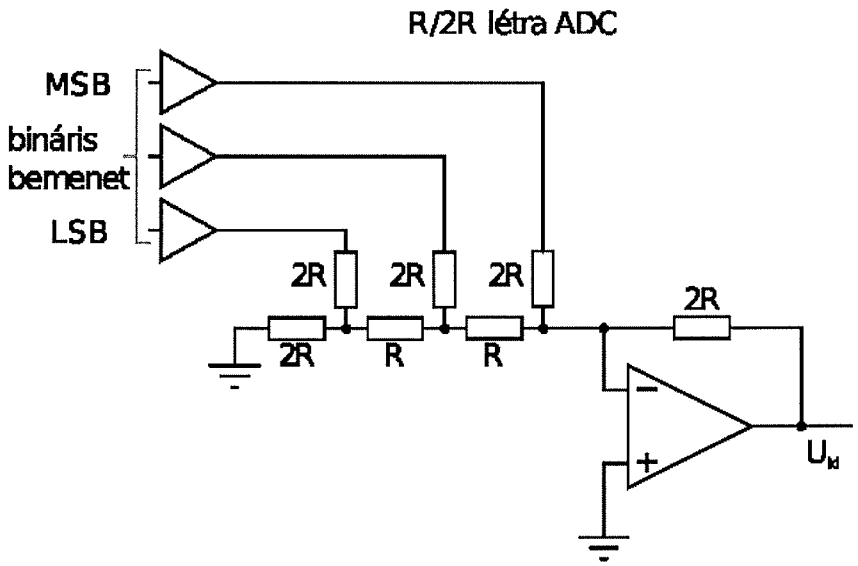



A bemenő jelet komparátorok sorozata hasonlítja össze az ellenálláslánc $V_{ref}$ feszültség alapján előálló referenciafeszültségeivel, majd a komparátorok kimenetét egy kódoló áramkör alakítja át bináris jellé (ez a kimentén megadja a legmagasabb logikai 1-et tartalmazó bemenet bináris címét).

Az ábrán látható, hogy az analóg jel változása hogyan változtatja a kimenő bináris értéket:




A kódoló áramkör felépíthető kizáró-vagy kapuk és diódás logika felhasználásával:




Látható módon a műveleti erősítő bemenetén levő létraáramkör összegezi a feszültségeket - ez könnyen igazolható, ha egy adott bemeneti konfiguráció esetén felírjuk és megoldjuk a Kirchhoff-egyenleteket (l. XXX fejezet). A létraáramkör bővítésével növelhető a bitszám, de természetesen itt is limitáló tényező a kapuk kimenetének pontossága.

### 1.5.4. Analóg-digitál konverterek

Az AD konverter az analóg jeltartományba eső amplitúdó-értékeket alakítja át digitális értékekké. Az eszközök nagyobb részében az analóg jeltartomány általában csak pozitív (vagy csak negatív) lehet. A jeltartomány egyik széle általában a nulla, a méréshatár szélét végértéknek (Full Scale, FS) jelölik, ez általában az alkalmazott referencia-feszültségtől függ.

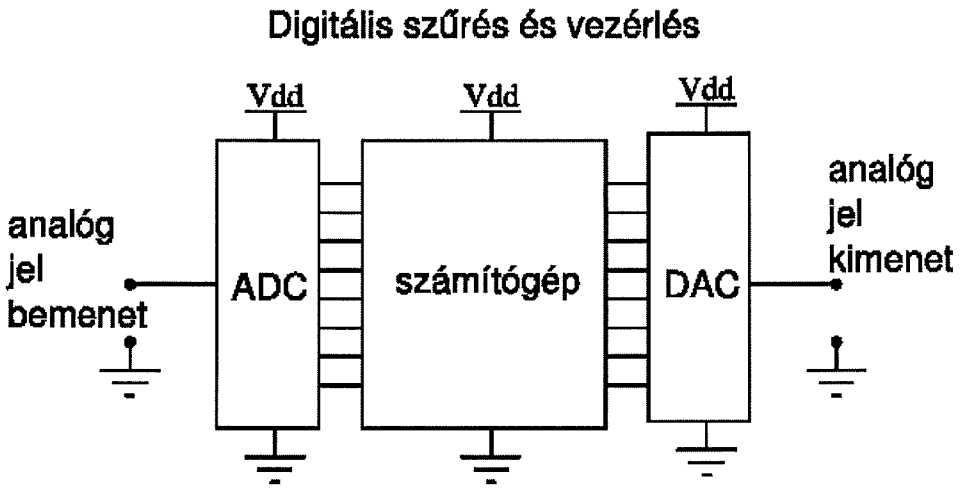
Az átalakítók általában lineáris karakterisztikájúak, a kvantáló átalakítási konstansát ill. karakterisztikájának meredekségét a a felbontóképességgel adják meg.

A bipoláris pozitív-negatív átalakítók általában valamilyen szinteltolást alkalmaznak a bemeneten: a mérhető jeltartomány így közrefogja a nulla értéket, és általában szimmetrikus (FSR = Full Scale Range).

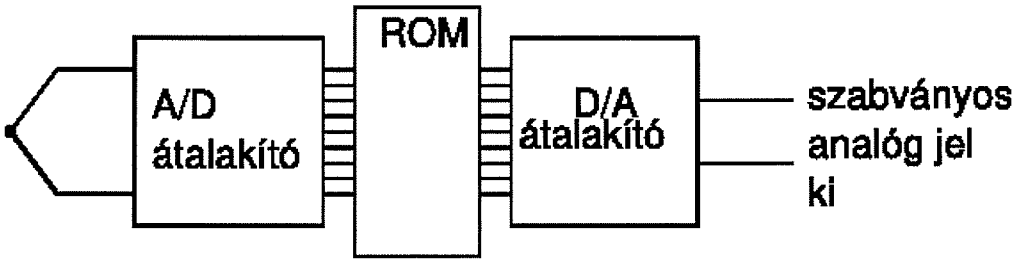
Az AD áramkörök bemenete általában ún. nagyszintű analóg jeleket fogad, mivel az átalakító elektronika relatív hibája nagyobb jelszinteknél kisebb. A szokásos jeltartományok a V-os tartományba esnek, pl.: $-10V \ldots +10V, 0 \ldots +10V, -5V \ldots +5V, 0 \ldots +5V, -1V \ldots +1V$. Kicsi (pl. millivoltos) mérendő jeleknél külön erősítőt kell alkalmazni a bemeneten: itt általában speciális (pl. kicsi offset, kicsi zaj) erősítőket alkalmaznak.

Az ADC kimenetén a digitális adatok megjelenítésére párhuzamos és soros formát egyaránt alkalmaznak. A párhuzamos kimenet egyszerűbbé teszi az illesztést a digitális rendszerekhez, a soros kimenet esetén viszont kisebb kivezetés-számú tokozás alkalmazható,

Általános digitális szűrőként (vagy általános mérő-vezérlőeszközként) egyszerre mindkettőt lehet használni: a szűrési ill. vezérlési feladatokat a számítógépes program végzi:




Erre példa a következő áramkör, amely egy hőelem jelét linearizálja:

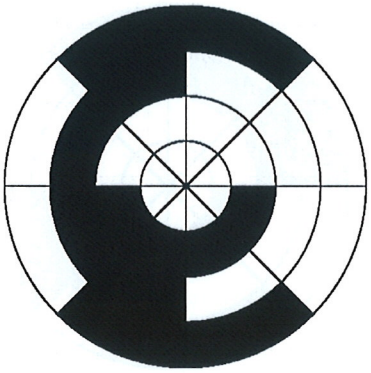



A hőelem két különböző fém közötti termoelektromos feszültség segítségével méri a hőmérsékletet. A hőelem magas hőmérsékletig használható, de a hőmérséklet-feszültség függvény nemlineáris. A linearizálás egyik módja, ha egy ADC átalakítóval digitalizáljuk a jelet, majd ezt egy olyan memória címvezetékére vezetjük, amely már a lineáris értékeket tartalmazza. A memória kimenetére DAC konvertert kötve már a hőmérséklettel arányos feszültséget kapunk. Ezzel az eljárással pl. szabványosítható az érzékelők működése, különböző technológiájú érzékelőket is lehet ugyanott használni.

Érdemes megjegyezni, hogy a mai PC-k nagy részén megtalalálható hangkártya 2 ADC és 2 DAC áramkört tartalmaz (a sztereó működés miatt), és ezek általános célra is használhatók (az ADC és DAC egyidőben is működhet a kártyák nagy részében). A hangkártya hangfrekvenciás (kb. 20Hz-48kHz közötti) jeleket kezel: a hangkártya egyenfeszültségű jellel sajnos nem tud dolgozni.

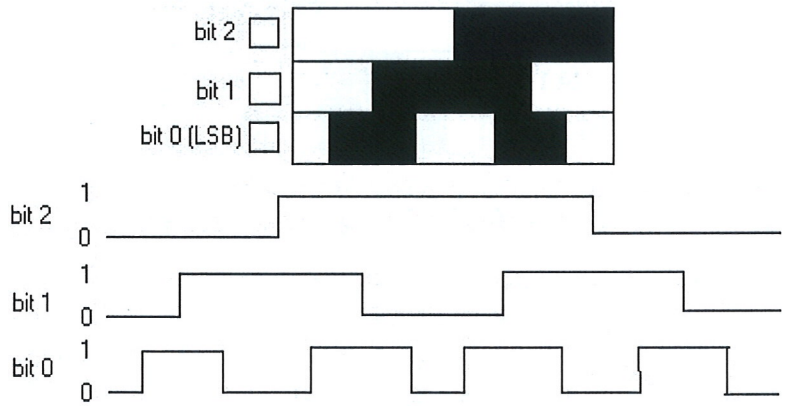
### 1.5.3. Digitális-analóg konverterek

A digitális-analóg konverterek egyik példája az ún. R/$2^n$R DAC. Ez a XXX fejezetben megismert műveleti erősítős összegző áramkör, a bitszámnak megfelelő összegző ellenállással.



(a)

1.6. ábra. Gray kódolású kódtárcsa.




(a)

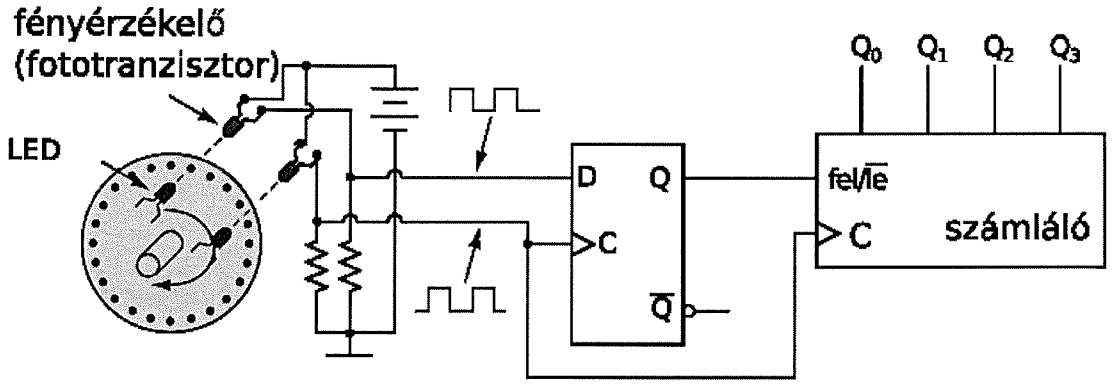
1.7. ábra. Gray kódolás működési diagramja.

adja ki (ezek között akár ugrálhat is kimenet), más érték még véletlenül sem jelenik meg.

A kódoló kimenetét természetes a mérőberendezésnek (számítógépnek) át kell alakítania (kódolnia) a szokásos bináris vagy decimális skálára, ha további műveleteket akarunk vele végezni.

Ez az áramkör az alapja az összes mechanikus forgásérzékelőnek (pl. mechanikus egér, robotok, körbeforgatható szabályozógombok az szórakoztatóelektronikai berendezésken, stb.). Elegendő pénzért akár 14 bites Gray kódolót (16384 állapot, $\approx 0.022$ fok lépésenként) is vehetünk... A tárcsa „kiegyenesítésével" olyan kódszalagot is létre hozhatunk, amellyel lineáris elmozdulást tudunk mérni, pl. egy robot számára.

Lehet meg kellene említeni, hogy párhuzamos adatátvitelnél a Gray-kód jelfeldolgozás szempontból is jobb lehet (egy szervezési ciklusban 1 bit változik).




(a)

1.3. ábra. Kódtárcsás mérőátalakító felépítése.

Az impulzusokat megszámlálva közvetlenül, digitálisan mérhetjük az elfordulás mértéket a számlálóval, ha az egyik érzékelő bemenetét az órajel bemenetre vezetjük rá. Az egyedül probléma az, hogy a tárcsa ellenkező irányú mozgása is ugyanolyan jelet ad.

Amennyiben az érzékelők mechanikai elrendezése olyan, hogy a két LED/fototranzisztor pár kimenete 90 fokos fázissal el vannak tolva egymáshoz képest, akkor a fázis detektálásával meg tudjuk különböztetni a forgásirányt. Ilyen fázisdetektort pl. egy D típusú flip-flop felhasználásával készíthetünk.

Amikor a tárcsa az óramutató járásának megfelelően mozog, akkor a $D$ bemenő négyszögjel megelőzi a $C$ órajelet, azaz a $D$ értéke 1, amikor a $C$ a 0-1 átmenetet mutatja, azaz 1-re állítja a D kimenetét. Fordított irány esetén a $D$ 0 értéket vesz fel akkor, amikor a $C$ órajel 0-1 irányba vált, azaz törli a D kimentét.

A D tároló $Q$ kimenetével a számláló fel/le bemenetét vezéreljük. A számláló így már helyesen működik, számértéke mutatja a tárcsa pillanatnyi állapotát. A tárcsa „kiegyenesítésével" egyszerű kódszalagot is létrehozhatunk: pl. tintás nyomtatók nyomtatófejének pozícióját szokták így detektálni.

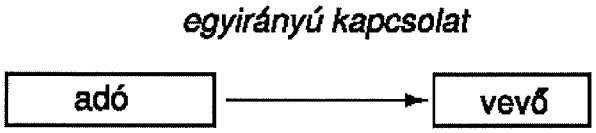
Ennek az elmozdulásérzékelőnek egy nagy hibája van: nem tudja az abszolút pozíciót megállapítani, a számláló a bekapcsoláskor 0 értékről indul (a nyomtató esetében a nullapontot egy extra szenzor biztosítja). Ezen segít a 1.4 ábrán látható elrendezés: a tárcsa nyolc részre osztja a kört, a fekete-fehér területek pedig binárisan kódolják az egyes szektorokat. Minden ívhez egy LED/fényérzékelő párt téve a kimeneten a mérendő értéknek megfelelő digitális érték jelenik meg, ahogy ezt a 1.5 ábrán is láthatjuk.

A kódtárcsának ebben az elrendezésben azonban van egy nagy hibája: a kimenete nem robosztus. Legyen a kimeneten a 000 érték, és forogjon visszafele a tárcsa, azaz következő érték legyen 111. A tárcsa kialakítása és egyéb mechanikai pontatlanságok miatt az átmenet nem időben biztos, hogy nem egyszerre történik a 3 biten, lehet hogy pl. 111-011-

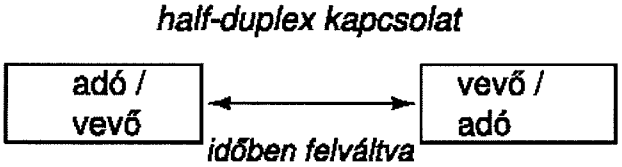
számítógépek SATA csatlakozója ellenpélda: ez nagysebességű soros adatátvitelt használ a diszk elérésére, ami gyorsabb a párhuzamos IDE kapcsolatnál - itt a zavarvédettség kompenzálja a kevesebb vezetéket.

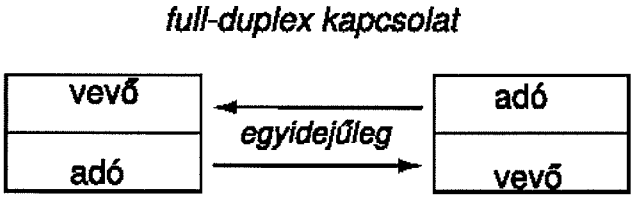
A kommunikációs szabványok szabványosítják a csatlakozókat, azok lábkiosztását, megadják a jelek feszültségszintjeit és időbeli paramétereit, végül szabványosítják az adatátvitel lefolytatásának módját (kapcsolat kiépítés, vezérlés, bontás).

Az adatok átvitele lehet szimplex (egyirányú):

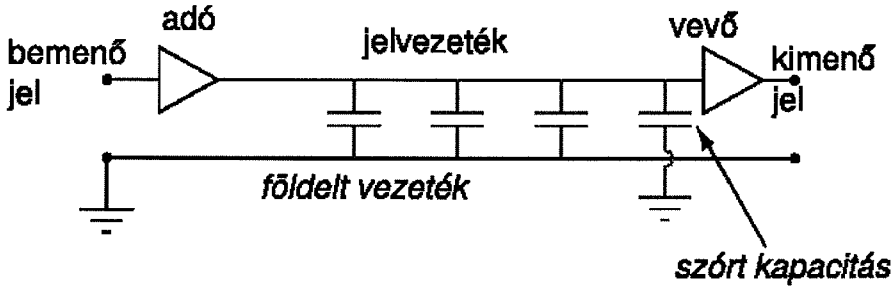



A duplex (kétirányú) átvitel lehet half-duplex, amikor időben felváltva kommunikál a két oldal, vagy full duplex, amikor mindkét irányba egyidejűleg mennek az adatok:




Nagyobb távolságok esetén a soros kommunikációban problémát jelentent a vezeték és a földelés (árnyékolás) közötti szórt kapacitás:




A jelvezeték véges ellenállása és a szórt kapacitás $RC$ aluláteresztőként működik hosszú vezetéknél, ami a fel- és lefutási idők miatt korlátozza a maximális sebességet.

A szórt kapacitásokat az ún. *differenciális* vonali meghajtással lehet hatékonyan csökkenteni, ekkor a két vonal ellenkező amplitúdóval kapja ugyanazt a jelet:

A rendszer működését a CONTROL UNIT vezérli, a tárolóknál már megismert órajeles eljárással. A kiinduló program és adatok (amit a gép indulása előtt be kell tölteni) az írható/olvasható memóriában találhatók. A memóriába van elhelyezve a külvilággal kapcsolatot tartó IN és OUT cella is (ezek pl. billenyű beolvasást és karakter kiírást végeznek).

A processzor a PC (program counter) regiszter által meghatározott memóriacellából beolvassa (fetch fázis) az *utasítás+kapcsolódó cím* adatot az IR (instruction register) utasítás regiszterbe. Az utasítást a következő ütemben az ALU hajtja végre (execute fázis) az A és B regiszterek között.

Az A és B regiszterek tartalmát feltölthetjük a memóriából ill. tárolhatjuk a tartalmat a memóriába. Az A regiszter tartalmát az RA regiszterbe is áttölthetjük.

Az ALU művelet után a processzor a PC értékét vagy az IR regiszter címet tartalmazó része, vagy az ALU által beállított RA regiszter alapján állítja be (ezt egy multiplexer választja ki, amit az utasítás egyik bitje vezérel), így lehetőség van ugrani is a memóriában.

A Neumann elvű számítógép a memóriában nem csak az adatokat, de a programot is tartalmazza: az elv sikerét a napjainkat sűrűn átszövő, erre épülő számítástechnika is mutatja.

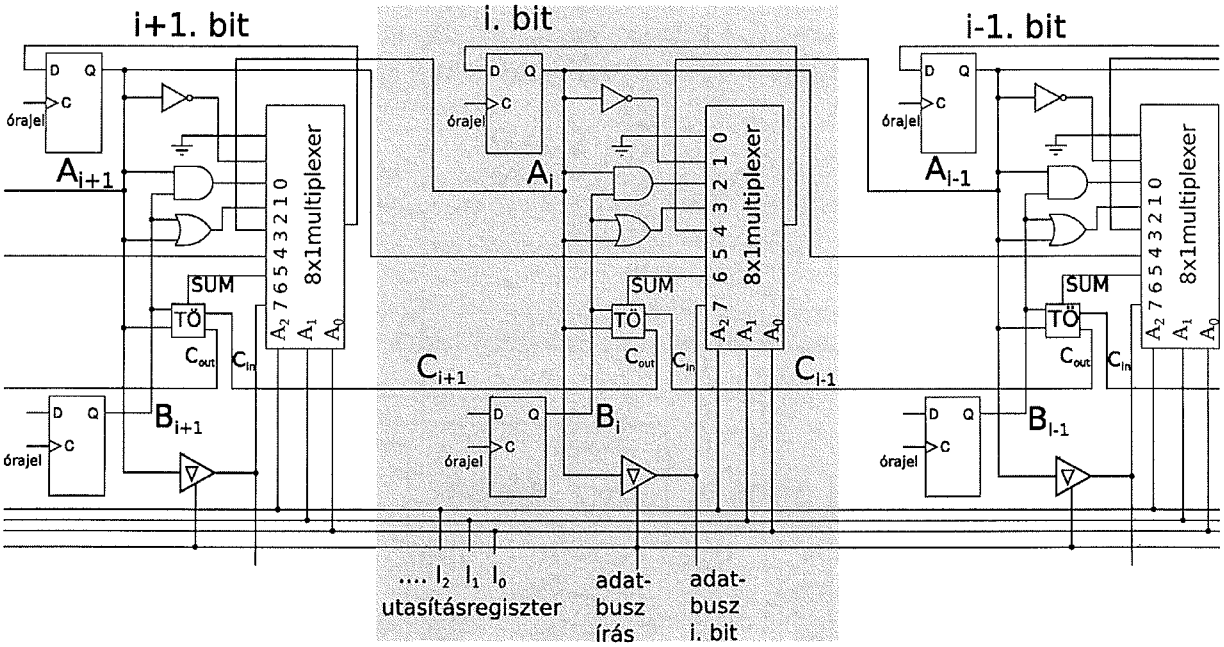
Végezetül nézzük meg az egyik első mikro processzor, a 4 bites 4004 blokkvázlatát, ahol jól felismerhetők az egyes építőelemek:

látható volt, hogy az egyszerűbb FSM rendszer kombinációs hálózatának megtervezése bonyolult lehet, éppen ezért megpróbáltak általános célú hálózatot tervezni.

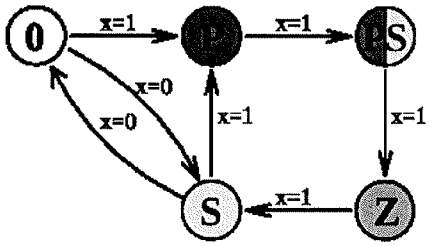
A XXX fejezetben megismerkedtünk az összeadó áramkörrel, ami segítségével már két számot össze tudunk adni. Kivonni is tudunk, ha egy számot negálni tudunk: pl. 4 bites tárolásnál $2_{10} = 0010$, ezért érdemes a $-2_{10} =: 1110$ definíciót választani (figyelem, az első bit ekkor az előjel lesz!): így $2_{10} + (-2_{10}) = 0 = (1)0000$, mivel az összeadás túlcsordul ezért a 4 bites szóban csak 0000 marad. Szorozni és osztani is tudunk a bináris számok bitenként jobbra-balra eltolásával, az osztást pedig visszavezethetjük kivonásra.

Érdemes a műveleteket operandusokon végezni: pl. két memóriarészt kinevezhetünk regiszternek, és építhetünk egy olyan logikai áramört, ami kiválasztható módon különböző műveleteket hajthat végre a regiszterek tartalmán. Az ilyen logikai áramkört aritmetikai-logikai egységnek (Arithmetic logic unit, ALU) nevezzük.

Példaként nézzük egy ALU 1 bites szeletét a következő ábrán:

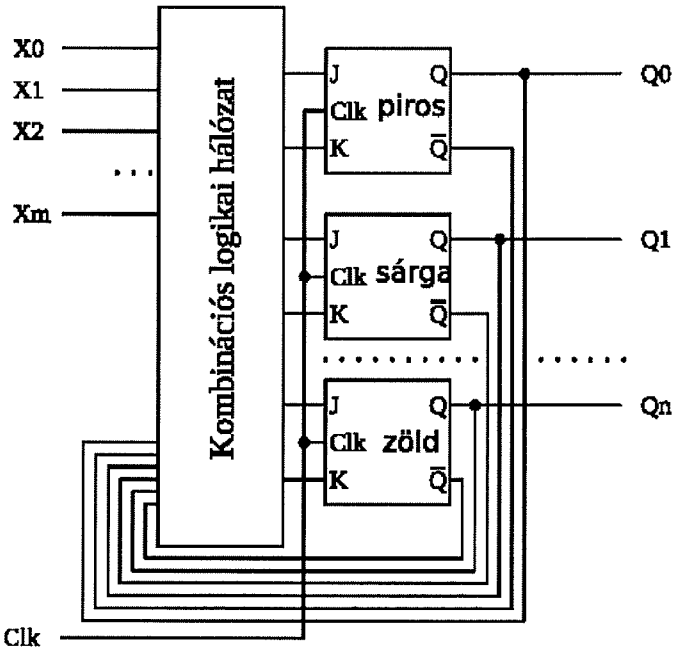



Itt az $A_i$ és $B_i$ bitek az $A$ és a $B$ regiszterek $i$-edik bitjei. Minden művelet végrehajtódik, de az utasításregiszter $I_2 I_1 I_0$ vonalai a multiplexeren keresztül kiválasztják azt a műveletet ill. kimeneti bitet, ami a órajel váltásakor bekerül az $A_i$ regiszterbe. Az egység pl. a következő műveleteket tudja végrehajtani (ezek sorban megfelelnek a multiplexer bemeneteinek):




A fizikai megvalósításhoz legyegyszerűbben pl. JK flip-flopokat használhatunk, amelyek bemenetét kombinációs logikai áramkör vezérli.

Használjunk minden lámpához egy JK-t (a rendszernek 5 állapota van, ezért úgyis legalább 3 bites tárolót kell alkalmazni), és vezessük vissza a $\overline{Q}$ vonalakat is a kombinációs áramkörre, ekkor egyszerűbb lesz a visszacsatolás:




Az $X_0 ... X_m$ vezérlő vonalakból csak az $X_0$-t használjuk.

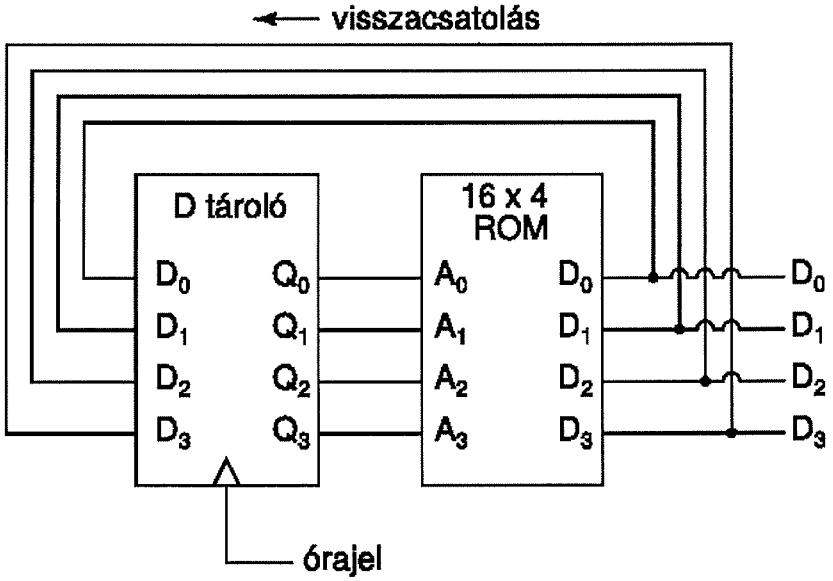
Ezek után elkészíthetjük a rendszer igazságtábláját, ami megmondja, hogy az $n$-edik állapotból az $X$ függvényében milyen $n+1$-edik állapot következik. Ez valójában az a „programozás" első lépése:

egy sor kiválasztása esetén párhuzamosan az összes oszlop aktivizálható.

### 1.4.4. Véges állapotú automata

Az álvéletlen generátor jól mutatja, hogy bonyolult áramköröket lehet építeni viszonylag egyszerű kapcsolással, ha visszacsatoljuk a rendszert (a bonyolult áramkör „definíciója": működése csak nehezen (vagy nem!) található ki, sokáig kell a működő áramkört megfigyelni, hogy egyáltalán tippelhessünk a működési elvére, felépítésére).

Például az álvéletlen generátort általános formában is megépíthetjük:




Itt egy 4 bites D tároló felhasználásával megcímzünk egy 16x4 bites ROMot, amely az álvéletlen generátor adatait tartalmazza. A ROMba a számokat ügyesen keverve kell beleírni (mivel minden állapot előfordul, ezért a rendszer állapotán végiglépünk az álvéletlen generátor generálta számlista sorrendjében, ezáltal az egyes cellák értéket meg tudjuk határozni).

Azt a rendszert, amikor egy tároló rögzíti a rendszer állapotát, és (az esetleges bemenetekkel együtt) egyértelműen meghatározza a következő állapotot, *véges állapotú automatának* (Finite State Machine, FSM) nevezzük.

Az FSM lehetőségeit tovább bővíthetjük, ha 8 bitre növeljük a D tároló és a memória címvezetékének méretét, és bemeneteket helyezünk el:

kombináció a lehetséges 89-ből a (32, 31, 30, 10), (32, 31, 29, 1) vagy (32, 31, 26, 18). Érdekes, hogy minden fokozatszámnál létezik legalább egy megoldása a feladatnak.

Mivel a rendszer minden bináris értéken áthalad, ezért speciális számlálónak is tekintető. Az egyes bitek ugyanannyiszor vesznek fel 0 vagy 1 értéket, látszólag véletlenszerűen (de mégiscsak egy hosszú ciklus szerint): az áramkör a *kvázi-véletlenszám* generátorok alappéldája. A kimenet nem igazi véletlen, mivel egy determinisztikus algoritmussal állt elő, ráadásul periódikusan ismétlődik. Az is látható, hogy a kimenetben maximum a fokozatok számával egyező számú 1 lehet egymás után - a valódi véketlen sorozatnál akármilyen hosszú 1-es osrozat lehet.

Érdemes megjegyezni, hogy a számítógépekben alapesetben használt RANDOM, RND, rand(), rnd() stb. véletlenszám-függvények szoftveresen megvalósítva ugyanezt az algoritmust használják!

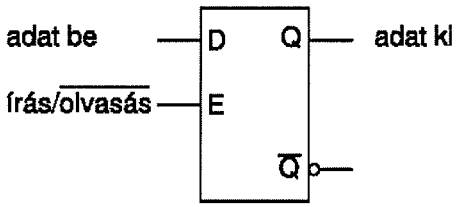
### 1.4.3. Digitális memória áramkörök

A digitális memória bináris (0 és 1) információk, azaz nagy számú bit tárolására és elérésére szolgál. A digitális tárolás előnye, hogy az analóg memóriánál, amilyen például a bakelit hanglemez, ellenállóbb a zajra és az adatvesztésre. A digitális adatokat könnyen lehet tömöríteni és további hibajavító kóddal ellátni (pl. paritás, checksum), és általában könnyen kereshetünk egy adott címen lévő adatot.

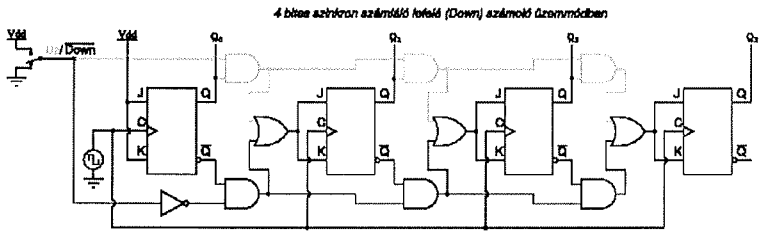
Egy adott információt a tárolás során az adattároló egy adott területén, az ún. címen tárolunk. A cím nem más mint egy bináris szám, ami az adott tárolt érték sorszámát jelenti, a multiplexereknél látott cím analógiájára. Egy adott címen több bitet is tárolhatunk egymással párhuzamosan, pl. 8, 16 vagy 32 bitet, a memória szervezésétől függően.

A számítógépek két fő memóriafajtát, a csak olvasható (Read-Only Memory, ROM) és írható-olvasható, véletlenszerűen elérhető memóriát (Random Access Memory, RAM) használnak. Mindkét memóriafajta esetén létezik az áramkör ki-bekapcsolása után az adatokat felejtő és nem felejtő verzió.

Az írható-olvasható memória logikailag egyik legegyszerűbb formája a D tároló, amely 1 bitet tud tárolni, az alábbi szerint.



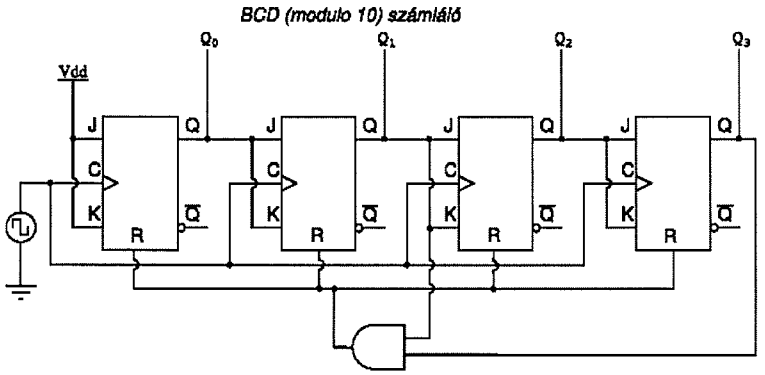

Több ilyen egységet csak akkor tudunk használható rendszerré szervezni, ha a ki tudjuk választani, hogy melyiket akarjuk elérni (ezt nevezzük címnek). A kiegészített tároló egységet az alábbi ábra mutatja. Háromállapotú kimenettel (l. XXX fejezet) az adat bemenetet és az adat kimenetet ugyanahhoz a közös buszvezetékhez köthetjük. Az



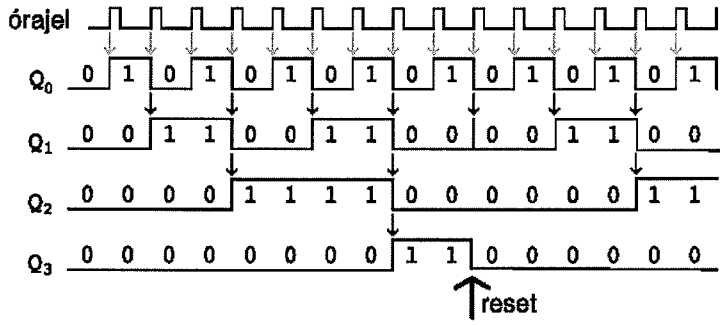
## Modulo N számlálók, álvéletlen generátorok

Az eddig tárgyalt számlálók mind kettes számrendszerben számoltak. Néha szükség van más számrendszerbe számoló számlálóra is: ezeket modulo N számlálónak hívák. A 10-es számrendszer kitüntetett, ha ebben számolunk, akkor a korábban már vizsgált 7 szegmenses kijelzővel könnyen építhetünk számlálót (10-es számlálót gyártanak is).

A 10-es számrendszerben számoláshoz 4 bit kell, de csak a 0-9 állapotokat használjuk: ezt a kódolást BCD (Binary-coded decimal) kódolásnak is nevezi. Egy egyszerű aszinkron számlálót egy ÉS kapuval kiegészítve elérhetjük, hogy a 10-es értéket elérve reset-elje magát (az ÉS kapu kimenete a következő fokozatra vezethető):




Az idődiagramm mutatja, hogy amikor a számláló eléri a 10-et, az ÉS kimenete 1 lesz, a reset vezeték aktivizálódik és a számláló visszaáll az eredeti állapotba:

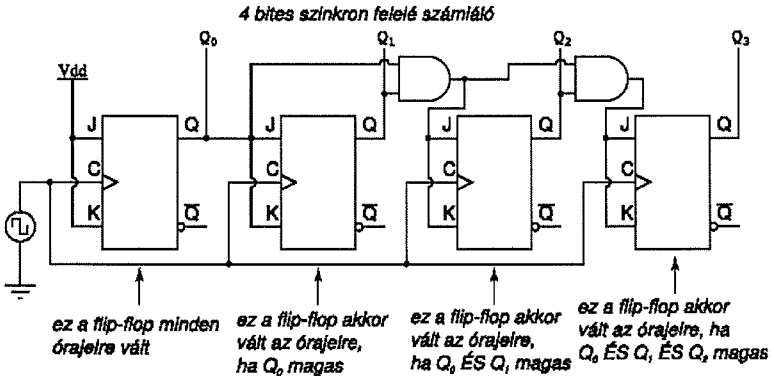



A reset vezeték csak rövid ideig lesz aktív, de vegyük észre, hogy a visszacsatolás miatt ez pontosan annyi idő, amíg a JK flip-flop nullázódik (feltéve, hogy ugyanolyan

1000 állapotokon megy át (decimálisan ez a $7 \rightarrow 6 \rightarrow 4 \rightarrow 0 \rightarrow 8$). Ha egy logikai áramkörrel - ami általában gyorsabb, mint egy JK kapu - szeretnénk detektálni a számláló 0 állapotát, akkor egy rövid ideig hamis jelzést kaphatunk - de ez éppen elég lehet pl. egy RS tároló triggerelésére, vagy reset kiadására! A logikai hazárdok egy tipikus esetével találkozunk tehát, amely által potenciálisan okozott problémát az élvezérlés technikájával, és az órajel elegendően alacsonyra csökkentésével küszöbölhetünk ki.

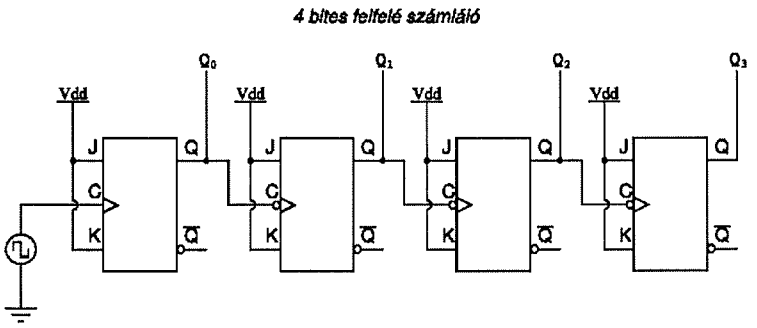
### Szinkron számlálók

Az előző példa kapcsolási rajzát nézve láthatjuk, hogy az elcsúszás oka az, hogy az egyes flip-flopok nem egyidőben kapják az órajelet. Az elcsúszás felgyülemlik, ezért sokáig tart míg a számláló a helyes értéket mutatja – hasznos lenne csökkenteni az effektust, hogy a lehetséges órajelsebességet jelentősen megnövelhessük. A megoldást az jelentheti, hogy minden JK bemenetére egyidőben ugyanazt az órajelet (azaz a számlálni kívánt impulzusokat *szinkronban*) vezetjük, és a J és K bemenet ügyes kapcsolgatásával oldjuk meg a megfelelő ütemű billenést:

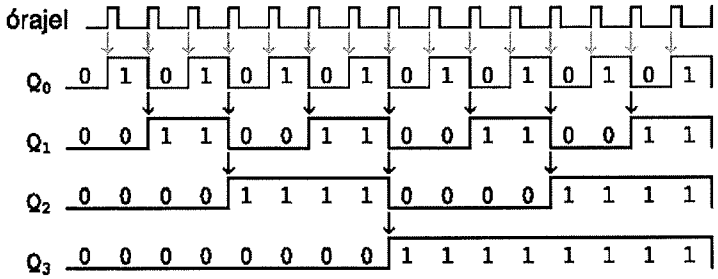



Az eredmény egy 4 bites szinkron felfelé számláló. Minden fokozaton a J és K bemeneten talalható ÉS áramkör engedélyezi az átváltást, ha minden előző fokozat kimenete 1. Az első flip-flop J és K bemenete 1-re van kötve. Megjegyezzük, hogy a MSB (negyedik) flip-flop bemenetén található ÉS kaput kicserélhetjük egy 3 bemenetű ÉS kapura (a bemeneteket a $Q_0Q_1Q_2$ kimenetekre kötve): ekkor az áramkör késleltetése egy kapuval kisebb lesz (mivel nem kell a jelnek átterjednie az első ÉS kapun), gyorsabban tudjuk a számlálót működtetni.

Ez az elvet továbbvihetjük, és hasonlóképpen ÉS kapukkal előállíthatjuk a szinkron lefele számlálót is:

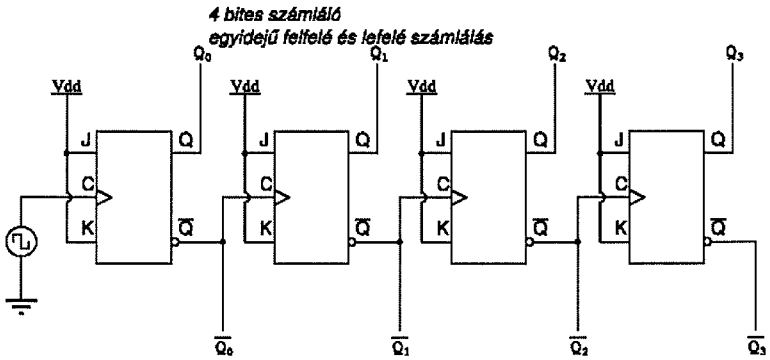



Ez az órajel hatására a pontosan a megkívánt számlálást végzi el, a számokat a $Q_3Q_2Q_1Q_0$ kimeneteken megjelenítve:




Az első flip-flop pozítív élvezérlésű, a többi flip-flop negatív élvezérlésű. Figyeljük meg, hogy az órajel kitöltési tényezője kevesebb, mint 50%, azaz az órajel alakja nem számít a számlálás során.

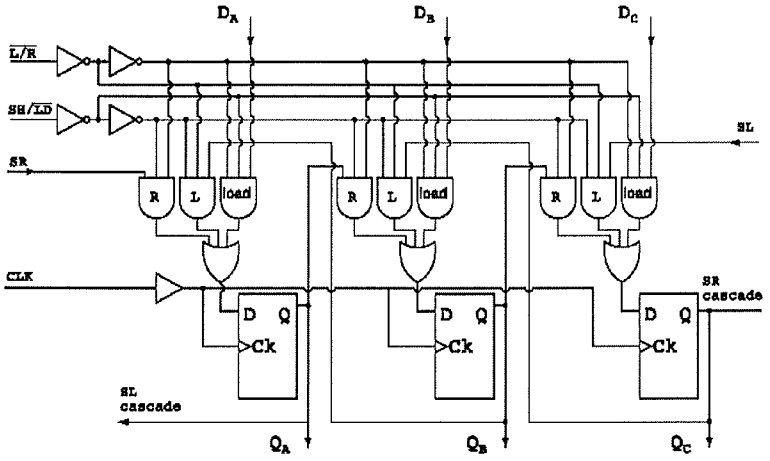
A számláló - ha figyelmesen megnézzük - egyben lefele számlálásra is alkalmas, ha csak pozítív élvezérlésű flip-flopokat használunk, az órajeleket az előző fokozat $\overline{Q}$ kimenetére kötve:




A $Q$ kimenetek előre számlálnak, míg a $\overline{Q}$ kimenetek visszafelé:

előző fokozatból (a legelső fokozat a soros bemenetet olvassa). A $Q$ kimeneten a soros bitszámmal megadott órajel után párhuzamos formában áll elő az adat.

Végül egy általános célú jobbra-balra shiftelő, beírással is rendelkező áramkör 3 bites része:




**3 bites jobbra/balra shift regiszter párhuzamos beolvasással**

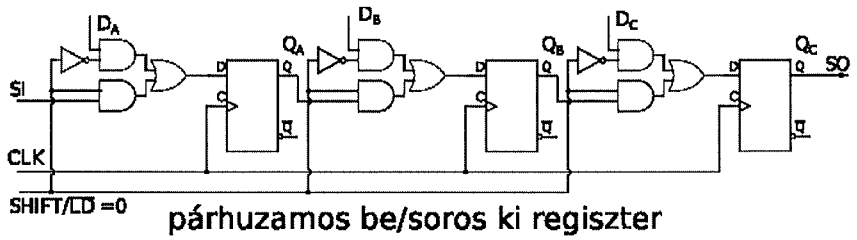
A jobbra-balra eltolást a $\overline{L}/R$ vezeték szabályozza az órajel alatt. A $SH/\overline{LD}$ vezeték a kapcsolódó ÉS kapukkal lehetővé teszi a $D_A D_B D_C$ bemenetekről az adatok párhuzamos beírását. Amikor $SH/\overline{LD} = 0$, és a $R$ és $L$ kapuk le vannak tiltva, valamint a load kapuk engedélyezve vannak, akkor a $D_A D_B D_C$ bemenek a flip-flopok bemenetéről a következő $CLK$ órajelre beolvasódnak, és megjelennek a $Q_A Q_B Q_C$ kimeneten.

## 1.4.2. Bináris számlálók

Bináris számlálásnál azt szeretnénk, hogy a bejövő jel impulzusai hatására pl. egy 4 bites rendszer a

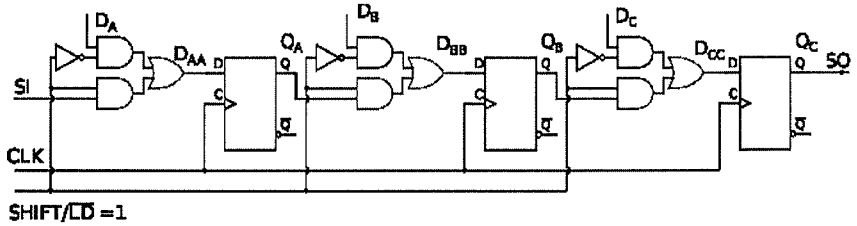
| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $D_0$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| $D_1$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $D_2$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $D_3$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

állapotokon menjen át ciklikusan. Egy adott helyiértéket nézve láthatjuk, hogy az az eggyel kisebb helyiérték változásának felével „oszcillál", azaz csak az eggyel kisebb helyiérték minden második változásánál változik. Ez alapján azt szeretnénk, hogy a számláló idődiagramm a következő legyen:




párhuzamos be/soros ki regiszter
párhuzamos beolvasás üzemmód

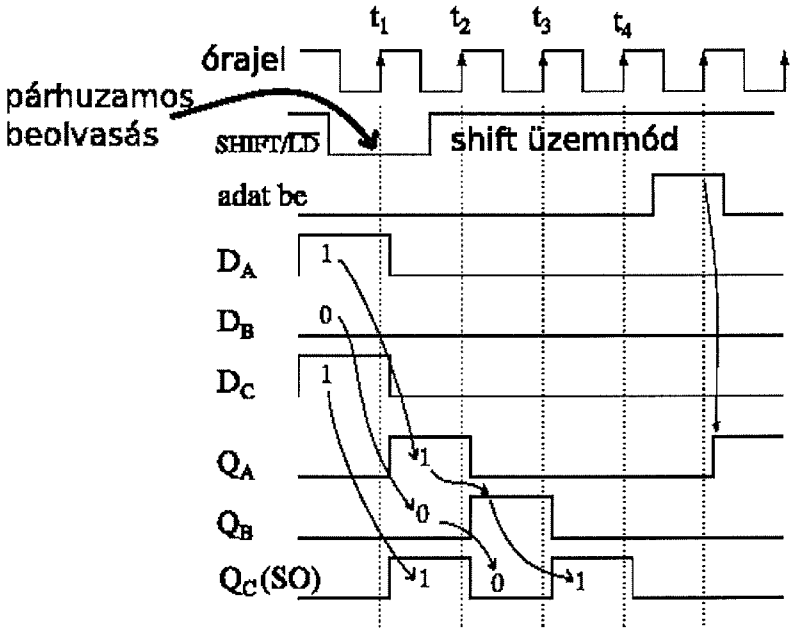
Amikor a $SHIFT/\overline{LD} = 1$, akkor az áramkör normál shift regiszterként működik:




párhuzamos be/soros ki regiszter, shift üzemmód

Az áramkör időbeli diagrammja a következő:

párhuzamos be/soros ki regiszter idődiagramm




Az adatok parallel beolvasása a $D_A, D_B, D_C$ párhuzamos bemenetről a $SHIFT/\overline{LD} = 0$ esetén történik az órajel felfutásakor. Látható, hogy ezután a $SHIFT/\overline{LD} = 1$ állapotban az egyes fokozatok az órajelek felfutásakor léptetik át az adatokat az előző fokozatból (a legelső fokozat a soros bemenetet olvassa). A $Q_C$ ($SO$, serial out) kimeneten a soros bitszámmal megadott órajel után párhuzamos formában áll elő az adat.

| $T$ | $Q_{i-1}$ | $Q_i$ | |
|---|---|---|---|
| 0 | 0 | 0 | nincs változás |
| 0 | 1 | 1 | nincs változás |
| 1 | 0 | 1 | ellenkező értékre vált |
| 1 | 1 | 0 | ellenkező értékre vált |

A kapu kimenete logikai függvénnyel is megadható:
$Q_i = TQ_{i-1} + \bar{T}Q_{i-1}$

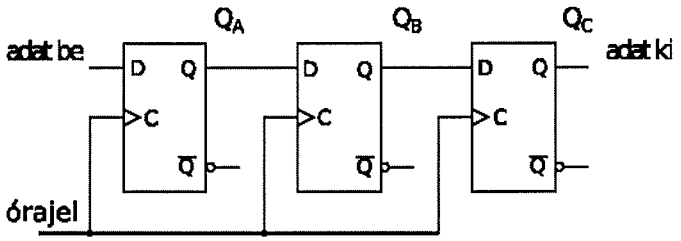
## 1.4. Szekvenciális áramkörök

Szekvenciális áramköröknek (sorrendi hálózatoknak) nevezzük azokat az logikai áramköröket, amelyeknél a rendszer állapota nem csak a pillanatnyi bemenettől, hanem a rendszer korábbi állapotaitól is függ. A szekvenciális áramkörök tehát memóriával, jól meghatározott belső állapotokkal rendelkeznek. A JK és D tárolók tekinthetők a legegyszerűbb ilyen áramköröknek.

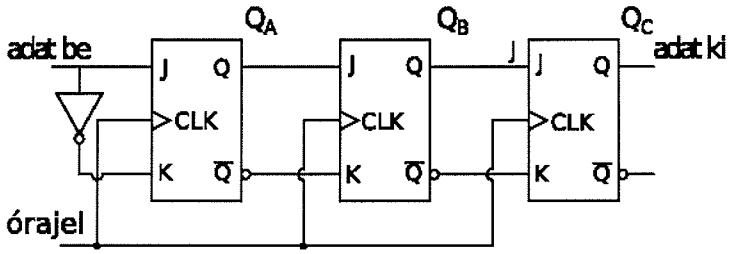
### 1.4.1. Shift regiszterek

A szekvenciális áramköröknél az egyik alapfeladat a JK flip-flopokban tárolt bitek mozgatása, azok rendezett áthelyezése a tárolók között. A tárolt rendezett bitsorozat léptetésére shift regisztereket használunk, ahol a shift angol szó utal az eltolásra. A shift regiszterrel számolhatunk is: a csökkenő helyiérték szerint beírt bináris szám a jobbra/balra eltolás esetén 2-vel szorozható ill. osztható.

Shift regisztert akár D kapukból:



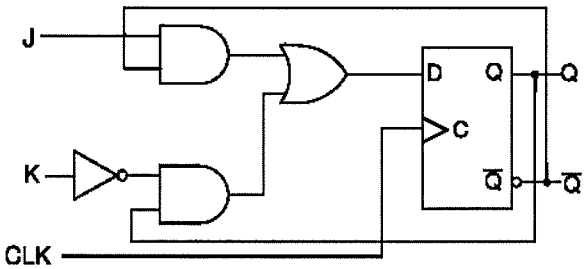

akár JK flip-flop felhasználásával is készíthetünk:



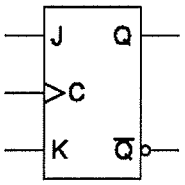
A kapu igazságtáblázata a következő:

| | J | K | Q | $\bar{Q}$ |
|---|---|---|---|---|
| ⎍ | 0 | 0 | tárol | tárol |
| ⎍ | 0 | 1 | 0 | 1 |
| ⎍ | 1 | 0 | 1 | 0 |
| ⎍ | 1 | 1 | vált | vált |
| | 0 | 0 | tárol | tárol |
| | 0 | 1 | tárol | tárol |
| | 1 | 0 | tárol | tárol |
| | 1 | 1 | tárol | tárol |

A kapu kimenete logikai függvénnyel is megadható:
$Q_i = JQ_{i-1}^{-} + \bar{K}Q_{i-1}$
ahol az $i-1$ index az órajel felfutás előtti értékre, az $i$ az órajel felfutása utáni értékre utal. Az élvezérelt JK áramkört a fenti egyenlet alapján egy élvezérelt D tároló segítségével valósíthatjuk meg legegyszerűbben:

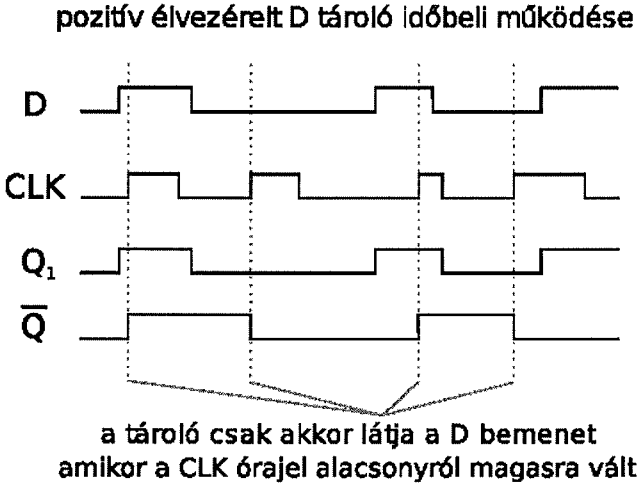



Az élvezérelt JK flip-flop áramköri jelölése a következő:



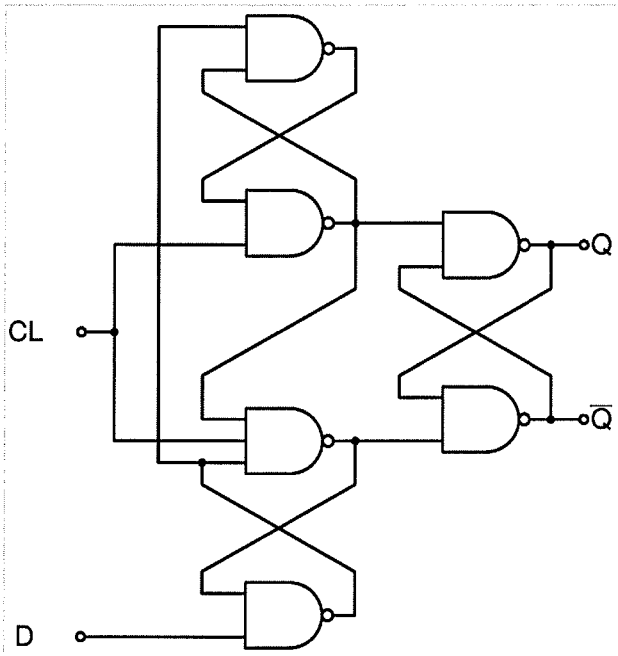

A JK tárolók praktikusan a legfontosabb elemei az összetett digitális áramköri rendszereknek, alap építőkövei a mikroprocesszoros, azaz a modern számítógépes eszközöknek.

A további alfejezetek gondolatmenetei mind az élvezérelt JK áramkörre fognak épülni, ezért nézzük meg a kapcsolás egy lehetséges idődiagramját. Akárcsak az élvezérelt D tárolónál, a szabály nagyon egyszerű és egyértelmű: a $J$ és $K$ értékei az órajel felfutásának pillanatában számítanak csak. A kimenet az igazságtábla szerint alakul: $J = K = 0$-nál a $Q$ kimenet megmarad (tárolás funkció), a $J = 1$ és $K = 0$-nál $Q = 1$ lesz (beírás funkció), $J = 0$ és $K = 0$-nál $Q = 0$ lesz (törlés funkció), $J = K = 1$-nél pedig $Q$ az ellentétére vált (átváltás funkció).

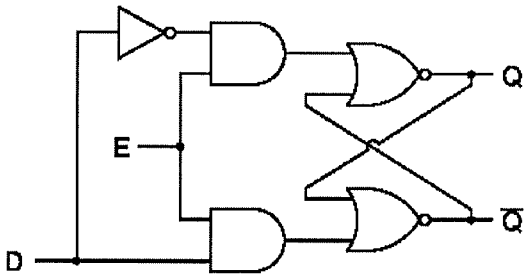



Ezzel az elegáns megoldással sikerült megvalósítani az élvezérlés feladatát: a $D$ bemenet értéke csak az órajel alacsony-magas átmenetének nagyon rövid ideje alatt számít, azaz ami abban a pillanatban a $D$ bemeneten van, az jelenik meg a $Q$ kimeneten.

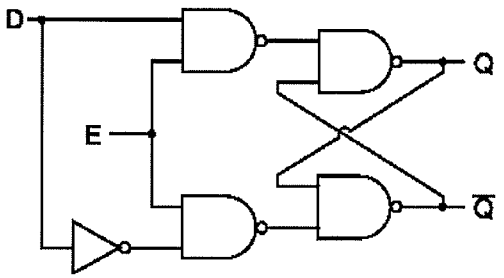
Másik lehetséges, népszerű megoldás az élvezérelt D tároló funkciójának kivitelezésére az alábbi. 3 RS tároló ügyes összekötésével, többszörös visszacsatolással élvezérelt D tároló állítható elő, ahol a bal oldali két RS tároló csak az órajel felfutásának pillanatában nyílik meg egy nagyon rövid (néhány kapukésleltetésnek megfelelő) időre:

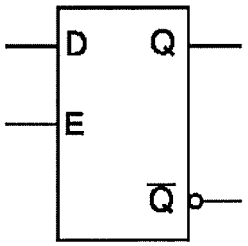



Funkció szempontjából valóban egyforma a két utóbbi megoldás, gyakorlati szempontból az utóbbinak jobbak az időzítési tulajdonságai.




| E | D | Q | $\overline{Q}$ |
|---|---|---|---|
| 0 | 0 | tárol | tárol |
| 0 | 1 | tárol | tárol |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

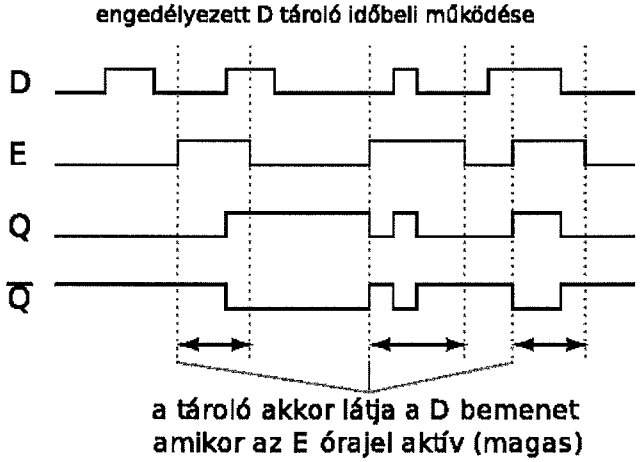


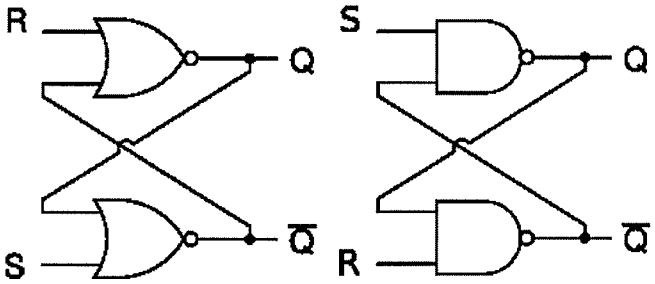




Ha megfigyeljük a kapcsolási rajzot, a $D$ bemenet (és negáltja) egy-egy ÉS kapun keresztül jut az RS tároló megfelelő részbe, az ÉS kapuk másik bemenete pedig az $E$ bemenet. Az RS tároló felé tehát a $D$ bemenet csak akkor megy tovább, ha az $E$ magas, azaz, ha $E$ engedélyezi a $D$ adat beírását. Ha $E$ alacsony, akkor $D$ értékétől függetlenül a tárolt értéket kapjuk a $Q$ kimeneten.

A fenti értelemben D tároló 1 bites memóriacellának tekinthető: a bitet D-vel írjuk be, ha $E$ magas.

Nézzük meg az engedélyezett D tároló idődiagramját. Az engedélyezett D tároló működése közben a kimenet a $D$ bemenetet követi, amikor az E engedélyező bemenet aktív - attól függetlenül, hogy ez mennyi ideig van így:






| S | R | Q | $\overline{Q}$ |
|---|---|---|---|
| 0 | 0 | tárol | tárol |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

A működést, melyet az igazságtábla is mutat, könnyen végigkövethetjük. Ha az $R$ bemenet 1, akkor a $Q$ kimenet biztosan 0 lesz. Ekkor ha az $S$ bemenet 0, akkor $\overline{Q}$ 0 értéket vesz fel.

Fordított esetben, ha $S = 1$ és $R = 0$, akkor a $Q = 1$ és $\overline{Q} = 0$ kimenet áll elő.

Ha $S = 0$ és $R = 0$, akkor visszakapjuk a bistabil multivibrátor esetét: a********** kimenetek bármilyen értéket felvehetnek (persze $\overline{Q}$ mindig $Q$ negáltja), és hogy $Q$ épp mekkora, azt az határozza meg hogy az $R$ és $S$ bemenetek közül melyik volt utoljára 1 – a rendszer tehát emlékszik arra hogy megelőzőleg mi volt az $R$ és $S$ bemeneteken.

Tekintve hogy az $S$ beírja a $Q = 1$ állapotot, ez a beíró vagy set bemenet. Az $R$ pedig a törlő, reset funkciót megvalósító bemenet.

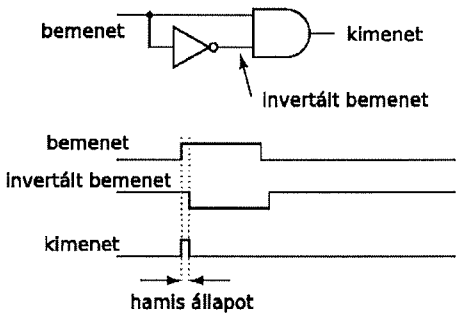
Ha az $R$ és az $S$ bemenet közül mindkettőt aktivizáljuk, akkor az áramkör viselkedése megjósolhatatlan. Az új állapot beállásánál, ha $R$ és $S$ egyszerre próbál 0 lenni, a gyorsabb kapu nyer, azaz versenyhelyzet alakul ilyenkor ki (race condition). Az áramkör előre nem meghatározható állapotba kerül, ezért ez az állapot tiltott a bemeneten.

Az RS tárolóval pl. egy motor vezérlő áramkört kapcsolhatunk, az $S$ bemenetre adott rövid impulzussal ekkor bekapcsolhatjuk, a $R$ bemenetre adott másik rövid impulzussal pedig kikapcsolhatjuk a motort.

### 1.3.2. Az engedélyező bemenet

Láttuk a kombinációs hálózatoknál, hogy kell bizonyos, több kapukésleltetésnyi idő, amíg a kimenet megkapja a helyes értékét – az addig eltelő időben gyors fel-le váltások, hazárdok jelenhetnek meg. Az RS tárolónál ugyanez azt jelenti, hogy amíg a két bemenet bizonytalanul változik, addig az RS tároló kimenete is változhat. Egy összetett rendszer szempontjából fontos lenne, ha a tároló(k) kimenete(i) csak egy alkalommal változna, ráadásul olyankor, ami egy kívülről megadott pillanat.

Induljunk el a megoldás irányába. Legyen egy újabb bemenete az RS tárolónak, és ez határozza meg azt hogy a bemenetek eljutnak-e az RS tároló lényegi, visszacsatolt részébe. Az RS áramkört tehát egyszerűen kiegészíthetjük egy engedélyező (enable, E, EN) bemenettel.




Alapesetben $A \cdot \overline{A} = 0$ mint azonosság teljesül, de látjuk az ábrán, hogy az invertált bemenet a bemenet váltásakor késik, és ez a késés elegendően hosszú lehet ahhoz, hogy az ÉS áramkör egy rövid de véges ideig logikai 1 kimenetet adjon. Ez a példa mesterkéltnek tűnhet, de egy több tucat kapuból álló áramkörnél, mint amilyen a több bites összeadó is volt, sokféleképpen előkerülhet ez a probléma: a kimenet egy ideig fel vagy le változhat, mielőtt az összes kapu kimenete stabilizálódik. Ezt a problémakört nevezik logikai hazárdnak.

Már az egyszerű áramköri hazárdok megszüntetése is speciális eljárást igényel, ezeket legtöbbször számítógépes tervezőprogramokkal végzik. A hazárdok felismerése nem mindig egyszerű. Pl. az

$$Q = A \cdot B + \bar{A} \cdot C$$

kifejezésben nem lehet rögtön észrevenni, hogy B és C logikai 1 értéke mellett előállhat az előző eset. Megoldás lehet a függvényt bővítése:

$$Q = A \cdot B + \bar{A} \cdot C + B \cdot C$$

Látható, hogy a hazárd elkerülése miatt a függvényt bővíteni kellett, azaz további kaput kell használnunk – ez pont az egyszerűsítés ellentettje.

Az elektronikai rendszerek fejlesztése során a hazárdok kiküszöbölésére olyan módszert kellett találni, amivel nagyon bonyolult rendszerek is tökéletesen jósolható módon viselkednek. A megoldás azon múlik, hogy garantáljuk: *csak abban a pillanatban nézzük meg az eredményt, amikor az már biztosan stabil.* Az alábbi fejezetekben végigkövetjük az ehhez vezető gondolatmenetet.

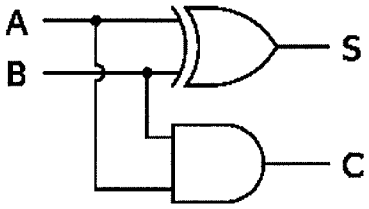
## 1.3. Belső állapottal rendelkező hálózatok

Egyszerű logikai függvényeket megvalósító kombinációs hálózatok esetén a kimenetet a bemenet(ek) pillanatnyi értéke egyértelműen meghatározza. A kimenő jel(ek) bemenet(ek)re való visszacsatolásával olyan memóriával rendelkező kapcsolásokat hozhatunk létre, amelyek állapotát a korábbi állapotok is befolyásolhatják. A memória kifejezést, ami a rendszer

| $A$ | $B$ | $C$ | $S$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

1.2. táblázat. A félösszeadó igazságtáblázata

egy átvitel bitet.

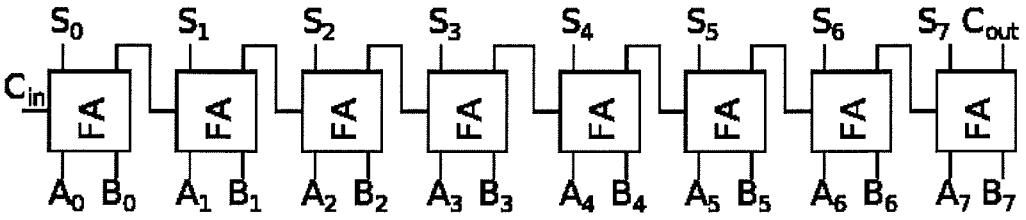



1.1. ábra. A félösszeadó felépítése kapukból.

A félösszeadó segítségével már tetszőleges bináris szám összeadására alkalmas áramkört is építetünk. A számokat jegyenként (bitenként) adjuk össze a legkisebb helyiértéktől kezdve (ahogy kisiskolában tanultuk), és most már figyelembe vesszük az áthozatal és átvitel biteket is.

A teljes összeadó (full adder, FA) működését leíró igazságtáblázat a 1.3 táblázatban látható. Az $i$-edik fokozatban a $x_i$ és $y_i$ biteket, azaz az $x$ és $y$ többjegyű számok $i$-edik jegyeit adjuk össze, az előző fokozatból kapott $c_{i-1}$ áthozatal bittel együtt (azaz 3 számot kell összeadnunk). Az eredmény a bitek $s_i$ összege, valamint a $c_i$ átviteli bit (ez lesz az áthozatal bit a következő fokozatban). Látható, hogy a teljes összeadó 2 darab 3 bemenetű logikai függvény megvalósítását igényli.

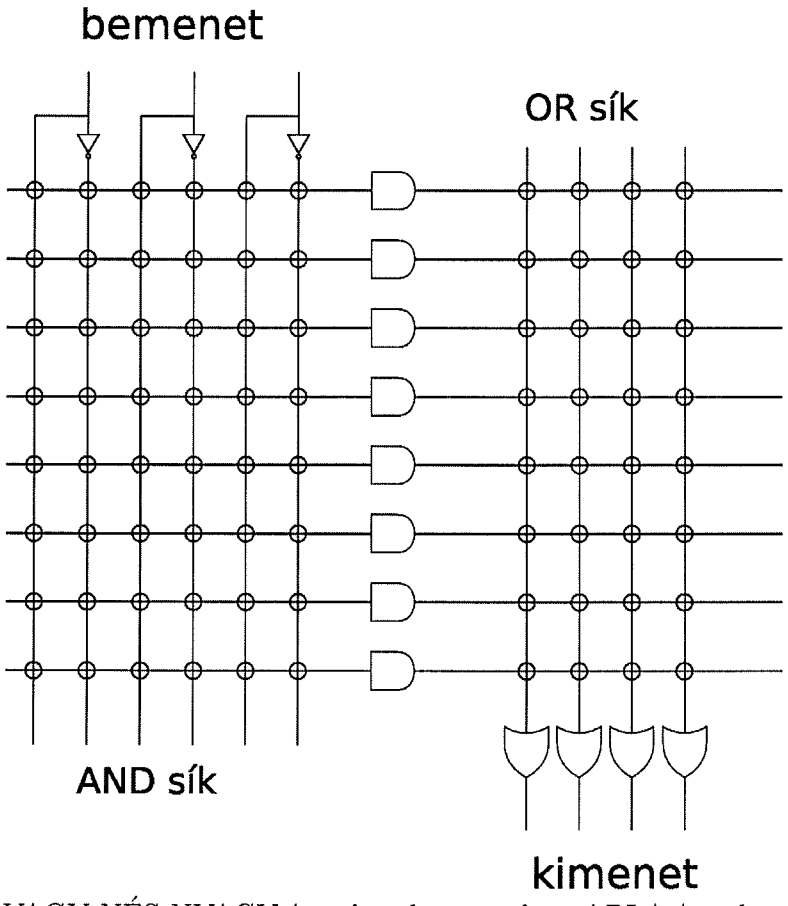
Az 1 bites teljes összeadó modul segítségével már tetszőleges jegyű, pl. 8 bites számokat is össze lehet adni:



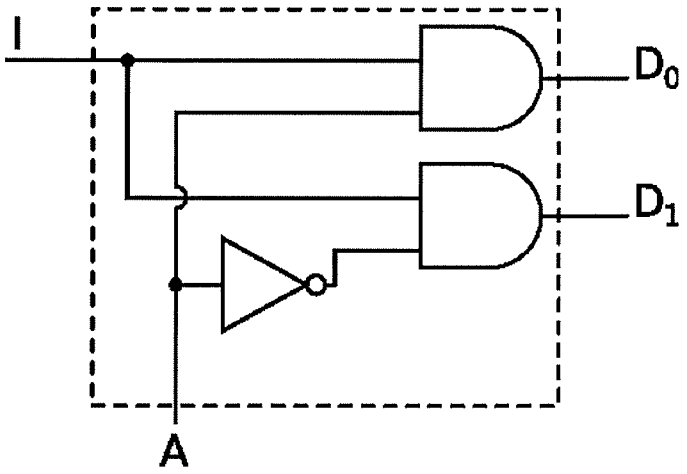

Szövegben: $x$ és $y$ van, itt $A$ meg $B$...

a hatékony megoldása lehet a programozható logikai tömb (Programmable Logic Array, PLA).

A PLA-ban programozható ÉS síkok vannak a bemeneten (ezek általában negálva is rendelkezésre állnak). Ezeket lehet összekötni a kimeneten talalható VAGY/NVAGY síkokkal. Ez az elrendezés gyakorlatilag az összes szorzatok összegét tartalmazó logikai függvény megvalósítására alkalmas (l. a 1.2 példáját). A PLA lényege, hogy a megvalósítani kívánt függvényeket külső programozó eszközzel lehet az áramkörbe beírni, a megfelelő csomópontok összekötésével. Az összeköttetések tetszőlegesen átprogramozhatók, az eszközt tehát fizikailag nem kell megváltoztatni ahhoz hogy más funkció szerint dolgozzon.




A csak ÉS-VAGY-NÉS-NVAGY áramköröket tartalmazó PLA áramkörök továbbfejlesztésével alakultak ki az FPGA (Field-Programmable Gate Array) áramkörök. Ezekben ún. logikai blokkok találhatóak, amelyek néhány adott funkciójú logikai kapuból állnak. A blokkokat nagyon gyors vezetékek és kétirányú busz köti össze a szomszédos blokkokkal.




Az áramkört kétféleképpen bővíthetjük: egyrészt növelhetjük a D kimenetek számát (ehhez növelnünk kell a dekóder logikájának méretét), másrészt növelhetjük az egyszerre kapcsolt kiválasztó típusú (A) bemenetek számát (ehhez egyszerűen párhuzamosítani kell az áramkört, miközben továbbra is egy dekódert használhatunk). Ismét igaz, hogy ha $2^n$ kimenetünk van, akkor $n$ darab kiválasztó bemenetre van szükség. A kiválasztó (A) bemenetet címnek (address) nevezzük, a kimenetet adatnak (data).

A multiplexer (mux) a demultiplexer fordítottja. Itt több I bemenet közül az A logikai vezérlővezetékkel megadott sorszámú értéket választja ki és továbbítja az egyetlen $D$ kimenetre.

A 2-1 multiplexer igazságtáblázata a következő:

| $I_1$ | $I_0$ | A | D |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

A kapcsolódó áramkör pedig:

| $I_3$ | $I_2$ | $I_1$ | $I_0$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

Az igazságtáblázat maradék 6 értékére nem kell megkötést adnunk, és így a Boole függvények könnyen előállnak:

$$D_0 = I_3 + I_1 + \bar{I}_3\bar{I}_2\bar{I}_1\bar{I}_0 + \bar{I}_3 I_2\bar{I}_1 I_0$$

$$D_1 = I_3 + \bar{I}_2\bar{I}_1 + I_2\bar{I}_1 + I_2\bar{I}_0$$

$$D_2 = I_2 + \bar{I}_3 I_2\bar{I}_1\bar{I}_0 + \bar{I}_3 I_2 I_1 I_0$$

$$D_3 = I_3 + I_1\bar{I}_0 + I_2\bar{I}_1$$

$$D_4 = I_1\bar{I}_0 + \bar{I}_2\bar{I}_1\bar{I}_0$$

$$D_5 = I_3 + I_2 + I_0$$

$$D_6 = I_3 + I_1\bar{I}_0 + \bar{I}_3\bar{I}_2 I_1 + \bar{I}_3\bar{I}_2\bar{I}_1\bar{I}_0 + \bar{I}_3 I_2\bar{I}_1 I_0$$

A kapcsolódó áramkör a következő:

bemenetek

| A | B | C | kimenet | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 0 | |
| 0 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 1 | $\overline{A}BC = 1$ |
| 1 | 0 | 0 | 0 | |
| 1 | 0 | 1 | 1 | $A\overline{B}C = 1$ |
| 1 | 1 | 0 | 1 | $AB\overline{C} = 1$ |
| 1 | 1 | 1 | 1 | $ABC = 1$ |

$$kimenet = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

### 1.2.1. Kódolók és dekódolók

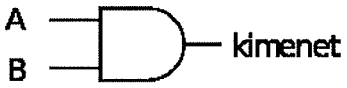
A kódoló vagy dekódoló egy olyan áramkör, amely több bemenettel és több kimenettel rendelkezik, és kimenetén a bemeneti számsornak megfelelő újabb számsort ad ki. A ki- és bemenetek száma nem szükségszerűen egyforma. Az egyik gyakran használt példa a bemenetén beadott $n$ darab értéket mint bináris számot tekint, és $2^n$ darab kimenete közül csak az az egyetlen lesz logikai 1, amelyiknek a sorszámát a bemenet megjelöli. Ennek, melyet a továbbiakban $n \rightarrow 2^n$ dekódolónak nevezünk majd, egy tipikus felhasználási módja, amikor egy bináris számmal választunk ki, aktivizálunk több közül egy adott áramkört. Pl. egy közös buszra csatlakozó 8 áramkör közül egy mindössze 3 jegyű bináris számmal kiválaszthatjuk az éppen aktivizálni kívánt egységet, azaz engedélyező, enalble jelet adhatunk neki.

A 2-ről 4-re ($2 \rightarrow 2^2$) dekódoló áramkör igazságtáblázata a következő:

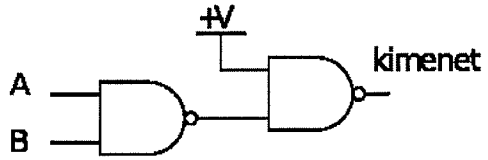
| $A_1$ | $A_0$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

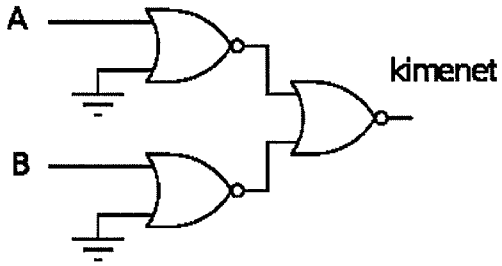
Ennek áramköri megvalósítása látható a következő ábrán:

2 bemenetű ÉS kapu

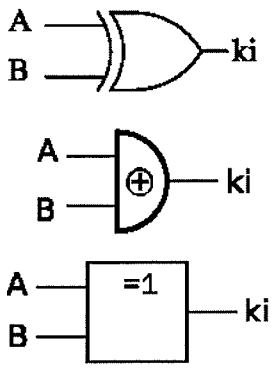



| A | B | kimenet |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



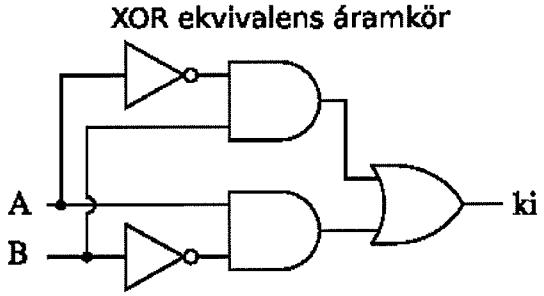




A NÉS áramkör helyettesítő kapcsolása NVAGY áramkörökkel felépítve:




| A | B | ki |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

A XOR funkciót előállíthatjuk a következő áramkörrel is:




| A | B | ki |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

A XOR áramkörök bináris összeadásnál, paritás ellenőrzésnél és kódkonverziónál hasznosak. Érdekes hogy a mindennapi beszédben a vagy szó inkább az XOR-ra utal mint az OR-ra (pénzt vagy életet). A Boole-értelemben vett VAGY-ra, ha nem egyértelmű, szokás a megengedő vagy kifejezést is használni a kizáró vagy-gyal szemben.

### 1.1.8. Egyszerűsítések és helyettesítések lehetőségei

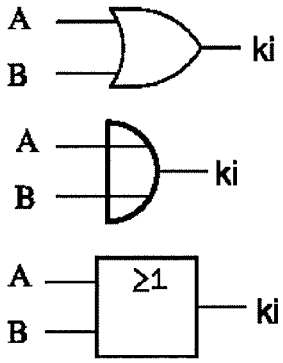
A bonyolult logikai függvények vagy összetettebb kifejezések egyszerűsítése egy hasznos lehetőség, és a matematikai kifejezések egyszerűsítésére hasonlít. Ez akár a logikai áramkör egyszerűsítését is jelenti, azaz pl. ugyanazt a funkciót kevesebb elemmel is megvalósíthatjuk. Például:

$$A + AB = A$$

vagy

### A VAGY (OR) és a NEM-VAGY (NOR) kapu

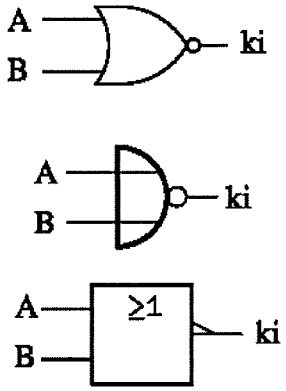
A kétváltozós Boole függvények közül a Boole + műveletet megvalósító VAGY (OR) kapu áramköri jelölései és igazságtáblázata a következő:




| A | B | ki |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

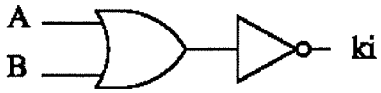
A kimenet akkor 1, ha *bármelyik* bemenet értéke 1.

A NEM-VAGY (NVAGY, NOR) kapu kimenete csak akkor 1, ha *mindkét* bemenet értéke 0. A kimeneti értékek pontosan az ellentettjei a VAGY kapuénak, így egy VAGY kapu után kapcsolt inverter ugyanezt az igazságtáblázatot adja:




| A | B | ki |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

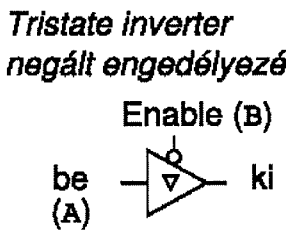
**ekvivalens áramkör**




A VAGY és NVAGY kapukat több bemenettel is gyártják, ezek ugyanúgy viselkednek: a VAGY kapu kimenete 1, ha bármely bemenete 1, a NVAGY kimenete 1, ha minden bemenet 0.

### Az ÉS (AND) és a NEM-ÉS (NAND) kapu

A Boole · műveletet az ÉS (AND) kapu valósítja meg:




| A | B | ki |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | High-Z |
| 1 | 0 | 1 |
| 1 | 1 | High-Z |

Amikor az *Enable* engedélyező vezeték magas (1) állapotban van, a kapu normál pufferként működik (ami a bemeneten az van a kimeneten). Ha az engedélyező vezeték aktív (alacsony, 0), akkor a kimenet lebeg, nagy kimenő ellenállású, az előző gondolatmenet alapján tehát bármilyen kívülről rákényszerített értéket felvehet.

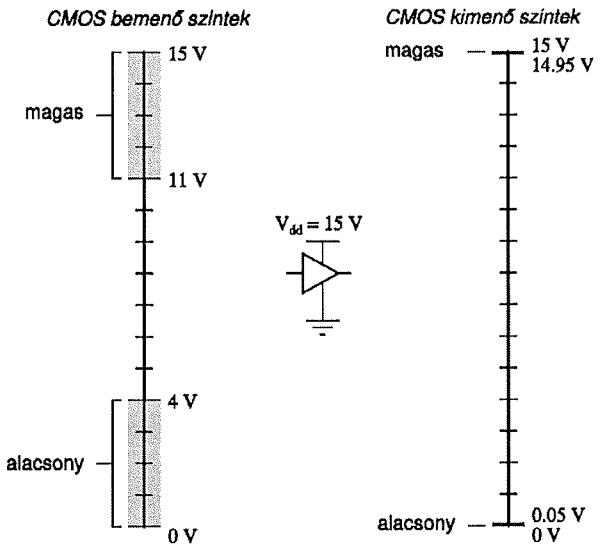
Összetett rendszereknél látni fogjuk, hogy az áramkör tervezése szempontjából nagyon hasznos tud lenni több kimenet összekötése, mert így azok be- és kimenetként egyszerre több, párhuzamosan futó vezetéket is használnak. Ezeket *busznak* ill. *buszvonalnak* nevezzük. A vezetékek száma a buszvonal szélességét (4, 8, 16, 32, 64 stb.) adja meg. Egy buszon így egyszerre 4, 8, 16, 32, 64 bitből álló információ vihető át (ennek a mérete a *szó*, word).

A buszok előnye, hogy ugyanarra a buszra több egység is rácsatlakozhat, és az adatáramlás két- vagy többirányú lehet - ehhez vezérelni kell a buszra kapcsolódó eszközöket a megfelelő vezérlő (enable) vezetékekkel. Ahhoz, hogy az inaktív egységek ne zavarják a kommunikációt az szükséges, hogy hogy az inaktív kimenetek nagyimpedanciás állapotban legyenek, ne csatlakozzanak a buszra, azaz ne terheljék és ne is akarjanak oda adatot kiküldeni. A vezérlés feladata annak garantálása, hogy egyszerre, egyidőben csak egy egység kapcsolódjon a kimenetével a buszra (logikai 0 vagy 1 szintet kiadva), a többi high-Z állapotban legyen. Ez a tri-state kimenetek legáltalánosabb felhasználási módja.

A busz mint kommunikációs vonal kicsit emlékeztethet egy társasági beszélgetésre: jó esetben mindig egyvalaki beszél és a többi hallgatja. Annak eldöntése bonyolult lehet, hogy ki legyen a beszélő, de ha valakire sor került, akkor a többieknek hallgatni illik.

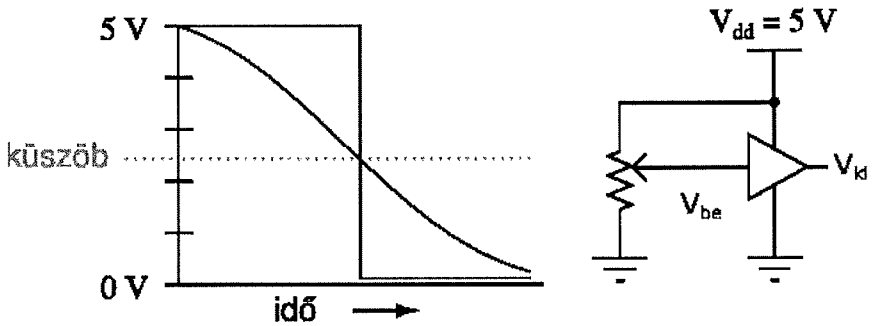
### 1.1.7. Digitális kapuk

A digitális áramkörök alapvető építőelemei a logikai (vagy digitális) kapuk. Egy kapu, bármelyik megismert családba is tartozzon, komplex áramköröket tartalmaz a lehető legnagyobb sebesség, kis fogyasztás és a nagy terhelhetőség érdekében: a kapcsolási



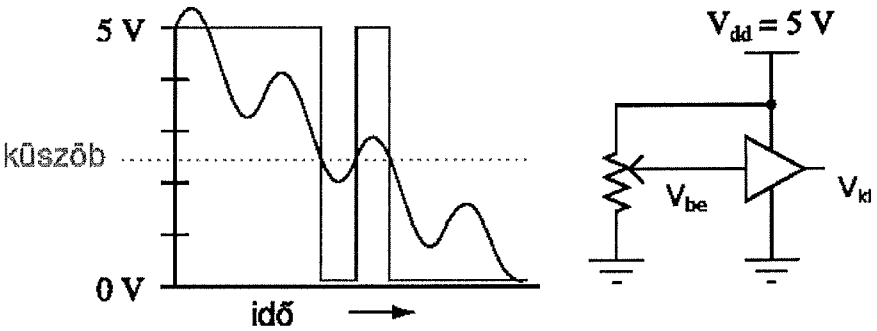

Egy normál kapu bemenete lassú jeleknél komparátorként viselkedik, zaj nélküli jelekre jól működik:

**lassan változó jel digitális kapu bemenetén**

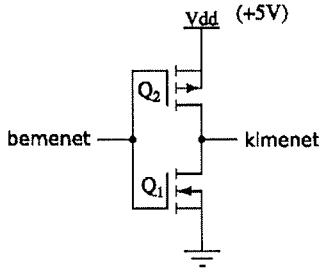



Ha a jelre valamilyen külső zaj rakódik, akkor a lassú zajos jelek helytelen működést okozhatnak, a kimeneten több impulzus jelenhet meg:

**zajos lassan változó jel detektálás komparátorral**



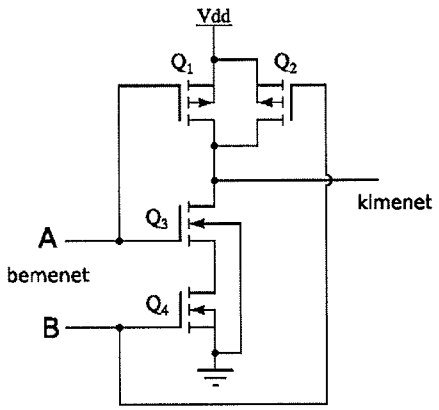
*CMOS inverter kapu*




Az áramkör működése egyszerű: ha a bemenet 0V, akkor a Q1 tranzisztor zár, a Q2 pedig nyit, a kimeneten 5V van. Ha a bemenet 5V, akkor a Q1 nyit, Q2 pedig zár: a kimeneten ekkor 0V van.

A másik példa a CMOS egy ÉS kapcsolat inverzét (negáltját) megvalósító kapu:

*CMOS NÉS kapu*




Amikor mindkét bemenet 5V, akkor a Q3 és Q4 nyit, a Q1 és Q2 pedig zár: a kimeneten ekkor 0V van. Ha bármelyik bemenet 0V, akkor a Q3-Q4 ágon zárva van legalább egy tranzisztor, a Q1-Q2 közül pedig legalább egyik nyitva van: kimeneten 5V jelenik meg.

A CMOS és a TTL kapuk logikai szintváltásnál jelentős áramot vesznek fel, ezért ügyelni kell arra, hogy ez zavarként ne befolyásolja a többi áramköri elem működését.

### 1.1.5. Jelszintek és zajtűrés

Normál működés esetén a jelek általában gyorsan (a gyors definíciója: a jel felfutása összemérhető a kapu késleltetés sebességével) váltanak át a 0-1 ill. az 1-0 szintek között. A jól kialakított kapuknál a transzfer karakterisztika kb. a billenési ponton (a tápfeszültség fele körül) a legmeredekebb, ezért alapesetben a szintváltás nem okoz bizonytalanságot.

Minden digitális rendszernél igaz egy alapvető jellemző: *a 0 és 1 logikai értéknek egy*

$$A + B = B + A$$
$$A \cdot B = B \cdot A$$
$$A + (B + C) = (A + B) + C$$
$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$
$$A \cdot (B + C) = A \cdot B + A \cdot C$$

Fontosak még az ún. DeMorgan azonosságok, amelyek következnek az eddigiekből:

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$
$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

Halmazelméleti analógia említése?

### 1.1.2. Általános Boole függvények

Egy Boole függvényt megadhatunk úgy is, ha az összes leheteséges operandus-kombinációra (minden tag 0 vagy 1 lehet) elkészítjük a függvény eredményét, azaz a kimenetet tartalmazó táblázatot. Ezt *igazságtáblázatnak* hívjuk. $N$ bemenet esetén ennek $2^N$ sora van, ahol minden függvényérték 0-t vagy 1-et vehet fel.

Például a kétváltozós ÉS művelet igazságtáblája a következő:

| $A$ | $B$ | $A \cdot B$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

A DeMorgan azonosságokat legegyszerűbben az egyenlőség két oldalán található logikai függvények igazságtáblázatának azonosságával igazolhatjuk. Pl. :

| $A$ | $B$ | $A + B$ | $\overline{A + B}$ | $\overline{A}$ | $\overline{B}$ | $\overline{A} \cdot \overline{B}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |

A kétváltozós Boole függvényeknél az igazságtáblázatnak $2 \times 2 = 4$ sora van. Ebben minden függvényértéknek 0-t vagy 1-et írhatunk, azaz az összes lehetséges különböző kétváltozós igazságtáblázatok száma $2^4 = 16$.

# 1. fejezet

# Digitális elektronika

## 1.1. Digitális eszközök működése

### 1.1.1. A Boole-algebra

George Boole (1815-1864) angol matematikus Arisztotelész logikai rendszerének tanulmányozására szimbolikus módszert fejlesztett ki. Az arisztotelészi logika szerint egy állítás vagy igaz lehet vagy hamis, más lehetőség nem választható. Ennek megfelelően minden műveletnek kétfajta eredménye lehet: igaz, amit az 1 jelöl, vagy hamis, amit a 0.

A Boole algebra kiválóan alkalmas kétállapotú rendszerek vizsgálatára is, és mi is erre fogjuk használni, a logikai kapukat tartalmazó áramkörök működésének formális leírására.

Az 1 (igaz) érték valamely definiált feszültségérték (pl. 5V) meglétét jelzi, azaz a logikai áramkör megfelelő kimenetén ezt az értéket figyelhetjük meg. A 0 (hamis) érték az előző feszültségérték nemlétét jelzi, pl. 0 V-ot. A kettő közötti értékeket az áramkörök a működési karakterisztikájuknak megfelelően kezelik, ezt az XXX fejezetben részletesebben is vizsgáljuk.

Fontos, hogy a Boole jelölést nem szabad a bináris rendszerben felírt számokkal összekeverni, mivel más objektumokat jelent a 0 és az 1: formálisan pl. a bináris rendszerben 1+1=10, míg a Boole algebrában nincs összeadás, jóllehet a + jelet ott is használni szokás.

A Boole függvények esetén a VAGY, az ÉS és a negálás (tagadás) alapműveleteket használjuk.

A Boole algebra a két logikai érték közötti VAGY (OR) kapcsolatot összeadás analógiájaként kezeli és a + jellel jelöli, a következő szabályok szerint:

# Tartalomjegyzék