

Erdélyi Magyar Műszaki Tudományos Társaság

**Magyar nyelvű szakelőadások
a 2000-2001-es tanévben**

**Kolozsvári Műszaki Egyetem
Számítástechnika Kar**

Szerzők

dr. Baruch Zoltán
Bíró Botond
dr. Buzás Gábor
dr. Farkas György
Sebestyén Pál György

Kolozsvár, 2001

Támogató
Apáczai Közalapítvány - Budapest

Lektor
Sebestyén Pál György -adjunktus
Kolozsvári Műszaki Egyetem
Számítástechnika és Automatizálás Kar

Kiadó
Erdélyi Magyar Műszaki Tudományos Társaság

Nyomdai előkészítés
Technorex Kft. - Kolozsvár

Nyomtatás
Incitato Nyomda - Kolozsvár

Tartalomjegyzék

dr. Buzás Gábor

A digitális elektronika alapjai

dr. Buzás Gábor

MOS/CMOS technológia és digitális alkalmazásai

dr. Baruch Zoltán

Digitális rendszerek tervezése a VHDL nyelv segítségével

dr. Buzás Gábor

A központi egység

dr. Farkas György

Memória technológiák

Sebestyén-Pál György

A számítástechnika ipari alkalmazásai

Bíró Botond

Fuzzy logika és alkalmazásai

A digitális elektronika alapjai

Dr. Buzás Gábor, egyetemi tanár

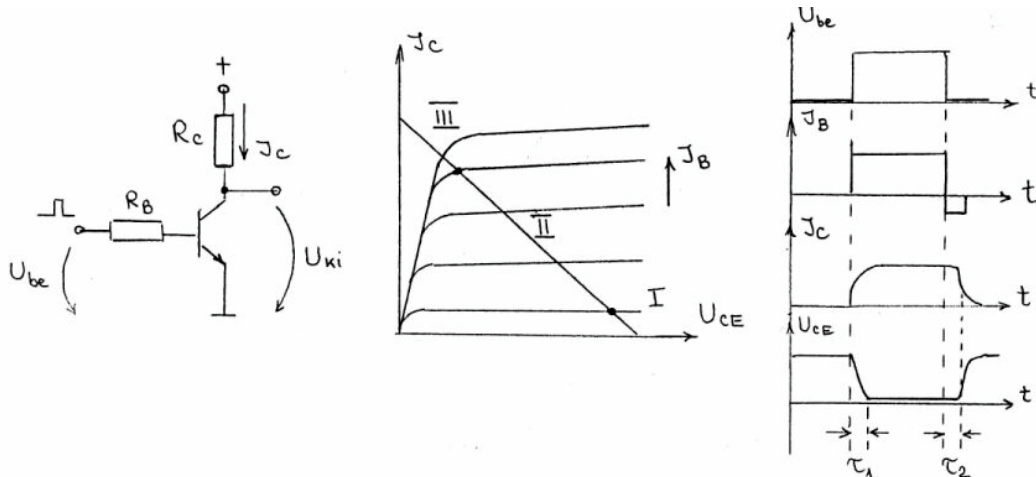
Babeş-Bolyai Tudományegyetem, Fizika Kar

A címben megjelölt témát célszerű a *digitális elektronika* alapáramkörei felől megközelíteni.

Kiindulásként tekintsük a *kapcsoló üzemmódban* működő tranzisztort, ez ugyanis a legegyszerűbb digitális áramkör, minthogy két egymástól jól elhatárolt állapota van: *zárt* és *telített*.

A *bipoláris tranzisztor* kapcsolóüzemű működését az 1. ábra szemlélteti.

A karakterisztika *munkaegyenese* mentén zajlik a működési folyamat. Az I. tartományban a tranzisztoron keresztül csak elhanyagolható áram folyik, ezért nyitott kapcsolóként viselkedik.



1. ábra. A kapcsolóüzemű tranzisztor

A II. szakasz a tranzisztor *aktív tartománya*, de kapcsoló szempontból ez most minket nem érdekel. Elérve a III. tartományt a tranzisztor telítődik, viselkedése pedig a zárt kapcsolóhoz hasonló. Ha még tovább haladunk, a tranzisztor *túlvezérlésbe* kerül, bázisrétegében nagy *töltésmennyiség* halmozódik fel. Ennek kiürítése időt igényel, amely a kapcsolási sebességre kedvezőtlen hatású. A működés gyorsításának érdekében, a telítődés meggátolására a bázis-kollektor átmenet *Schottky diódával* söntölhető. A kapcsolás idődiagramja ugyancsak az 1. ábrán látható. Megfigyelhető, hogy *ugrásjelre* a tranzisztor válasza nem azonnali, hanem csak egy bizonyos késleltetéssel jelenik meg.

Digitális elemként már csak ritkán alkalmaznak egyedi, diszkrét tranzisztort. E szerepet egyértelműen az *integrált áramkör* kapcsolások vették át, az úgynevezett *kapuáramkörök* (kapcsolóáramkörök).

Az integrált áramkörök igen sok szempont szerint osztályozhatók, mi most azonban az áramkör funkciójára és működésének időbeni megvalósulására fogunk koncentrálni. Ilyen

tekintetben léteznek alapáramkörök és összetett, de általános funkciót megvalósító áramkörök. Az időtényezőt is figyelembe véve *kombinációs* és *sorrendi* működésű áramkörök léteznek. Működésük leírható szöveges formában, *logikai függvények* segítségével, *igazságtáblázattal*, grafikus formában különböző diagramokkal, vagy valamilyen szimbolikus nyelven.

A logikai *alapfüggvények* a NEM, ÉS, VAGY kapcsolatok. Az ezeket megvalósító áramkörök a *logikai alapáramkörök* (alapkapuk). Egyszerű függvények még az ezekből származtatott ES-NEM, VAGY-NEM és a KIZARO-VAGY.

Az integrált áramkörök két nagy családja a szerint különül el, hogy a *morzsan* megvalósított aktív alkatrészek bipoláris, vagy fém-oxid-félvezető (MOS) típusú *térvezérlésű tranzisztorok* (TTL, MOS/CMOS áramkörcsalád). Az integrált áramkörök jellegzetes paraméterei: a *kapunkénti késleltetés* (a bemenetre adott ugrásjel és a kimeneten megjelenő válasz közötti idő), a *teljesítményfelvétel*, az *egységnyi felület* (egy aktív alkatrész létrehozására szükséges felület a morzsan), a *megbízhatóság*, az *integráltsági fok*, a zaj, a költségek, a technológia, stb.

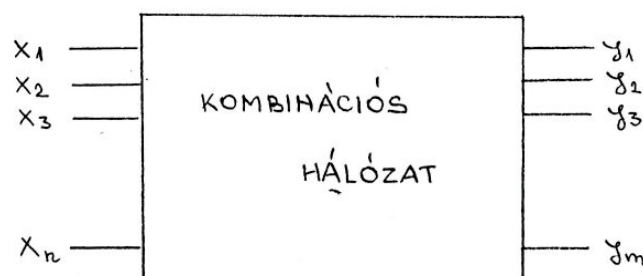
A bipoláris integrált áramkörökben megvalósuló *tranzisztor-tranzisztor logika* alapkapsolása az ES-NEM függvényt valósítja meg. Ezzel szemben a CMOS alapkapsolás a NEM függvénynek felel meg.

Az integráltság mértéke (foka) lényeges kihatással van egy adott rendszer kialakítására. Ma már ritkán találkozunk logikai rendszerekben alapáramkörökkel. A rendszerek nagy részét néhány igen magas integráltsági fokú áramkörből alakítják ki. Különös visszatérés az alapkapsokhoz az ún. *egyedi logika* (egy kapu egy tokban), amelyet legújabbán alkalmaznak tervezési hibák kijavítására, vagy utólagos hozzáadások esetében.

A következőkben azokról az áramkörökről lesz szó elvi szinten és a teljesség igénye nélkül, amelyekkel a digitális jelek feldolgozhatók és logikai funkciók megvalósíthatók.

1. Kombinációs logikai hálózatok

Kombinációs hálózatoknak nevezzük az olyan logikai függvényekkel jellemezhető hálózatokat, amelyek kimenetén a logikai értékeket a bemenetekre adott logikai érték kombinációk egyértelműen meghatározzák. Egy általános kombinációs hálózatot szemléltet a 2. ábra.



$$y_j = f(x_i) \quad , \quad i = 1, 2, \dots, n \\ j = 1, 2, \dots, m$$

2. ábra. Kombinációs hálózat és függvényrendszere

A *kimeneti változók* hozzárendelése a *bemeneti változókhoz* igazságtáblázat, vagy Boole függvények alapján végezhető. Megvalósításuk ROM típusú áramkörökkel, a logikai függvény megkövetelte működést biztosító kapukkal, vagy speciális *programozható eszközökkel* lehetséges.

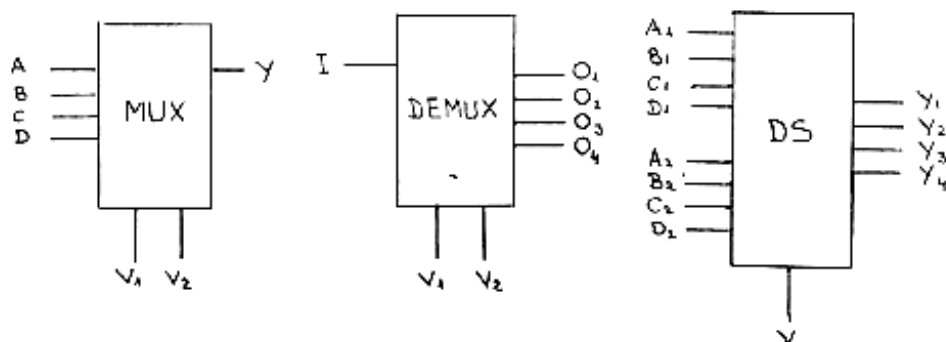
A legfontosabb kombinációs áramkörök a következők:

a) *Kódolók/dekódolók*. Ezek információk adott szabályok szerinti átalakítását megvalósító áramkörök, azaz, amelyek egyik ábrázolási módról egy másik ábrázolási módra való áttérést valósítják meg. A kódolás folyamata lehet feltételekhez kötött. A dekódolás a kódolás ellentett művelete.

b) A digitális jelek egyesítésére és szétválasztására szolgálnak a *nyaláboló* (multiplexer) és a *nyalábbontó* (demultiplexer) áramkörök.

A nyaláboló n bemenőjelből valamilyen logika szerint kiválaszt egyet, és azt megjeleníti a kimeneten. A kiválasztás *vezérlőjelek* (kiválasztójelek) segítségével történik. Amennyiben a kiválasztás nem egyedi jelekre vonatkozik, hanem adatok sorozatából kell kiválasztani egyet, akkor az eszközt *adat-szelektornak* nevezik.

A nyalábbontó a nyaláboló ellentett műveletét valósítja meg, vagyis segítségével a bemenőjel választhatóan több kimenetre csatolható. Az említett áramköröket a 3. ábrán tüntettük fel.



3. ábra. Nyaláboló, nyalábbontó, adat-szelektor

c) Az *összehasonlító* (komparátor) áramkörök két számot hasonlítanak össze egymással. Két szám azonosságának feltétele az, hogy minden megfelelő bitjük megegyezzen. Az összehasonlítás feltétele lehet a nagyobb, vagy kisebb állapot megállapítása is. Az ezt megvalósító eszköz az *amplitúdó komparátor*.

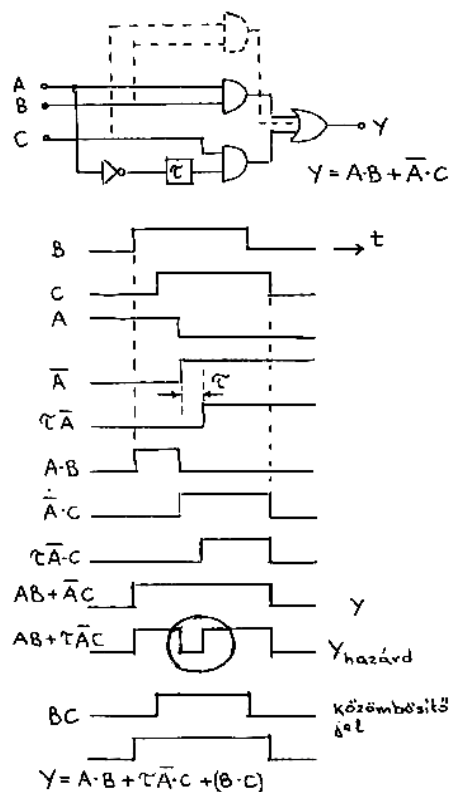
d) *Aritmetikai áramkörök*. Digitális áramkörökkel számtani műveletek is végezhetők. Az aritmetikai áramkörök számtani műveleteknek megfelelő logikai kapcsolatokat valósítanak meg a bemeneti változók között. Minden aritmetikai áramkör csak egyféle kód fogadására alkalmas. A legegyszerűbb aritmetikai áramkör a *félösszeadó*, amely két bináris számot képes összeadni az előző *átvitel* figyelembevétel nélkül. A *teljes összeadó* három bináris számot ad össze, amelyek közül az egyik az *átvitel*. Az összeadás műveletének az elvégzése történhet sorosan, vagy párhuzamosan. A többi aritmetikai művelet elvileg összeadásra vezethető vissza, de ez egyaránt bonyolítja és lassítja a műveletvégzés folyamatát. Léteznek nagy-teljesítményű és gyors műveletvégzésre optimalizált műveletvégző egységek, amelyek már a

mikroprocesszorokba is beépülnek. Műveletvégzéskor előfordul, hogy az ábrázoláshoz szükséges bitszám meghaladja a rendelkezésre állót, és *túlcsordulás* következik be.

e) *Programozható logikai eszközök.* A *programozható logikai tömb* (programozható logikai sík/hálózat) olyan logikai áramkör, amely valamely bemenő információnak valamilyen funkciót feleltet meg. Ez ugyanakkor lényeges eltérés a csak olvasható tárolókhöz képest, amelyek a bemeneti kombinációnak egy jól meghatározott tartalmat feleltetnek meg. Megvalósításuk azon alapszik, hogy csaknem minden kombináció ÉS és VAGY kapukkal létrehozható. Olyan *mátrix szerkezetet* alakítanak ki, amelyben a bemeneti változók és negáltjaik közötti ÉS kapcsolat egyszerűen kereszteződő vezetékekkel megvalósítható. Egy másik mátrixban az ÉS kapuk kimenete és a VAGY kapuk bemenete közötti kapcsolat valósítható meg hasonló módon. Kimenetként csak egy, vagy kevés számú VAGY szükséges. Mivel a logikai függvények általában *egyszerűsíthetők*, a szükséges tagok száma lecsökken, tehát rendszerint egy programozható logikai tömb több információt tárolhat, mint az azonos méretű ROM. A programozás *beíróberendezéssel*, feszültség-impulzusokkal történik. A feszültségnek megfelelő elektromos tér után még használják a *mezőprogramozható logikai tömb* megnevezést is. Előnyös tulajdonságai ellenére ezek a kapcsolások háttérbe szorultak a még fejlettebb programozható eszközök megjelenése óta. Ezekről azonban később teszünk említést.

f) *Digitális függvénygenerátorok.* A kombinációs áramkörök tetszőleges függvényalak létrehozására is alkalmasak. Egy $y=f(x)$ függvény ROM áramkör segítségével táblázatos módon valósítható meg. Nagy felbontáshoz több helyértékű számok és nagy tárolókapacitás szükséges. Ez azonban csökkenthető, ha csak a táblázat egy részét tárolják, a többit pedig ebből számolással vezetik le.

A kombinációs hálózatok használatakor fellépő *kritikus jelenség a hazard*. A késleltetések a működés során időbeli jeleltolódásokat idézhetnek elő, amelyekkel végzett logikai műveletek értelemszerűen hibákat eredményeznek. E jelenségeket nevezik hazardnak. A hazard kialakulását a 4. ábrán tanulmányozhatjuk, ahol egy tagadó áramkörnél megjelenő többletkésleltetést vettük figyelembe.



4. ábra. A hazard jelenség keletkezése

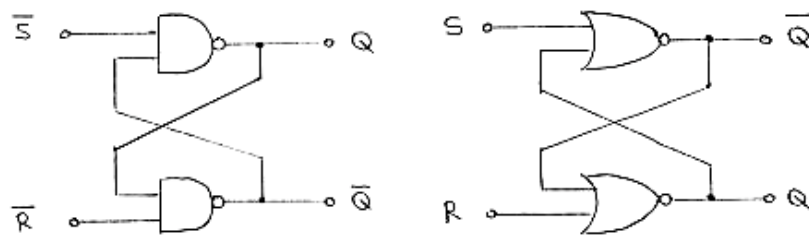
Látható, hogy a hazard az *élszomszédos* tömbök *határvonalán*, azok eltolódása miatt keletkezik. A hazard megszüntethető, ha a kritikus helyet egy *redundáns*, a logikai függvényt nem befolyásoló tömbbel lefedik (közömbösítik).

2. Sorrendi hálózatok

Sorrendi (szekvenciális) hálózatnak nevezik az olyan logikai hálózatokat, amelyek kimeneti változói nemcsak a bemeneti változóktól függenek, hanem az illető rendszer előző állapotától is.

A továbbiakban a legfontosabb sorrendi áramkörökkel fogunk megismerkedni.

a) A billenő áramkörök olyan pozitívan visszacsatolt áramkörök, amelyek kimenőjele ugrásfüggvény szerint változik és az átbillenés következtében csak két diszkrét értéket vehet fel (L,H). A bistabil billenő áramkör a számítástechnika kiemelt fontosságú eleme. A bistabil billenőkörök alapkapcsolásának az RS bistabil tekinthető. Megvalósítható ES-NEM, vagy VAGY-NEM kapukkal. Ezt és a működésének megfelelő igazságtáblázatot az 5. ábra mutatja be.



Vezérlés	Vezérlés utáni kimenetek	Funkció
R S	Q Q̄	
L L	Q, Q̄ előző állapot	Emlékezés az előző állapotra
L H	H L	Billenés
H L	L H	Visszabillenés
H H	Nem egyértelmű	Nem billenő állapot

5. ábra. RS billenőkör

Belátható, hogy a kapcsolás időfüggése a visszacsatolás miatt lép fel. Nagyon fontos észrevétel, hogy az egyik kimenetere vonatkoztatva (pl. Q kimenetre), a kapcsolás *elemi tárolócellának* tekinthető. Az RS billenőkört csak jobb tulajdonságokkal rendelkező egyéb billenőkörök alapvető részegységeként alkalmazzák. A jobb tulajdonságok alatt itt elsősorban azt értjük, hogy működése *órajellel* vezérelhető és hogy csak jól meghatározott állapotokkal rendelkezik. Az órajelvezérlés lehet *élvezérléses*, vagy *szintvezérléses* elvű. A legelterjedtebb kapcsolások a *D bistabil* és a *közbenső tárolót tartalmazó mester-szolga elvű* bistabil. A bistabil áramköröket rendszerint *nullázó* (Reset) és *egy-beíró* (Preset) bemenetekkel is ellátják.

b) *Számlálók*. A számlálók sorrendi áramkörökből felépülő funkcionális egységek, amelyek a bennük tárolt információ értékét a bejövő jel hatására eggyel növelik, vagy csökkentik. A számlálók hasznos belső állapotainak számát, amely egyben a maximálisan megszámlálható értékek számát jelenti *modulónak* nevezik és a számláló fontos jellemzője. A számlálók összekapcsolt bistabil áramkörökből épülnek fel. A számlálók több szempont alapján osztályozhatók:

- a számlálót felépítő bistabilok kapcsolása alapján: soros, párhuzamos
- a számlálót felépítő sorrendi hálózat jellege alapján: aszinkron, szinkron
- a számlálás iránya alapján: előre, hátra, kétirányú számláló
- a számlálandó információt kifejező kódrendszer alapján: bináris, BCD stb.

Az aszinkron számláló jellegzetes tulajdonsága, hogy az impulzusok csak az első egység órabemenetére kerülnek, ezért a tárolóelemek állapotváltozása a terjedési időkkal eltoltan láncolódva történik. Ez egy hátrányos tulajdonság. Szinkronszámlálóknál az órajel hatása minden tárolóelemnél azonos időben érvényesül. Az *előválasztó számláló* olyan számláló, amely akkor ad kimenőjelet, ha a bemenő impulzusok száma azonos egy előre meghatározott számmal.

c) *Regiszterek*. A regiszterek közös vezérléssel rendelkező egymással összekapcsolt bistabil áramkörök, amelyekbe digitális információk írhatók, vagy onnan olvashatók. Alapvető tulajdonságuk, hogy a beléjük sorosan, vagy párhuzamosan bevitt információt meghatározott ideig tárolni tudják, esetleg a tárolt információval relatív helyváltoztatást (*léptetőregiszter*) képesek végezni és a tárolt információ soros, vagy párhuzamos kiadására alkalmasak

d) *Tárolók.* A tárolók (memóriák) nagyobb mennyiségű információ átmeneti, vagy tartós tárolására szolgáló egységek. A tárolók jellegzetes csoportját a *félvezető tárolók* alkotják, itt csak ezeket fogjuk tárgyalni. A tárolók is sokféle szempont szerint csoportosíthatók. Ezek közül a leglényegesebbek:

- a tartalom módosíthatósága szerint: *változtatható*, vagy *rögzített* tartalom
- a tárolás időbeli módja szerint: statikus, dinamikus
- a címzés alaprendszere szerint: *közvetlen (véletlen) elérésű*, *soros elérésű*, *asszociatív elérésű*
- gyártástechnológia szerint: bipoláris, MOS
- adattárolás szervezése szerint: bit-szervezésű, szó-szervezésű

A tárolócellák az egy bit tárolását elvégző tárolóelemek. A statikus tárolók bipoláris, vagy MOS, a dinamikus tárolók pedig MOS technológiával készülnek. A dinamikus cella a tartalmát csak rövid ideig képes tárolni, ezért periodikus frissítést igényel. A PC típusú számítógépek operatív tárolója (RAM memória) dinamikus tároló, vagy ennek valamilyen továbbfejlesztett változata. A tárolók egyik legfontosabb jellemzője az *elérési (hozzáférési) idő*, azaz a címzés és a neki megfelelő adat megjelenése között eltelő időintervallum.

A tárolókkal kapcsolatosan számos fogalom szerepel a szakirodalomban, ezeket foglaljuk most össze.

- *olvasható írható tároló* (Read Write Memory - RWM)
- *közvetlen hozzáférésű tároló* (Random Access Memory - RAM)
- *csak olvasható tároló* (Read Only Memory - ROM)
- *programozható csak olvasható tároló* (Programmable ROM - PROM)
- *törölhető programozható csak olvasható tároló* (Erasable PROM - EPROM)
- *elektromosan törölhető programozható csak olvasható tároló* (Electrically Erasable PROM - EEPROM)
- *tartalom alapján címezhető (asszociatív) tároló* (Content Addressable Memory - CAM)
- *soros elérésű tároló* (Serial Access Memory - SAM)
- *flash tároló* (flash memory), ez nemfelejtő, írható-olvasható félvezető tároló, amely szavanként írható, blokkonként törölhető és a többi típushoz képest lassú

e) *Programozható eszközök.* A *mezőprogramozható kapumátrix* az *alkalmazásorientált* (ASIC) áramkörök egyik alapvető típusa. Három fő belső alegységből épül fel: *konfigurálható logikai blokk*, *bemeneti-kimeneti egységek*, *programozható összeköttetések*. A konfigurálható logikai blokk tartalmazza mindazon elemeket, amelyek valamely logikai függvény megvalósításához szükségesek. A bemeneti-kimeneti egységek mindegyike egymástól függetlenül bemenetként, kimenetként, vagy *kétirányú kapuként* határozható meg. A programozható összeköttetések segítségével az előbbieket közötti kapcsolat valósítható meg. Rendkívüli rugalmasságuk miatt a mezőprogramozható kapumátrixok alkalmazása igen sokrétű. Akár különleges mikroprocesszorok létrehozására is alkalmasak. A legmodernebb típusuk az ún. *memória-alapú mezőprogramozható kapumátrix*, amelyik az említett három alegységen kívül memóriát is tartalmaz. A bekapcsolás pillanatában az óhajtott konfigurációnak megfelelő tartalom az alkalmazási felületre telepített EPROM-ból a belső

A sorrendi hálózatok használatakor fellépő kritikus jelenség a *versenyfutás*. Versenyfutás több visszacsatoló hurkot tartalmazó sorrendi hálózatonál léphet fel. Akkor jelenik meg, ha egyidejűleg több visszacsatoló hurokban történik állapotváltozás és ekkor a hurkokban szereplő különböző késleltetések következtében a hálózat állapotváltozásai nem mindig lesznek egyértelműek.

Az elektronikai elemekből felépülő berendezésekben mind analóg, mind pedig digitális kifejezőmód előfordulhat. Vegyes rendszereknél a találkozási pontokon értelemszerűen *analóg-digitális*, illetve *digitális-analóg átalakítást* kell végezni.

Az analóg-digitális átalakítóknak három alaptípusa ismeretes:

- A digitális-analóg átalakítók a digitális jelek analóg típusú átalakítására szolgálnak. Ez áramkörileg a számkódolt adatoknak velük arányos feszültséggé, vagy árammá való átalakítását jelenti. Annak ellenére, hogy többtípusú digitális-analóg átalakító ismeretes, működésük rendszerint a *súlyozott áramok* összegezésére vezethető vissza.

Pillanatnyilag a digitális elektronika legbonyolultabb és legfejlettebb technológiát képviselő áramkörei a mikroprocesszorok, *mikrovezérlők* és a *digitális jelprocesszorok*, de ezek akár felületes bemutatása meghaladja e sorok kereteit.

- 1] Szittyá O.: *Analóg és digitális technika*, LSI, Budapest, 1999.
- 2] K.Beuth - O.Beuth: *Az elektronika alapjai* - III. kötet, *Digitális elektronika*, Műszaki, Budapest, 1994.
- 3] U. Tietze - Schenk: *Analóg és digitális áramkörök*, Műszaki, Budapest, 1981.
- 4] Buzás G.: *Alkatrészek, eszközök, technológiák*, Cédrus, Budapest, 1995.

MOS/CMOS technológia és digitális alkalmazásai

Dr. Buzás Gkábor, egyetemi tanár

Babeş-Bolyai Tudományegyetem, Fizika Kar

A térvezérlésű tranzisztorok (*Field Effect Transistor - FET*) a bipoláristól eltérő fizikai elven működnek. Szemben a bipoláris tranzisztorral, amelyben elektron- és lyukvezetés egyaránt előfordul, ezek áramát csak egy töltéshordozó biztosítja. Ezért ezt az eszközt *unipoláris* tranzisztornak is nevezik.

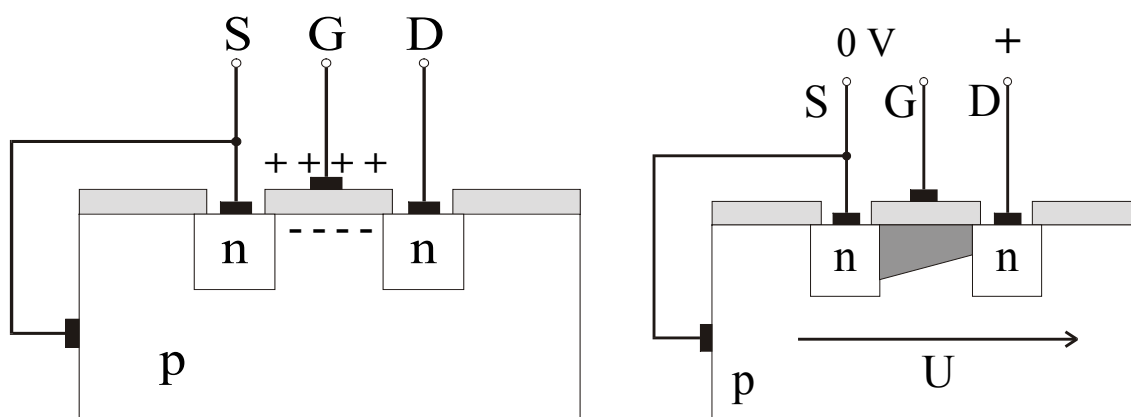
Két alapvető típusuk a záróréteges- és a szigetelt elektródás térvezérlésű tranzisztor.

1. Szigetelt vezérlőelektródájú térvezérlésű tranzisztor (MOSFET)

A térvezérlésű tranzisztorok e típusa 1960-ban jelent meg. Ez a megnevezés elsősorban típusukra utal (*Isolated Gate Field Effect Transistor - IGFET*). Elterjedtebb azonban a felépítésükkel összefüggő megnevezés, éspedig a *fém-oxid-félvezető térvezérlésű tranzisztor* (*Metal Oxid Semiconductor Field Effect Transistor - MOSFET*). Egyszerűsége és rövideje miatt, magyarul is gyakran a MOSFET vagy MOS tranzisztor megnevezést használják.

1.1. A MOS tranzisztor felépítése és működése

A MOSFET aktív része a *p* típusú szubsztrát, amelyben két *n* típusú szigetet alakítanak ki. Ezt oxidréteggel vonják be, amelyen a szigeteknél ablakot nyitnak. A oxidrétegre a két sziget közötti tartományban vékony fémréteget párologtatnak. Az oxidréteg (SiO_2) jó szigetelő és jó átütési szilárdságú. E struktúrát az 1. ábra mutatja be. A szigeteket és a fémréteget kivezetésekkel látják el, ezek alkotják az eszköz külsőleg hozzáférhető elektródjait. A két sziget a forrás és a nyelő, a fémréteg pedig a kapu szerepét tölti be.



1 ábra. A MOS tranzisztor elvi felépítése

A *p* szubsztrátban többségi lyukak és kisebbségi elektronok vannak. Ha a kapura pozitív feszültség kerül, akkor az elektronok a kapu alatti réteghez vándorolnak, és ott akár többségbe is kerülhetnek, annál is inkább mert a lyukak ellenkező irányba mozognak. Ezáltal a forrás és

nyelő között n vezető csatorna jön létre, amelyen keresztül polarizálás esetén a forrás-nyelő árama bezárulhat. A csatorna szélessége ebben az esetben is változó, a nyelő felé növekvő potenciálnak megfelelően. A pozitív kapufeszültség növelésével a csatorna elektronkoncentrációja megnő, tehát jobban vezetővé válik. Ha a pozitív feszültség csökken, a fordított jelenség játszódik le. Tehát a csatorna vezetőképessége a kapufeszültséggel szabályozható.

Hasonlóan a záróréteges térvezérlésű tranzisztorhoz, a nyelő árama a kapufeszültséggel teljesítmény felvétele nélkül vezérelhető.

A szubsztrát és a forrás rendszerint közös potenciálra (0 V) kerül. Érdemes megjegyezni, hogy a MOS struktúrákban lezajló vezetési mechanizmus a félvezető felületi rétegében megy végbe, ellentétben a záróréteges esettel, ahol a vezetés a félvezető tömbben valósul meg. Könnyen belátható, hogy a MOS tranzisztorok esetében is a forrás és nyelő szerepe felcserélhető.

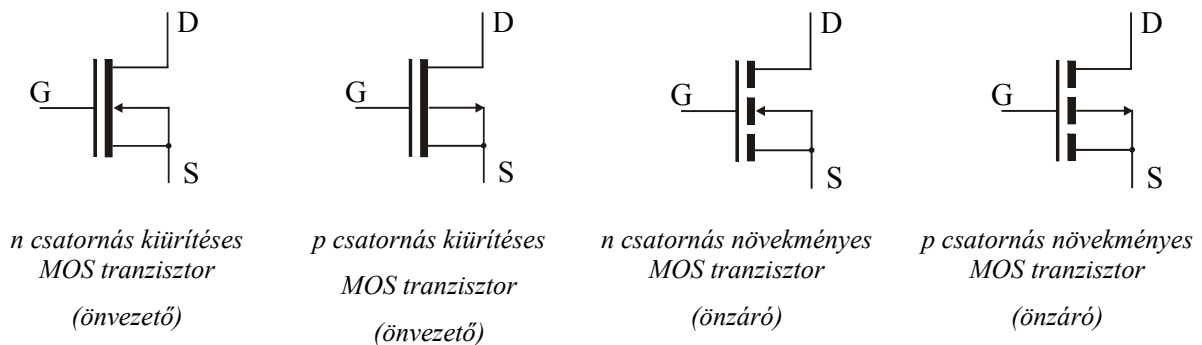
Vezérlés szempontjából a MOS tranzisztor két változata különböztethető meg:

A *növekményes (betöltéses)* MOS tranzisztornál nulla kapufeszültség esetén (nyitott kapu) a forrás-nyelő szakasz lezár. Ezért ezt *önzáró* típusnak is nevezik. Csatorna csak akkor keletkezik, ha az oxidréteg közelében elektrondúsulás lép fel, amely pozitív kapufeszültséggel érhető el.

A *kiürítéses* MOS tranzisztornál nulla kapufeszültség esetén is csatorna jöhet létre a forrás és nyelő között, tehát nyitott kapuval is vezet. Ez az *önvezető* típus. E típus elvileg mind negatív, mind pozitív kapufeszültséggel vezérelhető. Negatív vezérlés esetén a csatorna elektronokban elszegényedik (kiürül) és adott értékeknél elzáródik (*elzáródási feszültség*).

Hasonlóan p csatornás MOS tranzisztorok is létrehozhatók.

A 2. ábra a MOS tranzisztorok rajzjeleit tünteti fel.



2. ábra. A MOS tranzisztorok rajzjelei

A következőkben néhány más MOS tranzisztor változatot mutatunk be.

1.2. VMOS tranzisztor

Ellentétben a hagyományos MOS tranzisztorokkal, ahol a csatorna a félvezető felülettel párhuzamos, a VMOS tranzisztoroknál (*Vertical MOS*) a csatorna a felületre merőleges, az ezt vezérlő tér iránya pedig a felülettel párhuzamos.

1.3. SOSFET tranzisztor

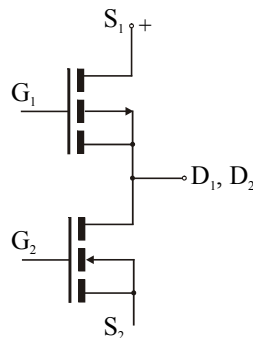
A szubsztrát alatt egy zafírréteg helyezkedik el. Innen származik a szilícium zafíron (*Silicon on Sapphire*) elnevezés. Mivel elmaradnak a hordozókban fellépő parazita elemek (a hordozó nagy ellenállása miatt) a működési sebességük nő.

A gyártási módszer alapján vékonyréteg tranzisztornak is nevezik. Működésük megegyezik a többi térvezérlésű tranzisztoréval.

1.4. Kettős kapujú MOS tranzisztor

E különleges tranzisztorfajta úgy jön létre, hogy egy MOS tranzisztor nyelőjét egy másik forrásával kötik össze, amelyek eredményeképp a csatorna hosszúsága megnövekedik, vezérlésére pedig két kapu áll rendelkezésére. Nagyfrekvenciás kapcsolásokban és keverő-fokozatokban használható előnyösen.

Hasonlóan, ugyanazon a hordozón sorosan kialakítható egy p csatornás és egy n csatornás MOS tranzisztor pár. E struktúrát *komplementer* MOS tranzisztornak (*Complementary MOS - CMOS*) nevezik. A CMOS elrendezését a 3. ábrán mutatjuk be.



3. ábra. A CMOS tranzisztor

A CMOS tranzisztor zaja alacsony, teljesítményfelvétele kisebb, működése pedig gyorsabb. A velük kapcsolatos leglényegesebb megjegyzés azonban az, hogy kiválóan megfelel az integrálási technológiáknak.

1.5. FAMOS tranzisztor

Ez egy rendkívül jelentős térvezérlésű tranzisztor. Egyik jellegzetessége, hogy a kapunak nincs kivezetése (*lebegő kapu*), felületét oxidréteg borítja. A nyelőre viszonylag magas feszültséget adva, a lezárt pn átmeneten lavinajelenség lép fel, hasonlóan a hagyományos diódák zárófeszültségű üzemeléséhez. Ennek eredményeként a nagyenergiájú „*forró elektronok*” áthatolnak a vékony oxidrétegen és a lebegő kapura kerülnek. Mivel innen nem vándorolhatnak el, a kapu negatívan töltődik fel. p csatornás struktúra esetében ez a forrás és nyelő között csatorna létrejöttét jelenti és az eszköz vezető állapotba kerül és stabilan az is marad. A lebegő kapu és a lavinajelenség adja az eszköz nevét: *lebegőkapus, lavianjelenséges* MOS tranzisztor (*Floating Gate Avalanche MOS - FAMOS*). Az üzembiztos működés feltétele, hogy a kapu ne veszítse el a töltéseket. Ez kb. 100 év. A kapu töltése gyorsan eltávolítható pl. ultraibolya besugárzással, amelynek hatására fotóáram jön létre. Ezáltal tehát a tranzisztor vezetése megszüntethető. A leírt „záras-nyitás” folyamat sokszor

megismételhető. Az eszköz említett rendkívüli jelentősége a programozható memóriák gyártásában van.

A FAMOS tranzisztor a törölhető újraprogramozható csak olvasható tároló (EPROM) elemi tárolócellája.

1.6. MOS kondenzátor és MOS ellenállás

Kialakítástól és üzemmódtól függően a MOS tranzisztor kondenzátorként vagy ellenállásként is működhet.

A *MOS kondenzátor* a tranzisztor oxidrétege alatt jelentkező kapacitás, amely az oxidréteg kapacitásával sorban jelenik meg. A kapura csatolt nyitófeszültség hatására a felület alatt egy meghatározható vastagságú és töltésű kiürített réteg keletkezik. Értékét alapvetően a kapufeszültség határozza meg. Jellemző értéke néhány pF.

Mint említettük, a térvezérlésű tranzisztor *ellenállásként* működik mindaddig, amíg a forrásnyelő feszültség a könyökfeszültség alatt van. E tartományban az eszköz feszültséggel vezérelhető lineáris ellenállásként üzemel. Értékei a gigaohm tartományig terjednek.

Az itt leírt megállapítások igen nagy jelentőséggel bírnak az integrált technológiában, ugyanis ugyanaz a struktúra, de különböző körülmények között üzemelhet mint ellenállás, kondenzátor vagy tranzisztor.

1.7. A bipoláris és a térvezérlésű tranzisztor összehasonlítása

A bipoláris tranzisztorokkal szemben a térvezérlésű tranzisztorok nagy előnye az egyszerűbb gyártástechnológia (ebből következően az előállítási költsége kisebb) és a sokkal csekélyebb hőmérsékletfüggés. Az utóbbinak oka az, hogy működésük a többségi töltéshordozók mozgásán alapul. A térvezérlésű tranzisztorok nagyobb hányadánál az áram hőmérséklet-együtthatója negatív. Ez azért kedvező, mert így a hőmérséklet növelésével a disszipáció csökken, amely kizárja a hőmegfutás veszélyét. A térvezérlésű tranzisztorok még abban is lényegesen eltérnek a bipoláris tranzisztoroktól, hogy a szokásos üzemmódokban csak elhanyagolható vezérlőtéljesítményt igényelnek.

Mivel a térvezérlésű tranzisztorok közül a MOS típusnak kiemelt jelentősége van az integrálhatóság szempontjából, a továbbiakban ezeket hasonlítjuk össze az alábbi táblázatban a bipoláris tranzisztorokkal, néhány fontos jellemző alapján.

Jellemző	Bipoláris tranzisztor	MOS / CMOS tranzisztor
Teljesítményfelvétel	közepes	igen kicsi
Kapcsolási idő	nagyon kicsi	relatív kicsi
Bemeneti ellenállás	kicsi	nagyon nagy
Terhelhetőség	jó	nagyon jó
Zaj	kicsi	nagyon kicsi
Gyártástechnológia	bonyolultabb	egyszerűbb
Integrálhatóság	elvileg gyengébb	igen magas fokú

1.8. Töltéscsatolt eszköz

A *töltéscsatolt eszköz* (*Charge Coupled Device - CCD*) olyan félvezető eszköz, amelynek egyik elemében tárolt töltésmennyiség átvihető a szomszéd elembe külső potenciálok megfelelő alkalmazásával. A töltéscsatolt eszköz 1969-ben jelent meg és néhány év alatt igen látványos fejlődésen ment keresztül.

A működési alapelv azon a felismerésen alapul, hogy a MOS kapacitás bizonyos körülmények között és rövid ideig analóg információ tárolására alkalmas. Két kellő közelségben kialakított MOS kapacitással elérhető, hogy kiürített tartományaik érintkezzenek egymással. Megfelelő kapufeszültséggel az is megvalósítható, hogy a közös potenciálgödör különböző mélységű legyen (lépcsőzetesség). Ennek következtében az elsőben eredetileg jelenlevő töltésmennyiség a másikba fog átvándorolni. Az egymással érintkező, különböző mélységű potenciálgödrök által lehetővé váló töltésátvitel a tulajdonképpeni töltéscsatolás. Lényegében tehát egy elektród (kapu) alatt tárolt töltésmennyiség a szomszédos elektróddal vezérelhető a potenciálgödör mélységének a befolyásolása által. A vázolt módon működő MOS kapacitáslánc képezi a töltéscsatolt eszközt. A kapupotenciálok változtatásához, tehát a töltésátvitel megvalósításához, egy három vagy négy fázisú jelre van szükség (órajel).

A felületi töltéstárolás és mozgatus a félvezető tömbben is megvalósulhat.

A töltéscsatolt eszközöket integrált kivitelben gyártják. Alkalmazásának lényeges területe az analóg jelfeldolgozás, minthogy működési elve analóg. Talán legfontosabb alkalmazása a töltéscsatolt eszköz alapú képfellevő, amelynél a mozgatusra kerülő töltéscsomagok megvilágítás hatására jönnek létre. Manapság kizárólag ilyen típusú képfellevőket használnak.

A működési elvet tekintve ugyancsak töltéscsatolt eszköz a *vödörlánc eszköz*, valamint a *töltésinjektálású eszköz*.

Az előbbi lényege az, hogy egy sorbakötött MOS tranzisztorkánc egymás után következő elemeit két vezérlőimpulzus felváltva, egymás után nyitja. Ezáltal az első tranzisztor bemeneti kondenzátorában felhalmozott töltés vödörláncszerűen töltődik tovább a többi tranzisztor között levő kondenzátorokba (kézről-kézre adott tűzoltó veder elve). Az elemek száma, valamint a vezérlőimpulzusok periódusa ismeretében a lánc késleltetése pontosan meghatározható.

A töltésinjektálású eszköz lényegében egy MOS kapacitás-párokból kialakított mátrixszerkezet. Egy páron belül, ha legalább egy kapu megfelelően polarizált, akkor például a fény hatására létrejött töltések tárolhatók és mozgathatók. Ha viszont mindkét kapu potenciálja nulla, akkor megszűnik a kiürítési tartomány (potenciálgödör) és a töltések a hordozóba injektálódnak, majd ott rekombinálódnak. A rekombináció viszonylag lassú, ezért az injektált töltések semlegesítésére más módszereket is alkalmaznak. A párok kapui a mátrix sorait és oszlopait képezik. A töltésinjektálású eszközöket főleg képfellevőként alkalmazzák.

2. Digitális MOS/CMOS alkalmazások

A következőkben néhány jellegzetes digitális MOS/CMOS alkalmazást mutatunk be.

2.1. MOS/CMOS logika

A MOS tranzisztorok jellegzetes alkalmazása a digitális elektronikában van. Emlékeztetünk a MOS tranzisztor figyelemreméltó tulajdonságára, hogy tranzisztorként, ellenállásként, vagy kapacitásként viselkedhet.

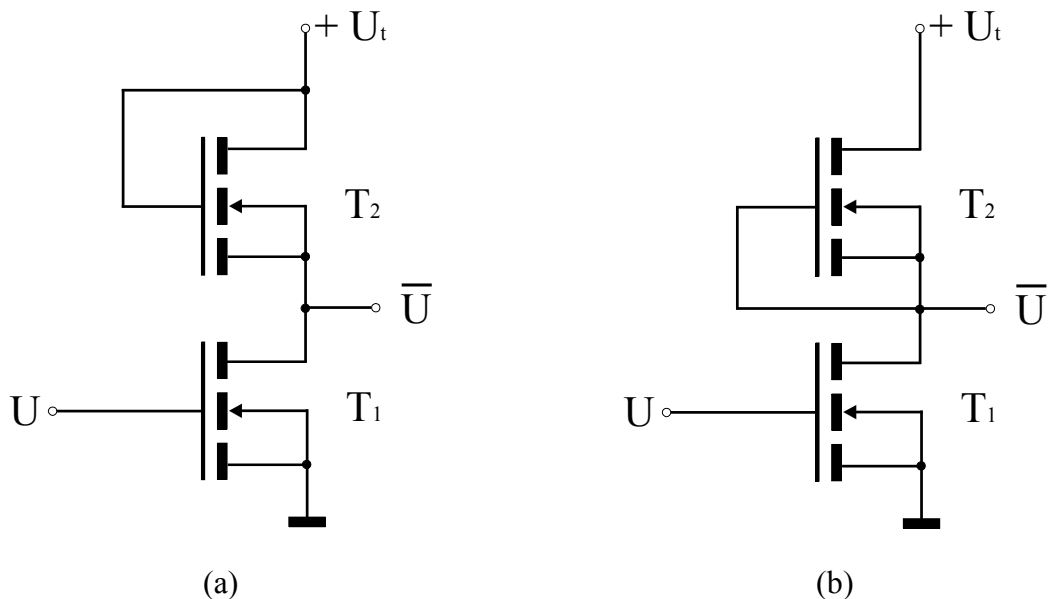
a) MOS logika

A MOS kapuáramkörök terhelő fokozata - amely egy ellenállásként felhasznált MOS tranzisztor - passzív, vagy aktív felépítésű lehet, és az a teljes kapuáramkör jellegét meghatározza.

A passzív terhelésnél a felhasznált MOS kapuját egy állandó értékre kötik, úgy, hogy az eszköz soha ne zárjon le.

Az aktív terhelésű MOS kapcsolásoknál maga a terhelőfokozat is vezérelt elem.

Az egyik MOS alapkapcsolás a tagadó (NEM) áramkör. Ennek két típusa létezik, amelyeket a 4. ábra mutat be.



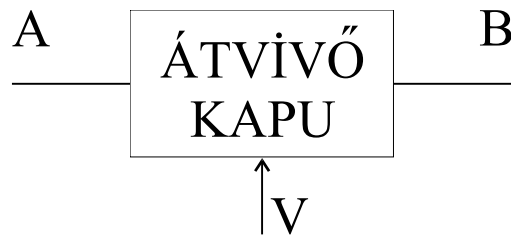
4. ábra. MOS tagadó kapcsolás, telítéses (a), kiürítéses (b)

Telítéses típusú tagadó áramkörnél, mindkét tranzisztor növekményes és azonos típusú (rendszerint n -csatornás). E típus előnye az egyszerű technológia. Hátránya, hogy a logikai L magasabb szintről indul, a H viszont alacsonyabb. További hátrány, hogy a kapcsoláskor a felfutó él meglehetősen lassú a lefutóhoz képest.

A kiürítéses változatnál a terhelő tranzisztor kiürítéses, a vezérelt tranzisztor pedig növekményes és mindkettő n -csatornás. A gyártástechnológiája bonyolultabb, de a telítéses típusnál említett hátrányok előnyösen módosultak. E két típus közül a kiürítéses típust használják elterjedtebben.

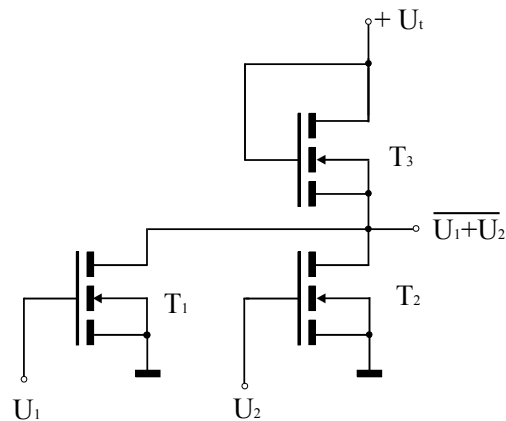
Hasonló kapcsolások p -csatornás tranzisztorokkal is megvalósíthatók. A MOS tranzisztor működési elve lehetővé teszi, hogy a rajta keresztül folyó áram forrás $\rightarrow$ nyelő, vagy nyelő $\rightarrow$ forrás irányú legyen. E tulajdonság lehetővé teszi kétirányú átvivő (transzfer) kapuk megvalósítását.

Az átvivő kapu elvi kapcsolását tünteti fel az 5. ábra. A V vezérlőjel állapotától függően az átvitel $A \rightarrow B$, vagy $B \rightarrow A$ irányú lehet.



5. ábra. Átvivő kapu elvi kapcsolása

Háromállapotú kimenet úgyszintén megvalósítható MOS logikával. A MOS logika másik alapkapcsolása a VAGYNEM áramkör (6. ábra)



6. ábra. VAGYNEM kapcsolás MOS logikával

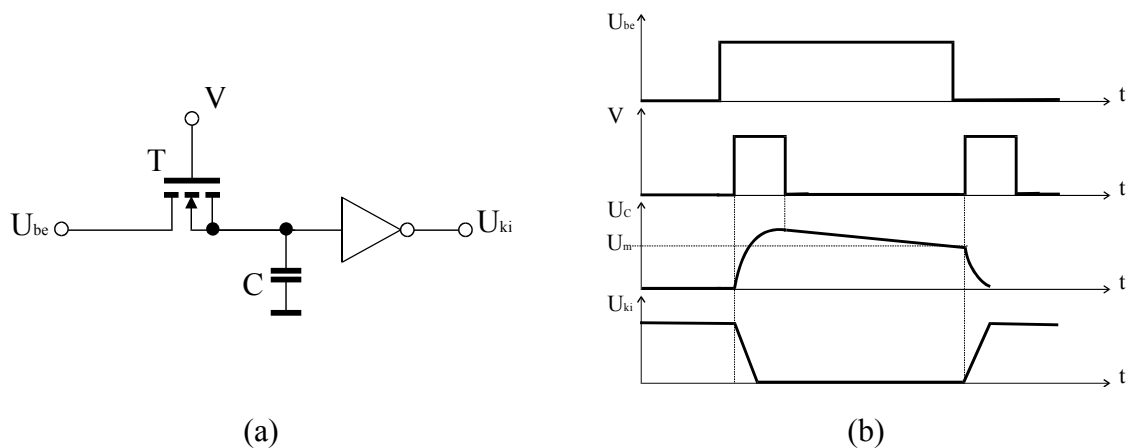
Belátható, hogy ha legalább egy bemenet H, akkor a kimenet L, ha viszont mindkét bemenet L, akkor mindkét (T_1 , T_2) tranzisztor zárt, és a kimenet H állapotú.

MOS logikával természetesen más típusú kapuk is létrehozhatók.

A MOS kapuk jellemző késleltetése n-csatornás esetben 4-20ns és p-csatornás megvalósítás esetében 20-100ns.

A teljesítményviszonyokat később fogjuk megvizsgálni, egyelőre csak annyit említünk meg, hogy a teljesítményfelvétel frekvenciafüggő.

Tanulmányozzuk az ún. MOS *dinamikus invertert*. Kapcsolása (a) és működési diagramja (b) a 7. ábrán látható.



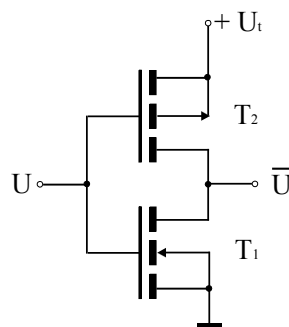
7. ábra. Dinamikus MOS inverter (a) és működési diagrammja (b)

Ha a bemeneti feszültség H szintű, a V engedélyező (vezérlő) jel a tranzisztoron keresztül a tagadó áramkörre juttatja, amelynek kimenete L lesz. Ezzel egy időben a C kondenzátor is H szintre telik. A V megszűnése után a kapu pn átmenetének visszaráma a kondenzátort kezdi kisütni. Ez a tagadó áramkör kimenetét mindaddig nem befolyásolja, amíg a feszültség el nem éri a kapu U_m billenési szintjét. Ha a bemeneten L szint van, akkor egy H szintet tároló kondenzátor a $V = H$ hatására kisül. A bemutatott emlékező tulajdonságon (a tagadó bemenete átmenetileg tárolja a töltést) alapulnak a dinamikus MOS áramkörök.

b) Komplementer MOS (CMOS) logika

Mint említettük, a MOS logika terhelő tranzisztora állandóan nyitva van és ez kedvezőtlenül befolyásolja a fogyasztást. Ha a kimenet H állapotában a tranzisztor lezárna, akkor a működési jellemzők megváltozása nélkül az áramfelvétel csökkenne.

Az elvet a CMOS NEM kapu szemlélteti (8. ábra).

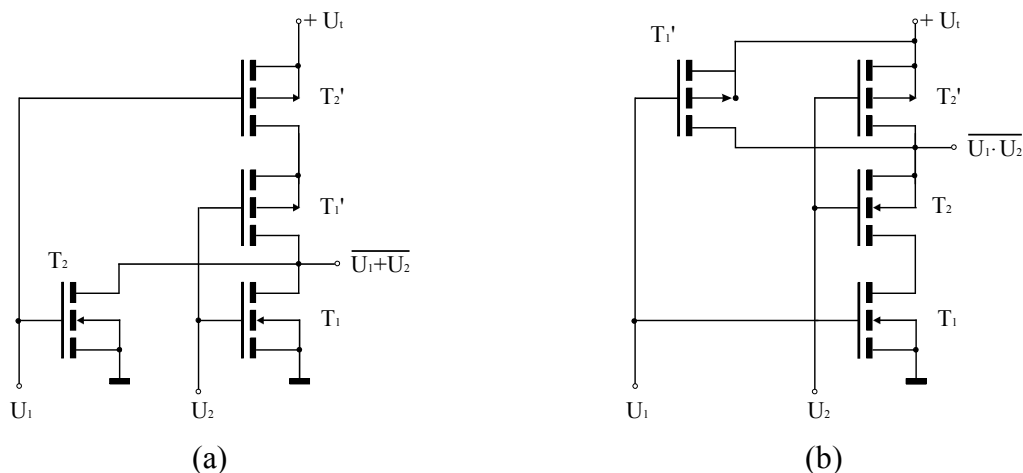


8. ábra. CMOS NEM kapu

Ha a bemenet szintje L, akkor T_1 lezár, T_2 pedig kinyit és a kimenet H állapotú lesz. Ha viszont a bemenet H szintű, akkor a T_2 zár és a T_1 nyit ki, a kimenet pedig L.

Megjegyezzük, hogy a tápfeszültség az elzáródási feszültségnél nagyobb kell legyen.

Hasonló elven működő CMOS párokkal könnyen létrehozható a VAGYNEM (9.a ábra) és az ÉSNEM (9.b ábra) kapcsolás.



9. ábra. CMOS VAGYNEM (a), ÉSNEM (b) kapu

A 9.a ábra alapján könnyen ellenőrizhető, hogy ha mindkét bemenet L, akkor T_1 , T_2 zárt, illetve a T_1 , T_2 tranzisztorok vezetnek, a kimenet pedig H. Ha bármelyik bemenet H, akkor a kimenet L-re vált.

Hasonlóan követhető a 9.b ábrán az ÉS-NEM kapcsolat magvalósulása. Megfigyelhető, hogy a kimenet csak akkor lesz L, ha mindkét bemenet H.

Vizsgáljuk meg, hogyan alakul a CMOS áramkörök teljesítményfelvétele. Több CMOS áramkör összekapcsolásakor a meghajtó áramkör minden billenésekor át kell töltenie a bemenetek kapacitásait. Példaként tekintsük a fordító kaput (8. ábra). Ha a kimenet H, akkor a nyitott T_2 tranzisztoron keresztül a következő fokozat kondenzátorát (terhelő kondenzátor) a tápfeszültség szintjére kell feltölteni. Ekkor a tápforrás $Q = C \cdot U_t$ töltést ad le, ahol C a MOS kapacitás. Ha a kimenet L-re vált, a terhelő kapacitást a nyitott T_1 tranzisztoron keresztül ki lehet sütni. Ilyenképp a tápforrás a bemeneti jel minden periódusánál Q töltésmennyiséget továbbít a föld felé. Az átlagos áram tehát $I = C \cdot U_t / T = fC \cdot U_t$. Innen egy nagyon nagy gyakorlati jelentőségű megállapítás következik, mégpedig az, hogy a CMOS áramkörök teljesítményfelvétele arányos a frekvenciával. Előfordulhat, hogy a CMOS kapu egyébként igen kicsi teljesítményfelvétele nagyfrekvencián nagyobb, mint a megfelelő TTL áramköré. Jellemzőnek tekinthető a $0,5 \mu\text{W/kHz/kapu}$ teljesítményfelvétel módosulás.

2.2. A bipoláris és CMOS logikai áramkörök összehasonlítása

Az összehasonlítás lényegében hasonló ahhoz a táblázathoz, amelyet a bipoláris és a MOS tranzisztor összehasonlításakor (1.7 pont) mutattunk be, ezért ennek megismétlésétől most eltekintünk.

Megjegyezzük, hogy eredetileg a MOS kapuk magasabb tápfeszültséget igényeltek mint a TTL kapuk. Eltérések mutatkoztak a H és L szintek tolerancia sávjait illetően is, amelynek következtében a MOS és TTL csatlakozások közé *illesztőket* (*interfészt*) kellett iktatni. E különbségek már régebb megszűntek és ma a MOS áramkörök nagy része TTL kompatibilis. Eltérések csak a paraméterek tekintetében észlelhetők az említett táblázat szerint.

A nagysebességű TTL kompatibilis MOS családok (High Speed CMOS-HC, Advanced High Speed CMOS-AHC) megszületésével a CMOS áramkörök sebessége (késleltetés) is lényegesen feljavult és nagyságrendben a TTL-hez hasonlóvá vált. Itt is alá kell húzzuk, hogy

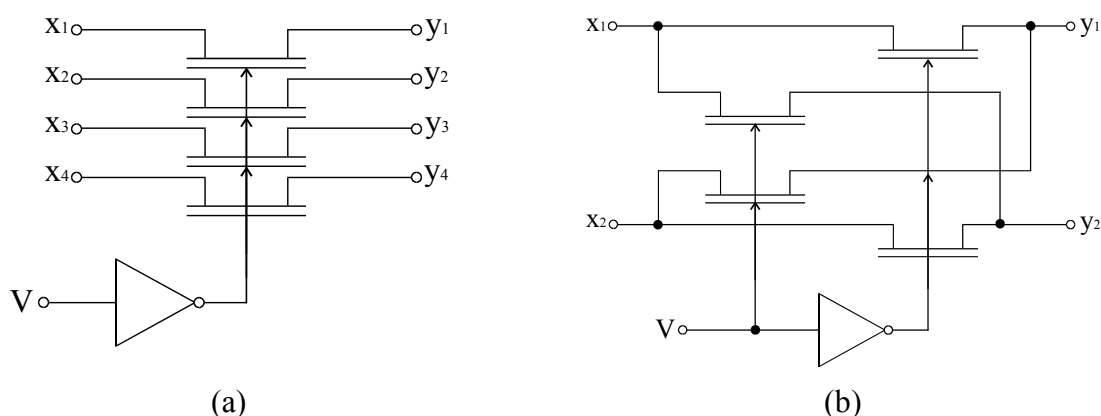
a CMOS áramkörök előnyös paramétereik és kedvezőbb gyártástechnológiájuk miatt valószínűleg további teret hódítanak és előbb-utóbb uralkodóvá válnak. Itt említhető meg, hogy a szakemberek az ABCMOS-t (Advanced Bipolar CMOS), vagy ABT-t (Advanced Bipolar CMOS Technology) tekintik a jövő egyik legígéretesebb integrált áramkör generációjának. Ez analóg és digitális jelfeldolgozásra kifejlesztett nagybonyolultságú bipoláris és CMOS technológiával készülő integrált áramkör család.

2.3. Töltéscsatolt logika

Töltéscsatolt eszközök (lásd 1.8 pont) felhasználásával logikai kapcsolások is létrehozhatók. Alapkapcsolások létrehozására nem alkalmas. Töltéscsatolt logikával csak bonyolult sorrendű hálózatokat készítenek. Ilyenek a soros hozzáférésű tároló, léptető regiszter, átviteli lánc stb.

2.4. Keresztrudas kapcsolólogika

Megnevezését a keresztrudas telefonközpontoktól kölcsönözte. E néhány különleges áramkört, viszonylag egyszerű logikával, azért fejlesztették ki, hogy nagysebességű jelátvitelt biztosítson két, egyébként nem csatolt eszköz között. Működési elve a 10. ábrán követhető.



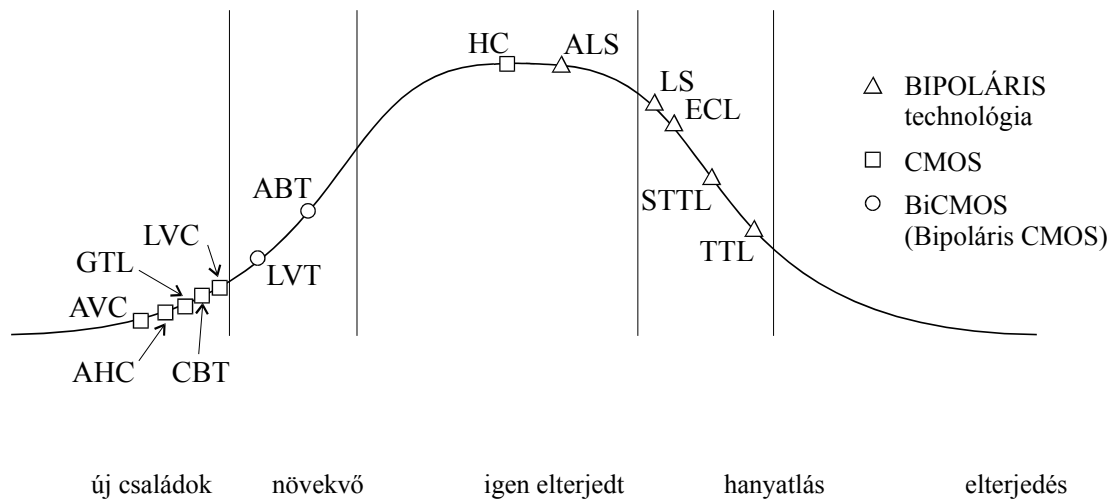
10. ábra. Keresztrudas kapcsoló, egyirányú (a), kétirányú (b)

Megfelelően egy időben vezérelve (V) több MOS tranzisztor kapuját a kimenetek (y) vagy a bemeneteket (x) követik, vagy pedig magas impedanciás állapotba kerülnek. Mindössze néhány típusú keresztrudas technológiával (Cross Bar Technology - CBT) készülő áramkört fejlesztettek ki, de szerepük igen hasznos a kapcsolómátrixok, színvezérlések, vagy leválasztó kapcsolások megvalósításában. Igen jónak mondható a működési sebessége, a késleltetés kisebb mint 0,25 ns.

A keresztrudas kapcsolólogika az átvivő kapuk továbbfejlesztésének tekinthető.

2.5. Logikai áramkör családok áttekintése

Érdekes áttekinteni az említett és néhány más MOS áramkör család elterjedtségét a 2000-es év szintjén (11. ábra). Ugyanakkor megfigyelhető az eddig igen népszerű bipoláris áramkörök hanyatlása, illetve a bipoláris CMOS eszközök térhódítása.



11. ábra. Logikai áramkörcsaládok helyzete

AVC - Advanced Very Low Voltage CMOS Logic, LVC - Low Voltage CMOS, LVT - Low Voltage Bipolar CMOS, GTL - Gunning Transceiver Logic

Irodalomjegyzék

- 1] Szittyá O.: *Bevezetés az elektronikába*, LSI, Budapest, 1998.
- 2] U.Tietze - C.Schenk: *Analóg és digitális áramkörök*, Műszaki, Budapest, 1981.
- 3] Kovács F.: *MOS integrált áramkörök*, Műszaki, Budapest, 1986.

Digitális rendszerek tervezése a VHDL nyelv segítségével

Dr. Baruch Zoltán, egyetemi tanár

Kolozsvári Műszaki Egyetem, Számítástechnika és Automatizálás Kar

1. Bevezetés

1.1. Hardver-leíró nyelvek

A hardver rendszerek klasszikus tervezése általában a következő részekből áll: a tervezési specifikációk leírása természetes nyelvben, a terv megvalósítása rajzokkal és a terv szimulálása. A jelenlegi hardver rendszerek bonyolultsága miatt szükség van új tervezési módszerekre. A tervezők számára előnyös a rendszer specifikációinak leírása egy hardver-leíró nyelv segítségével. Ennek a megközelítésnek a főbb előnyei a következők:

- A leírás szimulálásával lehetővé válik a tervezett rendszer működésének ellenőrzése a rendszer megvalósítása előtt;
- A leírás felhasználható egy szintézis program bemeneteként;
- A leírás a tervezett rendszer dokumentálását teszi lehetővé, ami technológiailag független;
- A hardver-leíró nyelvek magas szintű absztrakt specifikációt tesznek lehetővé;
- A típusok szigorú ellenőrzésével kiszűrhető a hibák nagy része.

1.2. A VHDL hardver-leíró nyelv

A VHDL (*VHSIC Hardware Description Language*) az ADA programozási nyelvre alapszik. A VHDL nyelv kifejlesztését az Egyesült Államok Védelmi Minisztériuma kezdeményezte, azzal a céllal, hogy egy szabványos nyelvet hozzon létre a VHSIC (*Very High Speed Integrated Circuit*) projekt részére. A nyelv első verzióját 1985-ben publikálta a Védelmi Minisztérium egy katonai szabványként (MIL-STD-454L), majd több módosítás után az IEEE szervezet szabványosította (IEEE 1076-1987 szabvány - VHDL'87). A nyelv egy újabb verzióját 1993-ban szabványosították (IEEE 1076-1993 szabvány - VHDL'93).

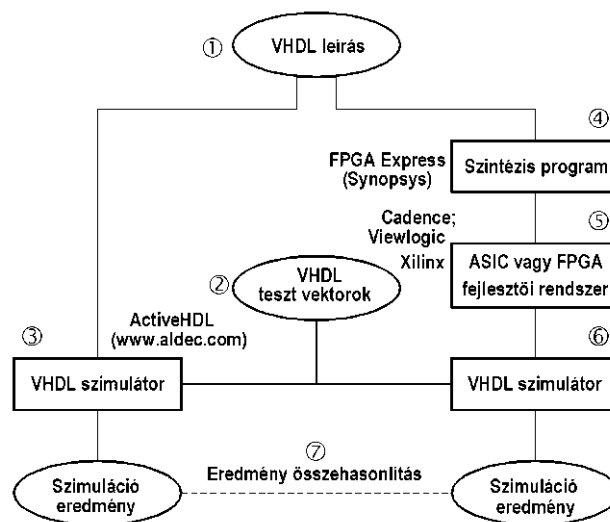
A VHDL nyelv főbb jellemzői a következők:

- Különböző leírási stílusokat tesz lehetővé: funkcionális leírások, adat-folyam típusú leírások és strukturális leírások.
- Különböző tervezési módszereket és modellezési technikákat tesz lehetővé: algoritmikus leírások, egyenletek, automaták.
- Különböző hardver technológiák használhatók, új primitív logikai típusok és technológiai attribútumok segítségével.
- Konkurens utasítások használhatók, amelyek egyidejű események modellezését teszik lehetővé.
- Új adattípusok definiálhatók.

1.3. A tervezési folyamat

A digitális rendszerek tervezési folyamata a VHDL nyelv segítségével (1. ábra) a következő lépésekből áll:

1. A rendszer modellezése egy VHDL leírással.
2. A modell kiegészítése tesztt vektorokkal.
3. A modell szimulálása egy VHDL szimulátor segítségével.
4. A VHDL leírás szintézise egy szintézis program segítségével, amely technológiailag független.
5. Egy ASIC (*Application Specific Integrated Circuit*) vagy FPGA (*Field-Programmable Gate Array*) fejlesztői rendszer felhasználása, egy bizonyos technológia szerint.
6. A generált hardver rendszer VHDL modelljének a szimulálása.
7. A két szimulációs eredmény összehasonlítása, amely alapján megállapítható, hogy a generált rendszer működése megegyezik-e az eredeti modell működésével.



1. ábra. A tervezési folyamat

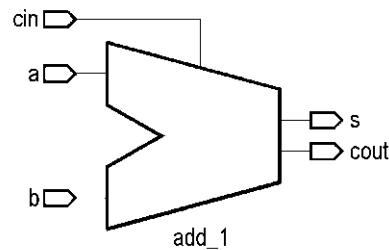
2. VHDL tervezési egységek

A tervezési egységek egy leírás alapelemei. Egy bizonyos tervezési egység egyetlen állományban kell szerepeljen, tehát nem lehet felosztani több állomány között. Egy állomány tartalmazhat viszont bármilyen számú tervezési egységet. A VHDL nyelv tervezési egységei feloszthatók *elsődleges* és ezeknek megfelelő *másodlagos* tervezési egységekre. A tervezési egységek a következők:

- *Entitás* (elsődleges) és *architektúra* (másodlagos);
- *Csomag deklaráció* (elsődleges) és *csomag törzs* (másodlagos);
- *Konfiguráció* (elsődleges).

2.1. Entitás

Az *entitás* egy elsődleges tervezési egység, amely egy teljes digitális rendszert, egy alrendszert vagy komponenst ábrázol. Egy entitás deklaráció a hardver rendszer interfészét határozza meg: a be- és kimeneteket (portokat), az adatok irányát (*in*, *out*), az adatok típusát és a paramétereket (generikusokat).



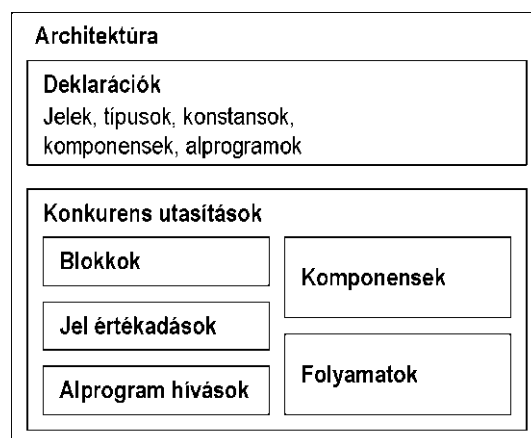
2. ábra. Az 1 bites összeadó grafikus szimbóluma.

A 2. ábrán szimbolizált 1 bites összeadónak megfelelő entitás a következő:

```
entity add_1 is
  port (a, b, cin: in bit;
        s, cout: out bit)
end entity add_1;
```

2.2. Architektúra

Az *architektúra* leírja a megfelelő entitás tartalmát vagy funkcióját. Minden architektúra tartalmazhat egy deklarációs részt és konkurens utasításokat. A deklarációs részben jelek, típusok, konstansok, komponensek és alprogramok definiálhatók. A konkurens utasítások lehetnek jel-értékadó utasítások, komponensek, folyamatok, alprogram hívások vagy blokkok. Az architektúra szerkezete a 3. ábrán látható.



3. ábra. Egy architektúra általános szerkezete

Az előbbi add_1 entitásnak megfelelő architektúra például a következő lehet:

```
architecture szintezis of add_1 is
    signal s1, c1, c2: bit;
begin
    s1 <= a xor b;
    c1 <= s1 and cin;
    c2 <= a and b;
    s  <= s1 xor cin;
    cout <= c1 or c2;
end szintezis;
```

2.3. Csomag deklaráció és csomag törzs

Egy *csomag deklaráció* tartalmazhat konstans deklarációkat, típus deklarációkat, komponens deklarációkat és alprogram deklarációkat. Egy csomagban deklarált információk felhasználhatók utólag több entításban és architektúrában. A csomag törzs a deklarált alprogramok törzsét tartalmazza.

A következő csomag deklaráció egy típus, altípus, jel, komponens és funkció deklarációját tartalmazza.

```
package pelda is
    type byte is range 0 to 255;
    subtype nibble is byte range 0 to 15;
    constant end_k: byte := 255;
    signal op1: byte;
    component adder is
        port (a, b: in byte;
              c:   out byte;
              over: out boolean);
    end component;
    function f_mul (a, b: nibble) return byte;
end pelda;
```

3. Leírási stílusok

Egy architektúra leírásában a VHDL nyelv több stílus felhasználását teszi lehetővé: funkcionális leírás, adatfolyam-típusú leírás (*dataflow*) és strukturális leírás. A különböző stílusokat egy 4 bites egyenlőségi komparátorral mutatjuk be, amelynek bemenetei a[3:0] és b[3:0] vektorok, a kimenet pedig eq, amely logikai '1' lesz, ha a két bemenet egyenlő.

3.1. Funkcionális leírások

A funkcionális leírások algoritmikus leírások, ezért a magas szintű programozási nyelvekre hasonlítanak. Egy bizonyos entitás struktúrájának a meghatározása helyett, a funkcionális leírások szekvenciális utasításokból és folyamatokból állnak, amelyeknek a végrehajtása az

entitás működését modellezi. A funkcionális leírások előnye, hogy a tervező nem kell figyeljen az entitás megvalósítására kapuk szintjén, hanem az entitás működésének a modellezésére koncentrálhat.

A következő példa az említett egyenlőségi komparátor funkcionális leírását tartalmazza.

```
entity komp_4 is
    port (a, b: in bit_vector (3 downto 0);
          eq: out bit);
end komp_4;

architecture funkcionalis of komp_4 is
begin
    komp: process (a, b)
    begin
        if a = b then
            eq <= '1';
        else
            eq <= '0';
        end if;
    end process komp;
end funkcionalis;
```

3.2. Adatfolyam-típusú leírások

Az egyenlőségi komparátor egy lehetséges adatfolyamat-típusú leírása a következő:

```
entity komp_4 is
    port (a, b: in bit_vector (3 downto 0);
          eq: out bit);
end komp_4;

architecture adat_f of komp_4 is
begin
    eq <= '1' when (a = b) else '0';
end adat_f;
```

Az adatfolyam-típusú leírások konkurrens utasításokat tartalmaznak folyamatok és szekvenciális utasítások helyett. Az előbbi példában az eq jel értékadó utasítása egy konkurrens utasítás.

A következő példa az egyenlőségi komparátor egy más architektúráját mutatja be, amely logikai egyenleteket tartalmaz.

```
architecture adat_f of komp_4 is
```

```

begin
    eq <=      not (a(0) xor b(0))
              and not (a(1) xor b(1))
              and not (a(2) xor b(2))
              and not (a(3) xor b(3));
end adat_f;

```

3.3. Strukturális leírások

A strukturális leírások kapcsolási listákat tartalmaznak, amelyekben a komponensek jelekkel vannak összekötve. Ezek a leírások hierarchikusak; a különböző komponensek csomagokban definiálhatók, amelyeket könyvtárakban lehet tárolni. A strukturális leírások főleg a bonyolult tervek esetén előnyösek, amelyek több részre oszthatók. Minden rész tervezése függetlenül valósítható meg.

A következő strukturális leírásban szereplő `xnor2` és `and4` komponenseket a `kapuk` nevű csomagban kell definiálni, és ezt a csomagot a `munka` nevű könyvtárban kell tárolni.

```

entity komp_4 is
    port (a, b: in bit_vector (3 downto 0);
          eq: out bit);
end komp_4;

use munka.kapuk.all;

architecture strukturalis of komp_4 is
    signal x: bit_vector (3 downto 0);
begin
    u0: xnor2 port map (a(0), b(0), x(0));
    u1: xnor2 port map (a(1), b(1), x(1));
    u2: xnor2 port map (a(2), b(2), x(2));
    u3: xnor2 port map (a(3), b(3), x(3));
    u4: and4 port map (x(0), x(1), x(2), x(3), eq);
end strukturalis;

```

4. VHDL szimulátorok

A VHDL szimulátorok *események által vezérelt szimulálást* végeznek. Egy esemény egy bizonyos jel állapotának módosítását jelenti. Az események által vezérelt szimulálás alapfogalmai a következők: a szimulációs idő, az események feldolgozása és a delta késés.

A szimulálás alatt a szimulátor nyilvántartja a szimulációs időt, amely azt az időpontot jelenti, amikor kezdetét veszi a szimulálás, és nem a szimuláláshoz szükséges időt. A szimulációs időt általában egy időegységnek a sokszorosában mérjük, amelyet felbontási határnak nevezünk. A felbontási határ általában 1 ps.

Egy esemény feldolgozásánál a szimulátor újraértékel minden utasítást, amelynek bemenete az a jel, amely az eseményt okozta (tehát, azokat az utasításokat, amelyek „érzékenyek” az

illető jelre). Az utasítások újraértékelése más jelek módosítását és más események létrehozását eredményezi.

A következő példa egy egyszerű folyamatot tartalmaz, amely egyetlen jel-értékadó utasításból áll.

```
p1: process (a, b, c)
begin
    x <= a and b and c;
end process p1;
```

Az a, b vagy c jel állapotának módosítása esetén, az x jel újraértékelődik és a szimulátor egy új értéket számít ki az x jelnek. A szimulátor a jelenlegi szimulációs időpontra egy esemény végrehajtását tervezi meg, amely az x jel módosításából áll. Potenciális problémák keletkezhetnek abban az esetben, ha az x jelet ugyanabban az időpontban kell aktualizálni, mint azok a jelek egyike, amelyből az x jel lesz generálva. Az ilyen problémák kiküszöbölésére a VHDL bevezeti a *delta késést* (δ). A delta késés egy végtelenül kis késés, amely egy *delta ciklus* végrehajtását eredményezi a jelek értékének aktualizálása végett. Tehát, az előbbi értékadó utasítás szemantikája a következő: a jobboldali kifejezés értéke a jelenlegi szimulációs időben, T_c , egy delta késéssel a jelenlegi szimulációs idő után adódik át az x jelnek, tehát a $T_{x+\delta}$ időpontban.

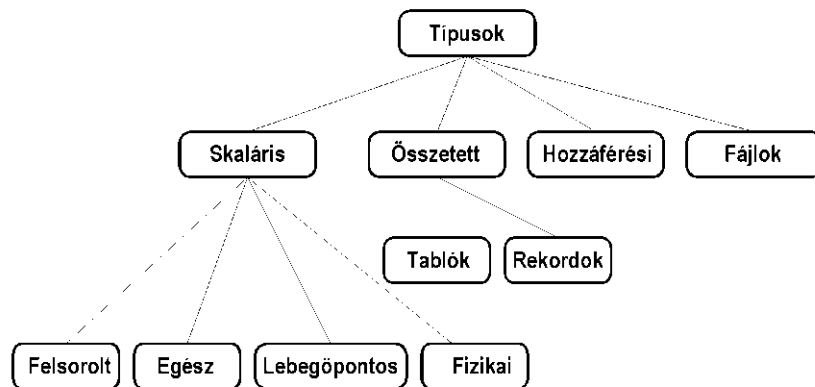
A $T_{x+\delta}$ szimulációs időben feldolgozásra kerül minden esemény, amely erre az időpontra volt megtervezve, és új események jönnek létre. Az új események egy része a jelenlegi szimulációs időpontra lesz tervezve, ami azt jelenti, hogy a $T_{x+2\delta}$ szimulációs időpontban kerül végrehajtásra. Egy újabb delta késés után a szimulátor újból feldolgozza az eseményeket, ami által új események jönnek létre, és így tovább, amíg nem terveznek más eseményeket a jelenlegi szimulációs időre. Ekkor a szimulátor megállapítja a következő szimulációs időt, és ennek megfelelően növeli a szimulációs időt.

5. Típusok

5.1. A típusok osztályozása

A 4. ábra a VHDL nyelv típusainak osztályozását mutatja be.

A *skaláris típusok* egyszerű típusok, tehát nem tevődnek össze más elemekből. A skaláris típusok a következők: felsorolt típusok, egész típusok, lebegőpontos típusok és fizikai típusok. Az *összetett típusok* a tablókat és rekordokat tartalmazzák. A *hozzáférési típusok* hasonlóak a programozási nyelvekből ismeretes mutatókkal (pointerek), és lehetővé teszik a tár dinamikus lefoglalását egy objektum részére és utólag a tár felszabadítását. A hozzáférési típusokat nem lehet szintézisre használni, csak szimulálásra. Egy *fájl típus* lehetővé teszi egy fájl típusú objektum deklarációját, amely egy bizonyos típusú értéke sorozatát tartalmazza. A fájl típusok csak szimulálásra használhatók.



4. ábra. A VHDL nyelv típusainak osztályozása

5.2. Standard típusok

Bizonyos típusok definiáltak a VHDL nyelv által. Ezek a típusok egy standard nevű csomagban találhatók. Az 1. táblázat bemutatja a standard csomagban definiált adattípusokat és a kategóriát amelyhez tartoznak.

Típus	Kategória	Felhasználás szintézisre
bit	Felsorolt típus	✓
boolean	Felsorolt típus	✓
character	Felsorolt típus	✓
integer	Egész típus	✓
natural	Az integer altípusa	✓
positive	Az integer altípusa	✓
real	Lebegőpontos típus	-
time	Fizikai típus	-
delay_length	A time altípusa	-
string	Character típusú vektor	✓
bit_vector	Bit típusú vektor	✓

1. táblázat. A standard csomagban definiált típusok

Az előbbi típusokon kívül, léteznek más standard típusok, amelyek más csomagokban vannak definiálva. Ezek közül a legfontosabb csomag a `std_logic_1164`, amely az IEEE 1164-es számú szabvány része. Az ebben a csomagban definiált típusokat a 2. táblázat tartalmazza.

Típus	Kategória	Felhasználás szintézisre
std_ulogic	Felsorolt típus	✓
std_logic	A std_ulogic altípusa	✓
std_ulogic_vector	std_ulogic típusú vektor	✓
std_logic_vector	std_logic típusú vektor	✓

2. táblázat. A std_logic_1164 csomagban definiált típusok

5.3. Egyszerű típusok

5.3.1. Felsorolt típusok

Egy felsorolt típus egy bizonyos objektum lehetséges értékeit sorolja fel egy lista formájában. A lista minden eleme egy szimbolikus név vagy egy karakter lehet. Például, a következő típus négy lehetséges logikai értéket sorol fel:

```
type log4 is ('X', '0', '1', 'Z');
```

Az IEEE 1076 számú szabvány által definiált legfontosabb felsorolt típusok, amelyek felhasználhatók szintézisre is, a következők: bit, boolean és character. Az IEEE 1164 számú szabvány definiálja a std_ulogic típust és ennek több altípusát, amelyek ugyancsak felhasználhatók szintézisre.

A bit típus két értéket definiál, amelyeket a '0' és '1' karakterekkel jelölünk. A definíció a következő:

```
type bit is ('0', '1');
```

A boolean típus a következőképpen definiált:

```
type boolean is (FALSE, TRUE)
```

Az összehasonlítási műveletek eredményei boolean típusúak. Szintézisnél a FALSE érték logikai 0-át jelent, míg a TRUE érték logikai 1-et.

A character típus ASCII karaktereket definiál. A nyelv VHDL'87 verziójában a character típus a 7-bites ASCII karakterszettet definiálja, míg a VHDL'93 verzióban az ISO 8859-1 (Latin-1) karakterszettet. A kontroll karakterek, amelyeknek kódja 00 és 1Fh között van, szimbolikusan vannak jelölve, például: BEL (07h), BS (08h), HT (09h), CR (0Dh).

5.3.2. Egész típusok

A legfontosabb egész típus az integer, amely a standard csomagban definiált:

```
type integer is range -2147483648 to +2147483647;
```

Az integer típus két altípusa a natural és a positive:

```
subtype natural is integer range 0 to integer'high;  
subtype positive is integer range 1 to integer'high;
```

Az integer'high érték az integer típus legnagyobb értékét jelenti. Egy natural vagy positive típusú jel alaptípusa az integer, de a lehetséges értékek a természetes számokra, illetve a pozitív számokra korlátozódnak.

Az integer típuson és ennek altípusain kívül a felhasználó tetszőlegesen definiálhat más típusokat. Az így definiált típusok főleg a szintézisre használt jelek és változók lehetséges értékeinek korlátozására hasznosak.

5.3.3. Fizikai típusok

A fizikai típusok nemcsak az objektumok értékeit jelölik meg, hanem az értékek mértékegységeit is. Egy fizikai típus deklarációjában két típusú mértékegységet lehet megadni: egy elsődleges mértékegységet és több másodlagos mértékegységet; az utóbbiak az elsődleges mértékegység függvényében vannak kifejezve.

A VHDL nyelv egyedüli előre definiált fizikai típusa a time, amelynek elsődleges mértékegysége a femtoszekundum (fs). A time típus definíciója a következő:

```
type time is range -2147483647 to +2147483647
units
    fs;
    ps  = 1000 fs;
    ns  = 1000 ps;
    us  = 1000 ns;
    ms  = 1000 us;
    sec = 1000 ms;
    min = 60 sec;
    hr  = 60 min;
end units;
```

5.4. Összetett típusok

5.4.1. Tablók

Egy *tabló* típusú objektum több elem gyűjteményéből áll, amelyben minden elem alaptípusa ugyanaz. A különböző elemek egy vagy több index alapján érhetők el, ahol az indexek száma a tabló dimenziójától függ.

A VHDL nyelvben két előre definiált tabló van, és mindkettő egy dimenziójú (vektor). A bit_vector tabló bit típusú, a string tabló pedig character típusú. E két tabló definíciója a következő:

```
type bit_vector is array (natural range <>) of bit;
type string is array (positive range <>) of character;
```

A bit_vector vagy string típusú jelek deklarációjánál a lehetséges értékek tartományát korlátozni kell, mint például a következő deklarációkban:

```
signal data: bit_vector (31 downto 0);
signal mes: string (1 to 40);
```


A data és a mes vektorok elemei data(31), ..., data(0), valamint mes(1), ..., mes(40).

Az IEEE 1164 számú szabvány két tablót definiál a std_logic_1164 csomagban, a std_ulogic típus, illetve a std_logic altípus alapján. A szabvány ugyanakkor operátorokat és konverzió funkciókat definiál a tablók számára. A két tábló definíciója a következő:

```
type std_ulogic_vector is array (natural range <>) of std_ulogic;  
type std_logic_vector is array (natural range <>) of std_logic;
```

Az IEEE 1076.3 számú szabvány két új csomagot definiál, a numeric_bit és a numeric_std csomagokat. Mindkét csomag tablók formájában definiál két típust, az unsigned és a signed típusokat. A két csomagot nem lehet egyszerre használni, mivel a két típust különböző módon definiálja. A numeric_bit csomagban a definíciók a következők:

```
type unsigned is array (natural range <>) of bit;  
type signed is array (natural range <>) of bit;  
A numeric_std csomagban a két típus definíciója a következő:  
type unsigned is array (natural range <>) of std_logic;  
type signed is array (natural range <>) of std_logic;
```

Az unsigned típus egy előjel nélküli bináris egész számot jelent, míg a signed típus egy előjeles bináris egész számot, kettes komplement ábrázolásban.

5.4.2. Rekordok

Egy *rekord* típusú objektum több elem gyűjteményéből áll, amelyben az elemek alaptípusa különböző lehet. Egy bizonyos elemhez a rekord neve és az elem neve által lehet hozzáférni. A következő példában egy két elemes rekord és egy ilyen rekord típusú jel van definiálva.

```
type complex is record  
    re: integer;  
    im: integer;  
end record;  
  
signal a: complex;
```

Az a jel reális részének értékadása a következőképpen történhet:

```
a.re <= 0;
```

Egy rekord típusú jel minden elemének az értékadása egy *aggregátum* felhasználásával lehetséges, például:

```
a <= (re => 0, im => 0);
```

vagy:

```
a <= (0, 0);
```

6. Attribútumok

Egy attribútum információkat szolgáltat egy bizonyos típusról vagy egy bizonyos típusú objektum értékeiről. A VHDL nyelv előre definiált attribútumai típusokra, tablókra és jelekre vonatkoznak. A típusokra vonatkozó attribútumok egy része felhasználható a tablók attribútumaiként is. A nyelv ezen attribútumain kívül léteznek a szimulátor által vagy a szintézis rendszer által definiált attribútumok. Ugyanakkor a felhasználó is definiálhat attribútumokat.

6.1. Típusokra vonatkozó attribútumok

A VHDL nyelv a következő attribútumokat definiálja a felsorolt és egész típusok részére (ezeknek egy része használható a fizikai típusok részére is):

<code>típus'left</code>
<code>típus'right</code>
<code>típus'low</code>
<code>típus'high</code>
<code>típus'pred (x)</code>
<code>típus'succ (x)</code>
<code>típus'leftof (x)</code>
<code>típus'rightof (x)</code>
<code>típus'pos (x)</code>
<code>típus'val (x)</code>

A 'left és 'right attribútumok egy típus baloldali, illetve jobboldali értékének meghatározására használhatók. Egy típus legkisebb és legnagyobb értéke meghatározható a 'low, illetve 'high attribútummal. A 'pred és 'succ attribútumokkal meghatározható az x érték elődje, illetve utódja (tehát, az x értékhez viszonyított közvetlenül kisebb, illetve közvetlenül nagyobb érték a típuson belül). A 'leftof és a 'rightof attribútumok alkalmazásával az x argumentum baloldali értéke, illetve jobboldali értéke kapható meg. A 'pos attribútum az x argumentumot, amely egy felsorolt típus értéke, egy egész számra konvertálja, amely az illető érték pozíciójával egyenlő a típuson belül. A 'val attribútum az x argumentumot, amely egy érték pozíciója egy felsorolótípuson belül, az ennek a pozíciónak megfelelő értékre konvertálja.

6.2. Tablókra vonatkozó attribútumok

A tablókra vonatkozó attribútumok információkat szolgáltatnak egy tabló dimenziójáról, tartományáról vagy indexeléséről. A legfontosabb attribútumok, amelyek tablókra vonatkoznak, a következők:

<code>tabló'left</code>
<code>tabló'right</code>
<code>tabló'low</code>
<code>tabló'high</code>
<code>tabló'range</code>
<code>tabló'reverse_range</code>
<code>tabló'length</code>

A 'left attribútum egy tábló indexének a baloldali határértékét téríti vissza, míg a 'right attribútum egy tábló indexének a jobboldali határértékét téríti vissza. Abban az esetben, ha a tábló többdimenziójú, használhatók a 'left (n), illetve 'right (n) formák, ahol n az index számát jelenti. A 'low és 'high attribútumok egy tábló indexének a legkisebb értékét, illetve a legnagyobb értékét térítik vissza. Egy többdimenziójú tábló esetén, a 'low (n), illetve 'high (n) formák a tábló n -edik indexének a legkisebb értékét, illetve a legnagyobb értékét térítik vissza. A 'range attribútum egy tábló indexének tartományát téríti vissza, míg a 'reverse_range attribútum egy tábló indexének fordított tartományát téríti vissza. Hasonlóképpen a többi attribútummal, használhatók a 'range (n), illetve a 'reverse_range (n) formák is. A 'length attribútum egy tábló elemeinek számát téríti vissza.

6.3. Jelekre vonatkozó attribútumok

A jelekre vonatkozó attribútumok információkat szolgáltatnak a jelek módosításáról. A legfontosabb attribútumok, amelyek jelekre vonatkoznak, a következők:

```
jel'event
jel'last_value
jel'last_event
jel'delayed (t)
jel'stable (t)
```

Az 'event attribútum a TRUE értéket téríti vissza, ha az illető jel módosult a jelenlegi delta ciklus alatt; ellentétes esetben, az attribútum a FALSE értéket téríti vissza. Az 'event attribútum nagyon hasznos egy órajel éleinek detektálására. Például, a clk órajel felmenő élének detektálására a következő utasítás használható:

```
if (clk'event and clk = '1') then ... end if;
```

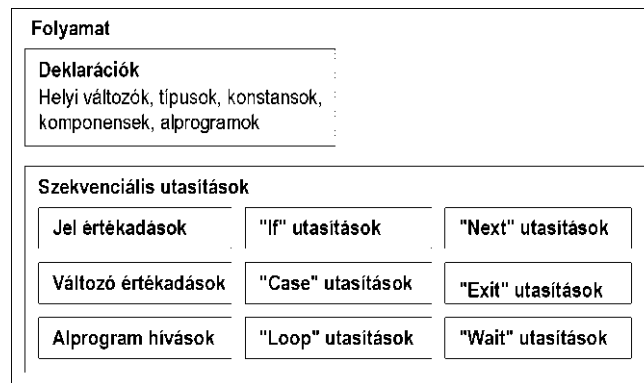
A 'last_value attribútum egy jel előző értékét téríti vissza, tehát az utolsó esemény előtti értéket. A 'last_event attribútum az illető jel utolsó eseményétől eltelt időt téríti vissza. Ez az attribútum a szimulálásnál hasznos, mivel lehetővé teszi a különböző időzítések tesztelését. A 'delayed attribútum egy új jelet hoz létre, amely az eredeti jel t időegységgel megkésett változata. A 'stable attribútum egy boolean típusú jelet hoz létre, amely TRUE értékű, ha a megadott t idő alatt az illető jel nem módosult.

7. Konkurens utasítások

7.1. Folyamatok

A folyamatok konkurens utasítások, tehát párhuzamosan futnak más konkurens utasításokkal. Minden folyamat viszont szekvenciális utasításokat tartalmaz, amelyek a megadott sorrendben futnak. Egy folyamat deklarációja egy szekvenciális részt határol el azon az architektúrán belül, amelyben a deklaráció megjelenik.

Egy folyamat általános szerkezetét az 5. ábra szemlélteti.



5. ábra. Egy folyamat általános szerkezete

A folyamat deklarációs része a `process` és `begin` kulcsszavak között van elhelyezve. Ebben a részben típusok, konstansok, változók, komponensek és alprogramok deklarálhatók. Az itt deklarált objektumok csak a folyamaton belül használhatók. Egy folyamat keretében nem deklarálhatók jelek.

Egy folyamat csak szekvenciális utasításokat tartalmazhat. A következő példában a `p1` folyamat két értékadó utasítást tartalmaz.

```
p1: process (a, b, c)
begin
  x <= a and b and c;
  y <= a xor b;
end process p1;
```

Az előbbi folyamat tartalmaz egy úgynevezett érzékenységi listát, amely az `a`, `b` és `c` jelekből áll. Amikor a listában szereplő jel bármelyike módosul, a folyamat utasításai végrehajtódnak, hasonló módon egy program utasításaihoz. Egy programozási nyelvtől eltérően, az `end process` kulcsszavak nem jelentik a folyamat végrehajtásának végét. Egy folyamat végtelen ciklusban fut; az utolsó utasítás után a folyamat végrehajtása felfüggesztődik, amíg az érzékenységi listában szereplő jelek valamelyike módosul.

Ha az érzékenységi lista hiányzik, a folyamat nem áll le. Ebben az esetben, a folyamat egy `wait` utasítást kell tartalmazzon, amely a folyamat végrehajtását felfüggeszti egy esemény megjelenéséig vagy egy feltétel teljesítéséig. A következő folyamat egy `wait` utasítást tartalmaz egy érzékenységi lista helyett.

```
p1: process
begin
  x <= a and b and c;
  y <= a xor b;
  wait on a, b, c;
end process p1;
```

A szintézis szempontjából kombinatorikus és szekvenciális folyamatokat lehet megkülönböztetni. Az egyedüli különbség közöttük, hogy a szekvenciális folyamatoknál a kimenő jelek bistabilokba vagy regiszterekbe íródnak.

A következő példa egy kombinatorikus folyamatot tartalmaz, amely egy AND kapunak felel meg.

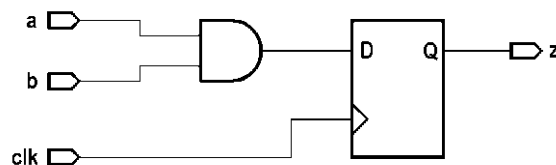
```
p2: process
begin
    wait on a, b;
    z <= a and b;
end process p2;
```

Azért, hogy egy folyamat egy kombinatorikus áramkört modellezzen, a folyamat érzékenységi listájának tartalmaznia kell minden bemenő jelet.

Az előbbi folyamatnak a következőképpen módosított változata szekvenciális folyamatként értelmezhető:

```
p3: process
begin
    wait until clk = '1';
    z <= a and b;
end process p3;
```

A p3 folyamat szintézise a 6. ábrában látható áramkört eredményezi.



6. ábra Egy szekvenciális folyamat szintézisének eredménye

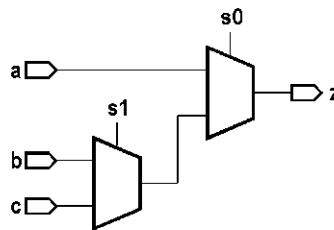
7.2. Feltételes jel-értékadó utasítás

A feltételes jel-értékadó utasítás egy when .. else konstrukció. Ez az utasítás az if szekvenciális utasításnak felel meg. A feltételes jel-értékadó utasítást a következő példával szemléltetjük:

```
z <= a when s0 = '1' else
    b when s1 = '1' else
    c;
```

A feltételek a leírt sorrendben kerülnek kiértékelésre. Az első teljesített feltételnek megfelelő kifejezés értéke lesz az, amely kiválasztódik a jel-értékadásra. Hardver szempontból az utasítás több sorba kötött multiplexernek felel meg, ahol az első feltétel a kimenethez

legközelebb levő multiplexert vezérli. Az előbbi utasításnak megfelelő áramkör a 7. ábrában látható.



7. ábra. Egy feltételes jel-értékadó utasításnak megfelelő áramkör

7.3. Szelektív jel-értékadó utasítás

Egy szelektív jel-értékadó utasítást egy `with .. select .. when` konstrukcióval lehet megvalósítani. Ebben az esetben is, az utasítás lehetővé teszi egy kifejezés kiválasztását több kifejezés közül. A különbség a feltételes jel-értékadó utasítással szemben az, hogy a szelektív jel-értékadó utasítás egyetlen feltételt használ. Ez az utasítás a `case` szekvenciális utasításnak felel meg.

A következő példa egy xor kaput ír le egy `with .. select .. when` konstrukció segítségével.

```
-- xor kapu (x <- a xor b)
architecture a_xor2 of xor2 is
    signal t: std_logic_vector (1 downto 0);
begin
    temp := a & b;
    with temp select
        x <= '0' when "00",
            '1' when "01",
            '1' when "01",
            '0' when "11";
        '0' when others;
end a_xor2;
```

8. Szekvenciális utasítások

8.1. If utasítás

Az `if` utasítás egy utasítás sorozatot választ ki végrehajtás céljából több utasítás sorozatból, a sorozatnak megfelelő feltétel alapján. Minden feltétel egy `boolean` típusú kifejezés kell legyen. Az `if` utasítást a következő példával szemléltetjük:

```
process (a, b, c, s0, s1)
```

```

begin
  if s0 = '1' then
    z <= a;
  elsif s1 = '1' then
    z <= b;
  else
    z <= c;
  end if;
end process;

```

Először az `if` kulcsszó utáni feltételt értékelik ki és, ha a feltétel igaz, lefut a megfelelő utasítás. Ellenkező esetben, ha jelen van az `elsif` kulcsszó, kiértékelődik az ennek megfelelő feltétel és, ha a feltétel igaz, lebonyolódik a megfelelő utasítás. Ha a feltétel nem igaz és más `elsif` kulcsszó is jelen van, folytatódik az ezeknek megfelelő feltételek kiértékelése. Ha egyik feltétel sem igaz, az `else` kulcsszónak megfelelő utasítás kerül végrehajtásra, ha ez a kulcsszó jelen van.

Az `if` utasítás hardver megfelelője, hasonlóan a feltételes jel-értékadó utasítás esetében, egy multiplexer. Az előbbi példában leírt `if` utasítás a feltételes jel-értékadó utasításnál illusztrált `when .. else` konstrukciónak felel meg, így a hardver megfelelője is ugyanaz (7. ábra).

8.2. Case utasítás

Hasonlóan az `if` utasításhoz, a `case` utasítás is kiválaszt végrehajtás céljából egy utasítás sorozatot egy kifejezés értékének függvényében. Az `if` utasítástól eltérően azonban, a kiválasztás nem egy boolean típusú kifejezés szerint történik, hanem egy felsorolt vagy egész típusú jel, változó vagy kifejezés szerint. A `case` utasítás akkor előnyös, amikor nagyszámú alternatívából lehet választani.

A következő példa a már használt `xor` kaput írja le egy `case` utasítás segítségével.

```

-- xor kapu (a xor b -> x)
process (a, b)
  variable t: std_logic_vector (1 downto 0);
begin
  temp := a & b;
  case temp is
    when "00" => x <= '0';
    when "01" => x <= '1';
    when "10" => x <= '1';
    when "11" => x <= '0';
    when others => x <= '0';
  end case;
end process;

```

Egy case utasítás hardver megfelelője ugyancsak egy multiplexer. Az if utasítástól eltérően, a case utasítás esetében lehetséges a feltételek optimalizálása, aminek eredményeképpen a létrehozott áramkörök egyszerűbbek.

8.3. Loop utasítások

A loop utasítások egy utasítássorozat ismételt végrehajtását teszik lehetővé, például egy tábló minden elemének a feldolgozását. A VHDL nyelvben három típusú loop utasítás létezik: az egyszerű loop utasítás, a while loop utasítás és a for loop utasítás.

Az egyszerű loop utasítás az adott utasítássorozat ismétlését eredményezi egy végtelen ciklusban. Ebben az esetben az egyedüli lehetőség az utasítássorozat befejezésére egy exit utasítás használata.

A while loop utasítás addig ismétli a megadott utasítássorozatot, amíg egy bizonyos feltétel igaz. Amikor a feltétel nem teljesül, az end loop kulcsszavak után levő utasítás végrehajtására kerül sor. A while loop utasítás egy példája a következő:

```
while szint = '1' loop
    n := n + 1;
    wait until clk = '0';
end loop;
```

A for loop utasítás egy megadott utasítássorozatot addig ismétel, amíg egy számláló a megfelelő tartományban van. Egy iteráció elvégzése után a számláló az illető tartomány következő értékét veszi fel. A következő for loop utasítás az 1 és 10 közötti egész értékek négyzetét számítja ki és ezeket a négyzet táblóban helyezi el.

```
for i in 1 to 10 loop
    negyzet (i) <= i * i;
end loop;
```

8.4. Next utasítás

A next utasítás egy loop utasítás törzsén belül helyezhető el. A next utasítás elérésénél a jelenlegi iteráció végrehajtása befejeződik. A végrehajtás a loop utasítás törzsének elején folytatódik feltétel nélkül, ha a when kulcsszó nincs jelen, vagy a when kulcsszóval megadott feltétel teljesítése után.

A next utasítás egy if utasítás helyett használható egy utasítássorozat feltételes végrehajtására. A következő példa egy for loop utasítást mutat be, amely egy vektor '1'-es bitjeinek számlálását végzi el egy next utasítás felhasználásával.

```
for i in v'range loop
    next when v(i) = '0';
    n := n + 1;
end loop;
```

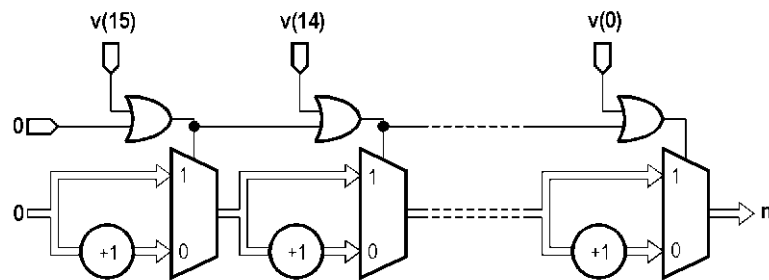

8.5. Exit utasítás

Az exit utasítás egy loop utasítás végrehajtásának teljes leállítását eredményezi. Általában a végrehajtás leállítása egy feltételtől függ, amit a when kulcsszóval lehet megadni.

A következő példa egy for loop utasítást mutat be, amely egy vektor elején található '0'-ás bitek számlálását végzi el egy exit utasítás felhasználásával.

```
for i in v'reverse_range loop
  exit when v(i) = '1';
  n := n + 1;
end loop;
```

Az előbbi leírás szintézise után a 8. ábrában látható áramkör jön létre. Az exit utasítást az OR kapukkal választjuk meg, amelyek a multiplexerek azon bemeneteit választják ki, amelyek nem tartalmazzák az inkrementáló áramkört, abban az esetben, ha az exit utasítás feltétele teljesül.



8. ábra. Egy exit utasítást tartalmazó for loop utasításnak megfelelő áramkör

Irodalomjegyzék

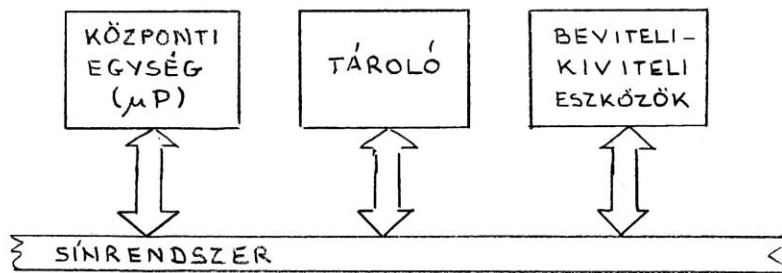
- 1] Mazor S, Langstraat, P: *A Guide for VHDL*, Kluwer Academic Publishers, 1993.
- 2] Perry, D. L.: *VHDL*, McGraw-Hill, 1991.
- 3] Rushton, A.: *VHDL for Logic Synthesis*, McGraw-Hill Europe, 1995.
- 4] Skahill, K.: *VHDL for Programmable Logic*, Addison-Wesley Publishing Co., 1996.
- 5] ActiveHDL dokumentáció, Aldec Inc.
- 6] FPGA Express dokumentáció, Synopsys Inc.
- 7] Baruch, Z. F.: *Structura sistemelor de calcul cu aplicații*, Todesco, Cluj, 2001.

A központi egység

Dr. Buzás Gábor, egyetemi tanár

Babeş-Bolyai Tudományegyetem, Fizika Kar

A számítógép legegyszerűbb és legáltalánosabb tömbvázlata mindössze a *központi egységből*, a *tárolóból* és a *beviteli-kiviteli eszközökből* áll, amelyek a *sínrendszeren* keresztül vannak egymással kétoldalú kapcsolatban (1. ábra).



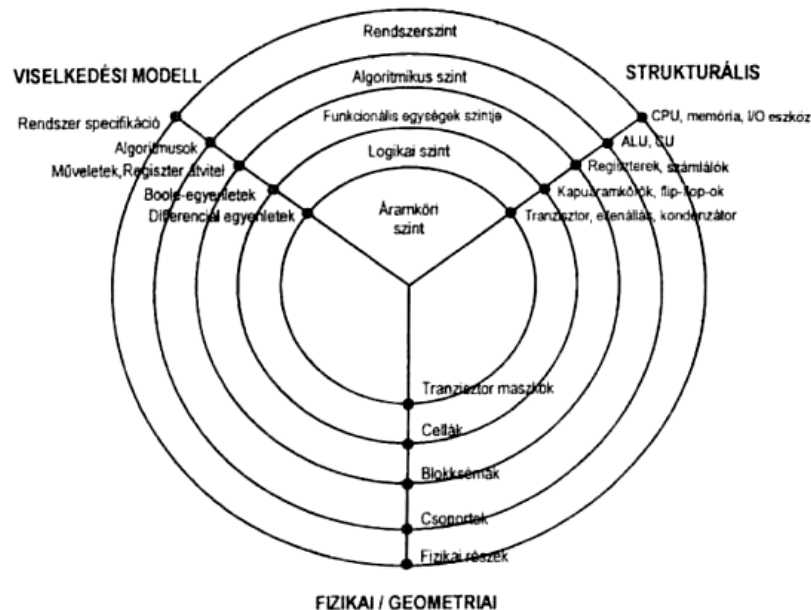
1. ábra. A számítógép általános tömbvázlata

Minthogy egyik egység sem kezelhető elszigetelten, ezért először néhány *rendszertехnikai* kérdést fogunk érinteni.

E jegyzetben megtalálható alapismeretek jórészt általánosak, de a kifejtés során főként a *személyi számítógépek* jellemzőire fogunk koncentrálni.

1. A rétegmodell

A hardver általános és átlátható leírására általában a hierarchikus szintekből felépülő *rétegmodellt* szokták használni. A hardver rétegmodelljében leggyakrabban öt szintet különböztetnek meg: *rendszer szint*, *algoritmikus szint*, *funkcionális blokkok szintje*, *logikai szint* és *az áramkörök szintje*. A rétegmodell ún. *Y-diagrammja* a 2. ábrán látható. A rendszer szinttől az áramköri szintig az építőelemek mérete fokozatosan csökken. Az egyes rétegek az absztrakció szintjében is különbözőek.



2. ábra. A hardver rétegmodellje

Számítógép architektúrának a számítógép funkcionális felépítését, a részegységek kommunikációs kapcsolatait, valamint a rendszer specifikációjának együttesét nevezik. Az architektúra három fő területet fog át: a számítási modellt, az értelmezés szintjét és a leírás irányultságát. Az architektúra fejlesztésének legfőbb oka a teljesítmény növelése. Ezért célszerű áttekinteni azokat a legfontosabb tényezőket, amelyek a számítógépek teljesítményét döntően befolyásolják. Lényegében minden jellemző, amely a teljesítményre kihatással van, az idővel áll kapcsolatban. Ezek az órajel frekvencia, a másodpercenként végrehajtott lebegőpontos műveletek száma és a teljesítményről összetettebb képet adó "Benchmark" tesztek futási ideje.

E mutatók mindig átlagteljesítményt fejeznek ki.

A számítógépek/számítógéprendszerek változatos szempontok szerint oszthatók. Ezek közül a lényegesebbek:

- Teljesítmény szerint
- Az utasítás és adatfolyam száma szerint
- Az utasításkészlet szerint (CISC, RISC)
- A működési elv szerint (Neumann elvű, nem Neumann elvű)
- Az egy időben kiszolgált felhasználók száma és a kiszolgálás időbelisége szerint

A teljesítmény növelésének két alapvető módszere van: a nem strukturális és a strukturális teljesítménynövelés. Az első esetben az órajel frekvenciájának a növelése, vagy a programok optimalizált fordítása említhető. A másodikhoz viszont olyan fogalmak tartoznak, mint szuperskalár processzorok, adatcsatornás feldolgozás megvalósítása, vektorszámítógépek, multiprocesszoros architektúrák használata.

A Neumann elvű, azaz hagyományos számítógépek legfontosabb jellemzői:

- közös tároló, amely egyaránt tartalmazza a program utasításait és az adatokat is
- a tárolt program utasításait sorba véve oldja meg

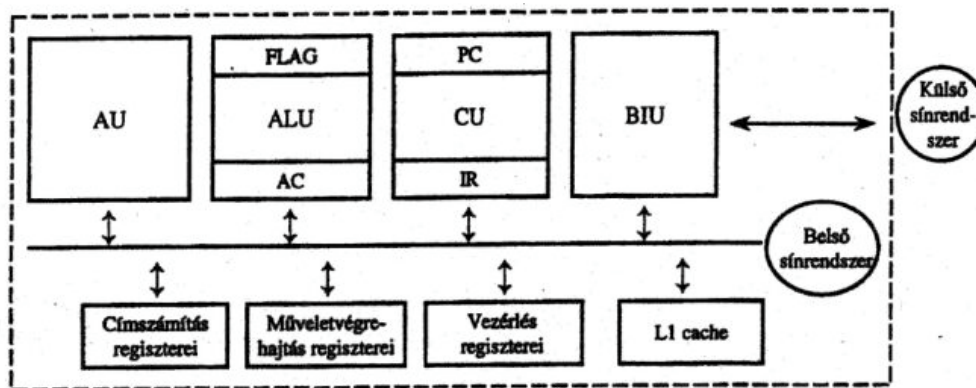
- az aritmetikai és logikai műveleteket egy önálló egység, a műveletvégző egység hajtja végre
- az adatok és a program bevitelére-kivitelére önálló egységek szolgálnak

2. A központi egység

A központi egységek elemzését a következő témák érintésével szokták elvégezni: felépítése, részei és ezek kapcsolata; üzemmódjai és állapotai; regiszterkészlete; utasítás-felépítése, utasításkészlete, utasítás-végrehajtás; memóriakezelése; megszakítások és kivételek kezelése.

A következőkben elfogadjuk azt az állítást, miszerint a *mikroprocesszor* egy olyan nagybonyolultságú áramkör, amely önmaga képes a központi egység feladatainak az ellátására. E szemlélet szerint a központi egység jellemzését és a processzor jellemzését első közelítésben egyenértékűnek fogadhatjuk el.

Egy manapság jellegzetesnek tekinthető központi egység logikai felépítését a 3. ábrán tüntettük fel.



3. ábra. A központi egység logikai felépítése

Az ábrán a következő fő alegységek figyelhetők meg: vezérlőegység, végrehajtó egység, címszámláló egység, sínillesztő egység, regiszterkészlet, belső gyorsító tároló és belső sínrendszer.

A nemrég még hagyományosnak mondott értelmezésben a vezérlőegységet, az aritmetikai logikai egységet és a regisztereket tekintették a központi egység részeinek.

- A vezérlőegység (Control Unit) fő feladata az összes részegység irányítása és összehangolása. A vezérlőegység két fontos regisztere az *utasításslámláló*, amely a következő utasítás címét tartalmazza, illetve az *utasítás regiszter*, amely az épp kiolvasott utasítást tárolja. A műveleti kódot dekódolva határozza meg a vezérlőegység a végrehajtásra kerülő műveletet.
- A végrehajtó egység (Execution Unit) az előírt aritmetikai és logikai műveleteket hajtja végre. Kiemelt jelentőségű az ún. *akkumulátor* regisztere, amely a műveletben résztvevő operandusokat átmenetileg tárolja. Ugyancsak nagyfontosságú az *állapotregisztere*, amelyben az adott művelet végrehajtása során megváltozott állapotok (pl. *átvitel*, *túlcsordulás*) kerülnek egy-egy bit formájában tárolásra.

- A címszámító egység (Address Unit) feladata mindenekelőtt a címmezőben megjelenő címek leképezése a tárolóban fizikailag létező címekre. A korszerű rendszerekben a címszámító egységnek egyéb fontos feladatai is vannak a tárolókezeléssel kapcsolatban (pl. *védelmi hibák* felismerése)
- A sínillesztő egység (Bus Interface Unit) biztosítja a központi egység kapcsolódását a külső rendszersínhez. A rendszersín az *adatsín* (adatvonalak), a *címsín* (címvonalak) és a *vezérlősín* összessége. A sínillesztő rendszerint saját sínvezérlőt, valamint az adatok és címek számára *átmeneti tárolókat (puffer)* tartalmaz.
- A központi egység regiszterei három csoportba sorolhatók. A *rendszerregiszterek* a felhasználói programok számára nem hozzáférhetőek (pl. utasításregiszter). A *speciális célú regiszterek* csak bizonyos utasításokban szerepelhetnek (pl. *állapotregiszter*). Az *általános célú regisztereket* a felhasználói programok korlátozás nélkül használhatják. Ezen kívül a legmodernebb processzorok külön regiszterkészletet használnak, amikor a CISC utasításokat RISC utasításokra bontják.
- A gyorsító tároló (Cache) a rendszer központi tárolója egy részének a tükre. Célszerű ugyanis, hogy a soron következő utasítások és a hozzátartozó adatok a sorra kerülés pillanatában már a processzoron belül legyenek. Ezáltal gyorsul a végrehajtás.
- A belső sínrendszer (Internal Bus) a központi egység részegységei között biztosítja a kétirányú kapcsolatot.

A processzorok különböző üzemmódjaira védelmi szempontok figyelembevétele, az operációs rendszer és a felhasználói programok elkülönített üzemeltetése és az előző processzorokkal való programkompatibilitás biztosítása miatt van szükség. Például a Pentium processzoroknak négy üzemmódjuk van.

Valós üzemmódban úgy működik, mint egy 8086-os processzor.

A *védett üzemmódban* megvalósul a *többfeladatos (multitasking)* funkció és kihasználhatók a processzor összes lehetőségei.

Védett valós üzemmódban a processzor a 8086-ost egy taszkban emulálja.

A *rendszermenedzselő üzemmód* a processzor energiatakarékos működési módja.

Utasításszerkezet alatt a gépi kódú, a processzor számára közvetlenül értelmezhető utasítások felépítését értjük. Ez határozza meg, hogy az utasítás egyes részeit hogyan kell értelmezni. Az utasításszerkezet leglényegesebb részei:

- *műveleti rész* (meghatározza a művelet típusát)
- *operandus hivatkozások* (milyen operandusokkal kell a műveletet végrehajtani és ezek hol találhatók)
- *kiegészítő rész*

Utasításkészlet az elemi szintű, gépi kódú utasítások összessége, amelyek végrehajtására a processzor hardver szinten képes, és amelyre a fordítóprogram a programokat lefordítja. A különböző utasításokat több szempont szerint lehet csoportokba sorolni. Leggyakrabban *adatmozgató, műveletvégző* és *vezérlő utasításokról* beszélünk. Esetleg külön említhetők az ún. *multimédiás* és *3D utasítások* (SIMD utasítások), amelyekkel a *3D grafikákhoz* és a *képfeldolgozáshoz* szükséges vektoros adatokat dolgozzák fel.

A központi egység utasítás-feldolgozása az ALU műveletvégrehajtásától, a feldolgozás műveleti vezérlésétől és az utasításfeldolgozás gyorsításától függ. Az ALU működése és képességei a műveletekben résztvevő adatok alapján értékelhető (fixpontos, lebegőpontos, BCD, MMX adatok, *karakteres mezők*, *bitmezők* stb.).

A vezérlőegység legfontosabb jellemzője maga a vezérlés megoldása, amely *huzalozott*, vagy *mikroprogramozott* lehet.

Az utasításvégrehajtás pl. *adatcsatornás*, *szuperskalár* szervezéssel gyorsítható.

A tárolókezelés egyik legfontosabb eleme a különböző címzési módok megvalósítása. A másik fontos elem a védelmi rendszer, amely a biztonságos működést alapvetően befolyásolja.

A *Harvard architektúrájú számítógépek* felépítése ugyanaz, mint a Neumann elvű gépeké, azzal a különbséggel, hogy külön program- és külön adattárolót használ a központi egység. E két tárolási feladat szétválasztásával csökken a közös tároló és közös sínrendszer használatából eredő szűk keresztmetszet és így növelhető a gép teljesítménye.

A programvégrehajtást időszakosan felfüggesztő okok a *megszakítások* és a *kivételek* (processzoron belül fellépő esemény miatti megszakítás) lehetnek. A *megszakítási rendszert* a megszakítások kiszolgálásának lépései, a *maszkolhatóság* és nem maszkolhatóság, a megszakításkezelés hardver megvalósítása, illetve a *vektoros* és nem vektoros kezelés jellemzi.

A központi egységen belül (akár a processzoron belül is), de a számítógép egységei között is a kommunikáció a sínrendszeren keresztül valósul meg. A sínrendszert három sín alkotja, amelyek mindegyikének a legfontosabb jellemzője a *sín szélessége*.

A hagyományos felépítésű központi egység legnagyobb hátránya, hogy a rendelkezésre álló *erőforrásokat* rosszul használja ki, adott időben csak egy program végrehajtására kerülhet sor és az erőforrások nem megoszthatók. Hátrány továbbá a processzor és a tároló közötti adatátvitel viszonylag alacsony sebessége. A hagyományos gépeknél a sebesség növelésére csak kevés lehetőség kínálkozik az erőforrások bizonyos párhuzamosítása révén. E célt szolgálja a többfeladatos feldolgozás (multitasking), a felhasználói programok és a beviteli-kiviteli műveletek egyidejű elvégzése, a mikroprocesszor egyes egységeinek a többszörözése, valamint az egyes végrehajtási folyamatok *átlapolt feldolgozása*.

Az erőforrások szétosztása tekintetében többféle módszer alkalmazható: a *prioritásos elv* a feladatok fontosságát veszi tekintetbe, az *időosztásos üzemmódban* az erőforrások egyenlő időközönként férnek hozzá a processzorhoz, *valós idejű üzemmódban* egy feladat kiemelt fontossága azt eredményezi, hogy amennyiben igényli a processzor, vagy más erőforrás használatát akkor ezeket azonnal át kell számára engedni.

A rendelkezésre álló erőforrások jobb kihasználását megfelelő szervezéssel az operációs rendszeren keresztül is lehet javítani.

3. Műveleti vezérlés

Műveleti vezérlés alatt a gépi utasítások elemi lépéseinek végrehajtásához a számítógép hardver részeinek a gépi utasítás alapján történő irányítását értjük.

Az utasítások elemi lépései a következők:

- *Utasításlehívás* - az utasításszámláló regiszter alapján az utasítás a főtárból az utasításregiszterbe kerül
- Az utasításszámláló tartalmának növelése
- Az utasítás dekódolása - a műveleti kód és az utasításszerkezet értelmezése, az operandusok címének kiszámítása
- Operandusok kiolvasása a főtárból
- A művelet végrehajtása az előkészített operandusokkal
- Az eredmény beírása az előírt tárolóhelyre

Az elemi lépések áramköri szinten további részekre bonthatók.

A vezérlőegység működése során vezérlőjeleket ad ki az egész rendszer számára. A belső vezérlőjelek a központi egységen/mikroprocesszoron belüli részegységek működését irányítják. A külső vezérlőjelek a processzor és a beviteli-kiviteli eszközök közötti adatátvitelt, a megszakításkezelést, a sínvezérlést stb. irányítják. A vezérlőjelek az ún. vezérlési pontokban fejtik ki hatásukat.

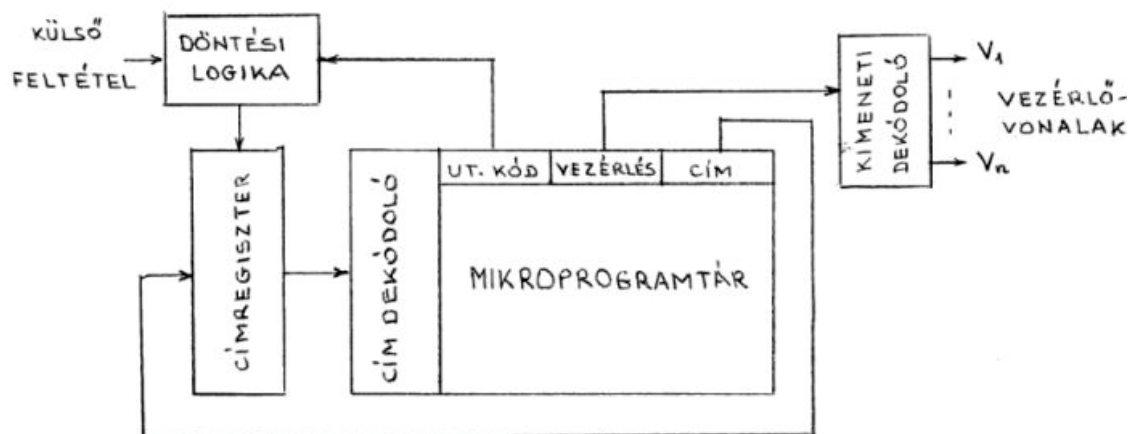
A műveleti vezérlést huzalozott, vagy mikroprogramozott módon lehet megvalósítani.

A huzalozott vezérlés a műveletvégrehajtás közvetlen kapcsolatokon keresztüli irányítását jelenti.

A mikroprogramozott műveleti vezérlés esetében az utasítás elemi lépéseit egy mikroprogram tárolóban (ROM típusú tároló) található mikroprogram vezérli. Ebben az esetben a gépi kódú utasítás műveleti része egy mikroprogram lefutását eredményezi. A mikroutasítások végrehajtása sorrendben történik.

Az említett két műveleti vezérléstípus közül a huzalozott gyorsabb, ennek ellenére elterjedtebb a mikroprogramozott műveleti vezérlés, mindenekelőtt a rugalmassága miatt.

A mikroprogramozott vezérlés horizontális, vagy vertikális lehet. Itt a horizontális mikroprogramozás elvét fogjuk kissé részletesebben megvizsgálni. A mikroprogramtár minden sora egy mikroutasítás. A mikroutasítás a vezérlővonalakat állítja be és kijelöli a következő mikroutasítás címét. Az új cím a címregiszteren és a dekódolón keresztül címezi a következő mikroutasítás sorát. A külső feltételek figyelembe vételét a döntési logika teszi lehetővé. Egy tetszőleges feltétel teljesülése, vagy nem teljesülése esetén, más-más sor fogja a következő mikroutasítás helyét kijelölni. A horizontális mikroprogramvezérlés vázlatát a 4. ábrán tüntettük fel.



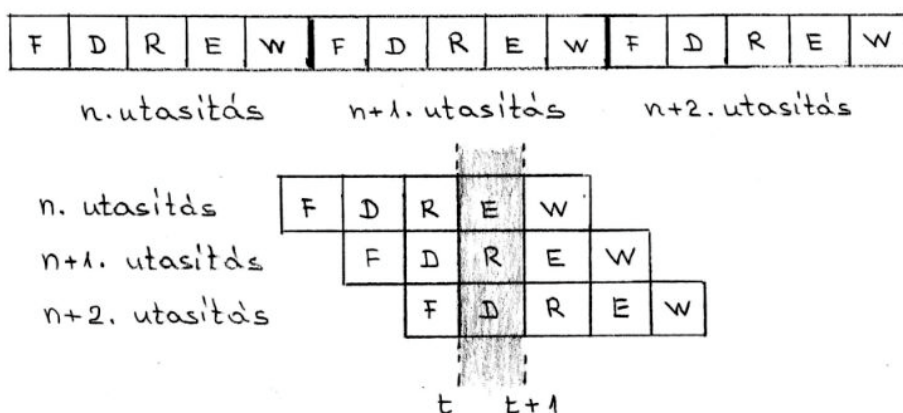
4. ábra. A horizontális mikroprogramvezérlés vázlata

A vezérlési mező minden egyes bitje az áramkörök egy-egy vezérlési pontját állítja be.

A vertikális mikroutasítás felépítése a gépi kódú utasításhoz hasonlít.

4. Az utasításvégrehajtás gyorsítása

Mint már említettük a gyorsítás nemstrukturális, vagy strukturális módszerekkel valósítható meg. Mivel az órajelfrekvencia növelésének a mindenkor technológia korlátokat szab, előtérbe kerültek azok a módszerek, amelyek a gyorsabb műveletvégrehajtást rendszertechnikai eszközökkel oldják meg. Ezek közül az egyik legjelentősebb módszer az utasításvégrehajtás szintjén *átlapolt feldolgozás (pipelining)*. Ennek elvét szemlélteti az 5. ábra. Az összehasonlíthatóság végett ugyanez az ábra tartalmazza a soros végrehajtás elvét is.



5. ábra. A soros és az átlapolt feldolgozás elve

Feltételezzük, hogy a gépi utasítás elemi lépései a következők: utasításkioltás (F), dekódolás (D), operandus kioltás (R), végrehajtás (E), eredményviisszairás (W). Az átlapolásos feldolgozás elve azon a felismerésen alapul, hogy az első utasítás egyes lépése befejezésekor, mielőtt áttérnek a második lépésre, azonnal megkezdődhet a második utasítás első lépésének a végrehajtása.

Három utasítás esetében fogjuk a folyamatot vizsgálni az ábra szerint. Megfigyelhető, hogy a $t - t+1$ időintervallumban mindhárom utasításból egy-egy lépés épp feldolgozás alatt van. Az is észrevehető, hogy a módszer jelentős időmegtakarítást eredményez.

5. Tárolókezelés

A tárolószervezésnek az egyik legfontosabb elve, hogy az adatokat, amelyekre a műveletvégrehajtás során gyakran szükség van gyors elérésű tárolón kell elhelyezni. Mivel a nagyon gyors tárolók igen drágák, ezeket csak kisebb kapacitású egységekben szerelik be. Emiatt a számítógép tároló felépítése *hierarchikus*, éspedig a processzor közelében a gyors elérésű kis kapacitású tároló (cache) helyezkedik el. Távolodva a processzortól a tárolók hozzáférési ideje és a kapacitása nő. A tárolóhierarchia egységei (szintjei) a hozzáférési idő növekvő sorrendjében a következők: regiszter tárolók, gyorsítótár, főtár, háttértár (merevlemez), adatmentő táruk (pl. mágnesszalag).

A tárolókezelés alapvető feladatai:

- biztosítja a processzor műveletvégrehajtásához a szükséges adatokat

- összehangolja és irányítja a tárolóhierarchia egyes szintjein levő tárolók működését

E feladatok végrehajtását a központi egység *tárolókezelője* végzi. A tárolókezelő rendszerint a processzorba beépített egység (pl. a címszámító egység), vagy önálló hardver elem.

A tárolókezelés módja minden korszerű mikroprocesszor/központi egység esetében az ún. *virtuális tárkezelés*. Itt az alapfeladat az, hogy hogyan lehet olyan nagy tárolóigényű programfolyamatot futtatni, amely nem fér be a főtárba. Ennek az ismertetése azonban meghaladja e jegyzet kereteit.

Irodalomjegyzék

- 1] Németh, Horváth: *Számítógép architektúrák*, Akadémiai, Budapest, 1993.
- 2] Budai A.: *Mikroszámítógép rendszerek*, LSI, Budapest, 1999.
- 3] Czerny L.: *Mikroszámítógépek*, LSI, Budapest, 1996.
- 4] Agárdi, Hadi: *A Pentium*, LSI, Budapest, 1998.
- 5] Czerny L.: *RISC processzorok*, LSI, Budapest 1996.

Memória technológiák*

Dr. Farkas György, nyugalmazott főkutató

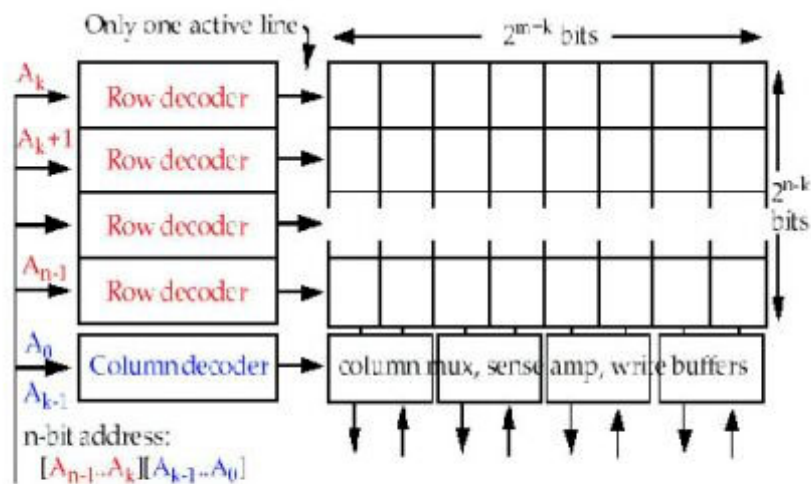
A memória nélkülözhetetlen komponens bármilyen számítógéprendszerben. Feladata a programok és a hozzájuk tartozó adatok tárolása.

A memóriatechnológiák a következőképpen oszthatók:

- Read Write Memory (RWM) - olvasható, írható memória
- Random Access Memory (RAM) - a hozzáférési idő nem függ az adatok memóriában való elhelyezésétől.
- Static RAM (SRAM) - gyors, viszonylag kis kapacitású memória típus.
- Dynamic RAM (DRAM) - lassúbb, nagy kapacitású memória típus.
- Non - RAM: Serial Access Memory - FIFO, LIFO; - változó hozzáférési idő.
- Content Access Memory (CAM) - tartalomfüggő kiválasztó rendszerű memória.
- Non - volatile Read Write Memory (NVRWM): EPROM, EEPROM, FLASH - nem illékony, olvasható, írható memória típusok; a beírási idő jóval nagyobb, mint az olvasása.
- Read Only Memory (ROM) - gyártási maszk által programozott, csak olvasható memória.
- Programmable ROM (PROM) - felhasználó által csak egyszer beírható, többször olvasható memória.

Static vs. dynamic - a statikus memória megőrzi a tartalmát, amíg tápláló feszültséget kap; a dinamikus memória tartalmát időnként fel kell frissíteni.

Synchronous vs. asynchronous - a szinkron memóriák egy órajel élen veszik át vagy adják ki az adatokat; az aszinkron memóriák felismerhetik a címváltozást, és ennek alapján indíthatnak be ciklusokat.



For example: Let $N = 1,048,576$ and $M = 8$ bits for a 1 million byte memory.
 $n = \log_2 N = 20$, $k = 8$ and $m = \log_2 M = 3$.
 Then there are 2^{n-k} rows = $2^{12} = 4096$ and
 2^{k+m} columns / 2^3 bits per word = $2^8 = 256$ words.

* **Forrás:** James F. Plusquellic http://www.cs.umbc.edu/~plusquel/vlsi/slides/chap8_2.html

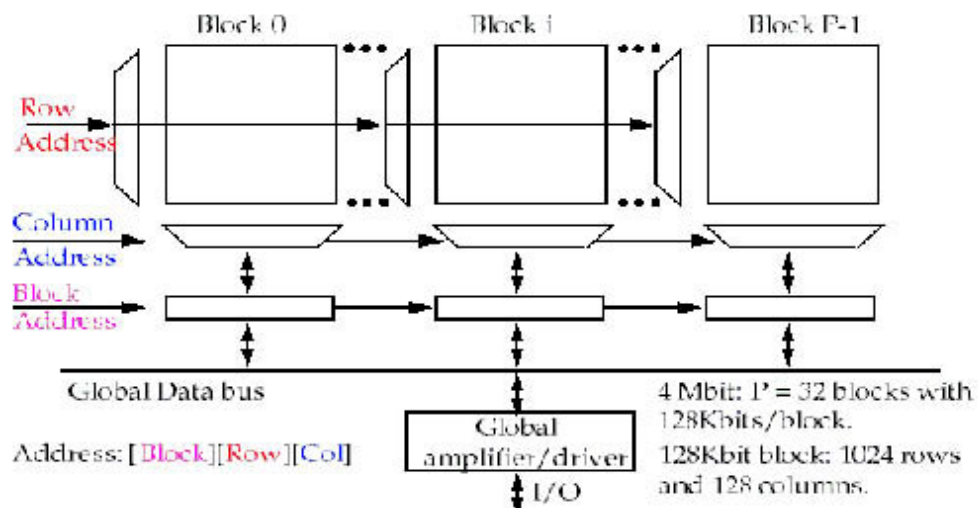
Memória architektúra

Egy tipikus memória áramkör belső felépítése a következő:

- Row decoder - Sor kiválasztó áramkör.
- Column decoder - Oszlop kiválasztó áramkör.
- Sense amplifier - Olvasó erősítő.
- Write buffer - Beíró erősítő.
- Column multiplexer - A kiválasztott oszlopot a ki/bemenethez kapcsoló áramkör.

A fentebb ábrázolt egy modulos modell kis kapacitású memóriák esetében (256kbit) használható.

Nagy kapacitású memóriák több modulos (P block) architektúrát használnak.



Az ábra szerint egy 4Mbit-es memória címszerkezete a következő:

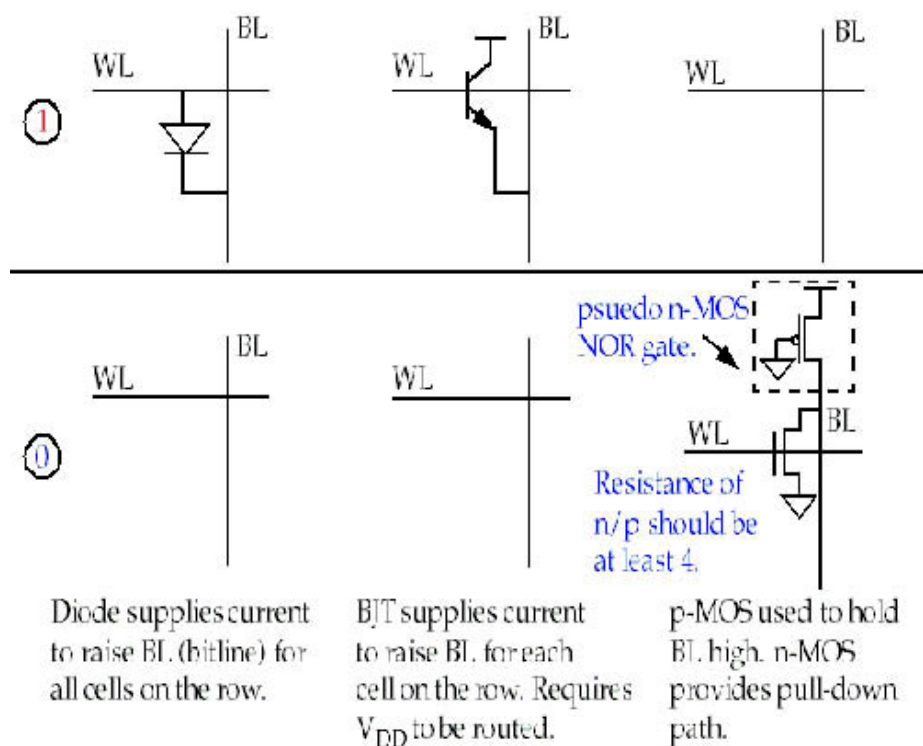
Block-5bit, Sor-10bit, Oszlop-7bit, összesen 22bit.

ROM memória

A ROM cellák tartalma változhatatlan. Több fizikai megoldás használható.

A következő ábrán dióda, réteg és MOS tranzisztor alapú cellák láthatók, logikai 1-es és 0 állapotban.

A dióda és a réteg tranzisztor pozitív jelet továbbít a kimenő bitvonalra (BL), ha a megfelelő kiválasztó szóvonal (WL) aktív. A diódás megoldás a WL feszültségét adja át a kimenetre, a tranzisztor pedig a kollektor tápfeszültségét, a megfelelő feszültségesések figyelembevételével és/vagy kimenő erősítőkön keresztül.



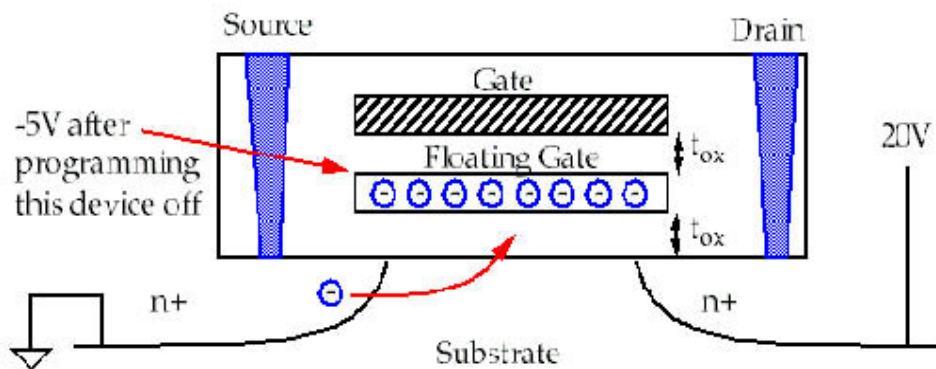
A MOS tranzisztoros változat esetében minden BL-t egy-egy pMOS tart 1-es szinten. Ha 0-t kell generálni, egy, a megfelelő bit helyzetre beépített nMOS lehúzza a pMOS által tartott BL-t, alacsony feszültségre. Hogy a pMOS, nMOS feszültségesztő jól működjön, az nMOS ellenállása legalább négyszer kisebb kell legyen mint a pMOS-é.

Nem illó olvasható, írható memóriák

A memória felépítése hasonlít a ROM memória szerkezetéhez. A tranzisztorok szelektív be/kikapcsolása a küszöbfeszültség módosításával történik. A küszöbfeszültséget egy lebegő elektróda feltöltésével/kisütésével változtatják.

Beíró (magas, 15-20V) feszültség alkalmazása a source (forrás) és drain (csapolás) között erős elektromos mezőt hoz létre és lavina -(forró elektron)- belövést okoz a lebegő elektródába, a normális feszültséggel táplált kiválasztó elektróda alatt.

A forró (nagy sebességű) elektronok áthatolnak az első oxidrétegen, és negatív töltést tárolnak a lebegő elektródán, megnövelve a küszöbfeszültséget kb. 7V-ra.



A kitörlési módszer határozza meg a különböző újraprogramozható nem illékony memóriák típusát.

EPROM: Erasable Programmable ROM - Törölhető, programozható ROM

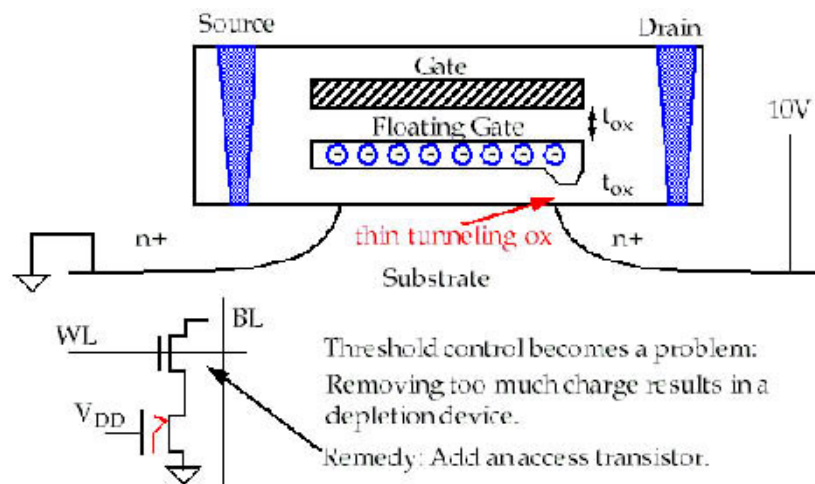
- ultraibolya (UV) fény hatására a lebegő elektródán tárolt elektronok egy része eltávozik, csökken a küszöbfeszültség, ez a törölt állapot. Az összes memória cellát egyszerre törlik, egy átlátszó ablakon keresztül.
- A törlés lassú, percek alatt megy végbe.
- A programozás is lassú, 5-10 mikroszekundum szavanként.
- A programozási ciklusok száma véges, kb. 100-1000.
- Jó helykihasználás, egy tranzistor cellánként.

EEPROM- Electrical Erasable PROM - Elektromosan törölhető PROM

Nagyon vékony oxid rétegen keresztül, a Fowler-Nordheim alagúthatás alapján töltődik fel beírásakor és sül ki törléskor a lebegő elektród.

Törléskor a beíráshoz képest fordított feszültséget alkalmaznak.

Érzékeny a küszöbfeszültség értékére, különálló kiválasztó tranzisztort kapcsolnak minden tároló tranzistorhoz.

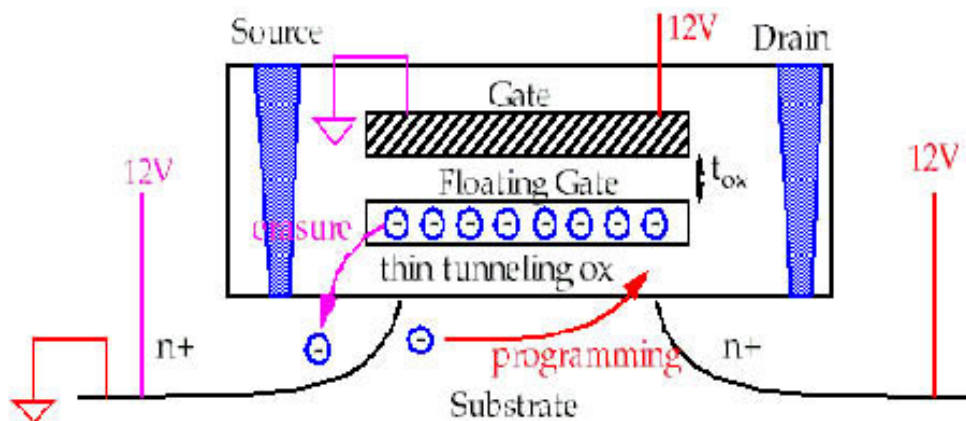


Flash EEPROM

A Flash technológia az EPROM és EEPROM kombinációja. Programozáskor forró-elektron-injektálást, törléskor Fowler-Nordheim alagúthatást használnak.

Törléskor a hardver ellenőrzi a küszöbfeszültség értékét, biztosítva, hogy a kitörölt tranzisztor működőképes maradjon.

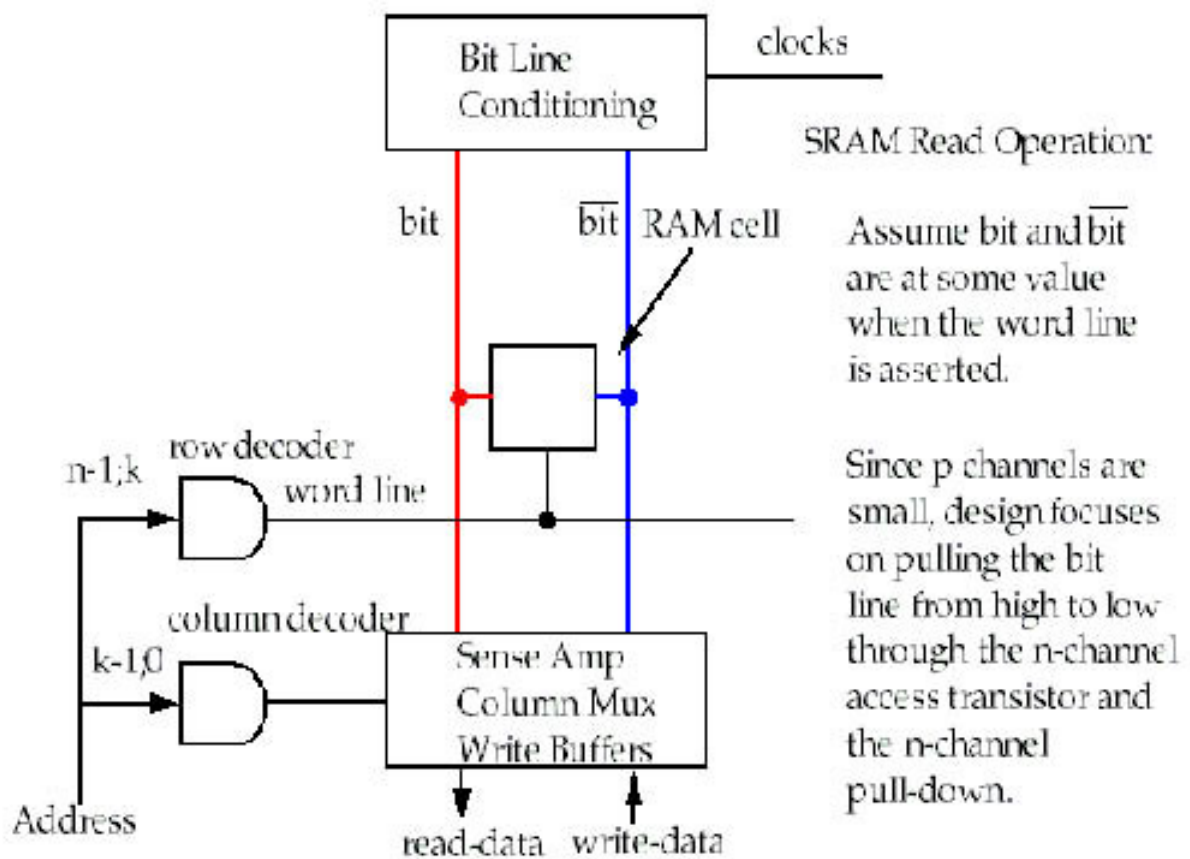
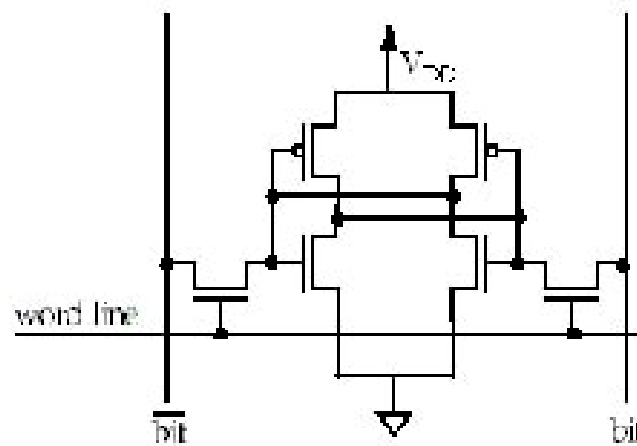
Programozáskor a source (forrás) földpotenciál, a gate (kapu) és a drain (csapolás) 12V-feszültségen van.



Törléskor a kapu földpotenciál, a forrás 12V-on van, a csapolás táplálása meg van szakítva.

SRAM Sztatikus RAM

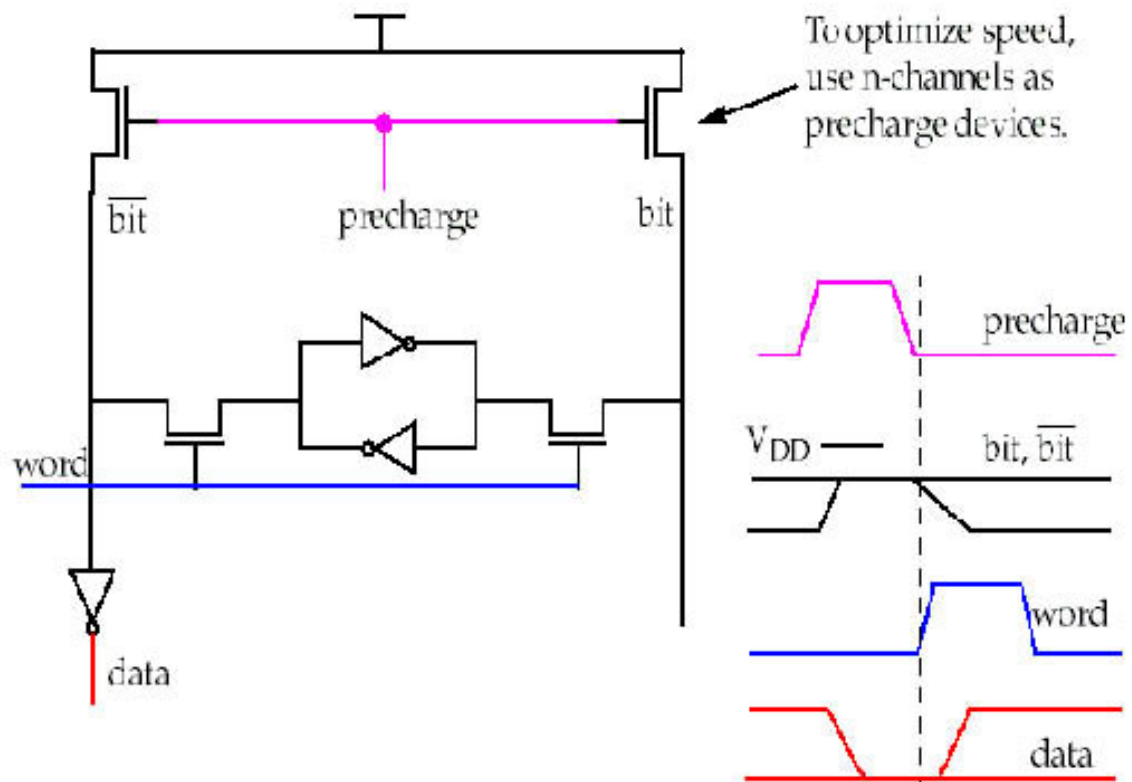
A sztatikus RAM hat tranzisztort tartalmaz. A cella alapja egy 4 tranzisztorból álló billenő áramkör. A szókiválasztó vonal által vezérelt 2 tranzisztor kapcsolja a cellát a bitvonalakhoz. Olvasáskor a cella tartalma határozza meg a bitvonalak potenciálját. Beíráskor a bitvonalak határozzák meg a cella állapotát.



Olvasó műveletnél a bit és /bit vonalakat előtöltik 5V-ra, mielőtt aktiválnák a szókválasztó vonalat.

Előtöltés után a címzett cella vezető nMOS tranzisztorja lehúzza "0" szintre a megfelelő bit vagy /bit vonalat, meghatározva ezáltal a kiolvasott jel értékét. Az előtöltést azért alkalmazzák, hogy az olvasás pillanatában a kiválasztott cella képes legyen beállítani a bitvonalak logikai szintjét, és külső tápfeszültségek ne befolyásolják a cella tartalmát a két kiválasztó tranzisztoron keresztül.

Az előtöltést meg kell szakítani a kiválasztó vonalak aktiválása előtt. Ellenkező esetben a kiválasztott cellák parazitán billenhetnek az "1"-es szintre beállított bitvonalak hatására.



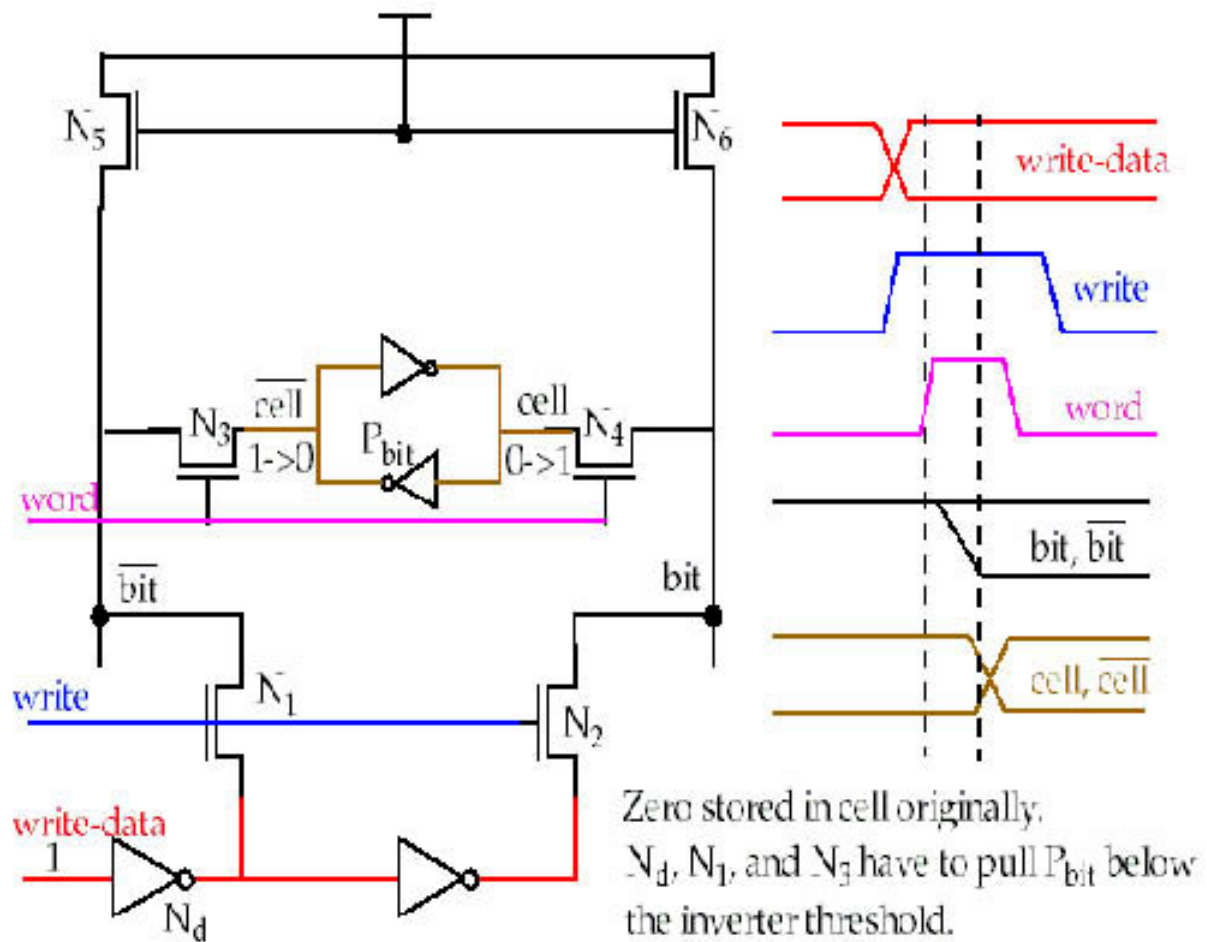
SRAM: Read Operation - Olvasási művelet

A beírás pillanatában a bitvonalakat megfelelően méretezett tranzisztorok révén hajtják meg, amelyek a beírandó adatbit szerint vezérlik a cella tranzisztorait a kiválasztó tranzisztorokon keresztül.

A beírási ábrán az N5, N6-os tranzisztorok a tápforráshoz kötött ellenállások szerepét töltik be.

N1, N2 a beíró, N3, N4 a kiválasztó tranzisztorok.

Az ábra egy "0" tartalmú cella "1"-be való átírását illusztrálja.



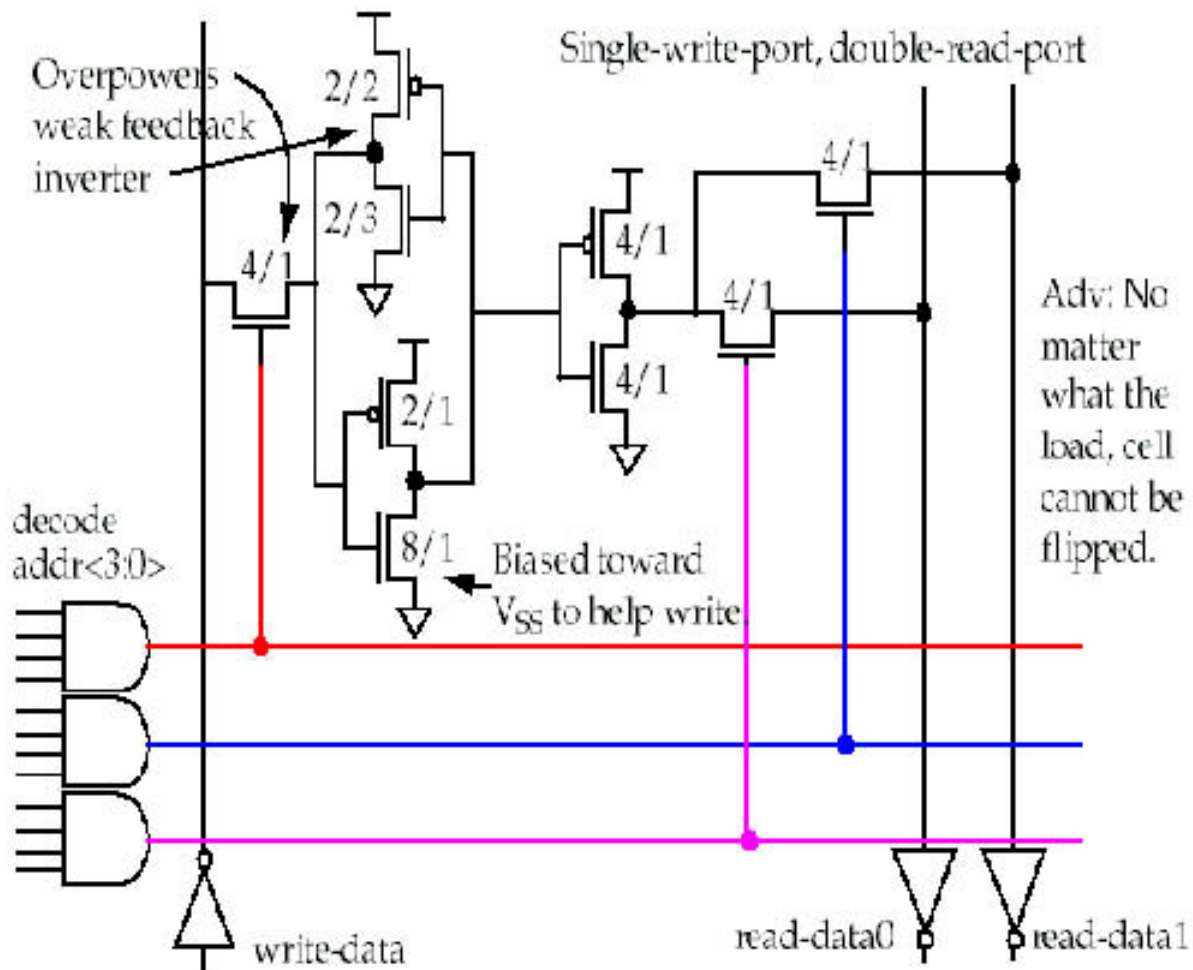
SRAM: Write Operation - Beírási művelet.

Register files - Regiszter blokk

A következő ábra egy-író, két-olvasó hozzáférésű regiszterblokkot mutat be.

A tranzisztorok melletti számok a tranzisztorok geometriáját és ezáltal a belső ellenállásukat jellemzik. Minél nagyobb a tört szám értéke, annál kisebb a tranzisztor ellenállása, nagyobb teljesítményt tud vezérelni, és meghatározza az áramkör magatartását.

A regiszter alapcellája egy 4 tranzisztoros billenő áramkör. A cella egy fordító áramkörön keresztül kapcsolódik az olvasó vonalakhoz. Teljesítménynövelés mellett a fordító áramkör a cella szigetelését is biztosítja, az olvasó bitvonalakon keresztül nem lehet parazitán beírni a cellába.



Register file: két olvasás, egy írás hozzáféréssel.

DRAM : Dynamic Random Access Memory - Dinamikus RAM

A DRAM cella általában egy félvezető kondenzátoron tárolja az információt.

Mivelhogy, akármilyen jó is a szigetelés, egy bizonyos idő múlva a kondenzátor kisül, a cellát periodikusan újra kell tölteni (írni), ezért nevezik dinamikusnak a cella működését. Az újratöltés ugyanazzal az információval történik, amely be volt írva a cellába, ezért ezt a műveletet frissítésnek ("refresh") nevezik. A frissítés tehát egy olvasás, amelyet a kiolvasott érték automatikus visszairása követ. Mivel minden cellát fel kell frissíteni msec nagyságrendű periódussal, ez bizonyos idővesztést okoz a memória működésében.

3T DRAM

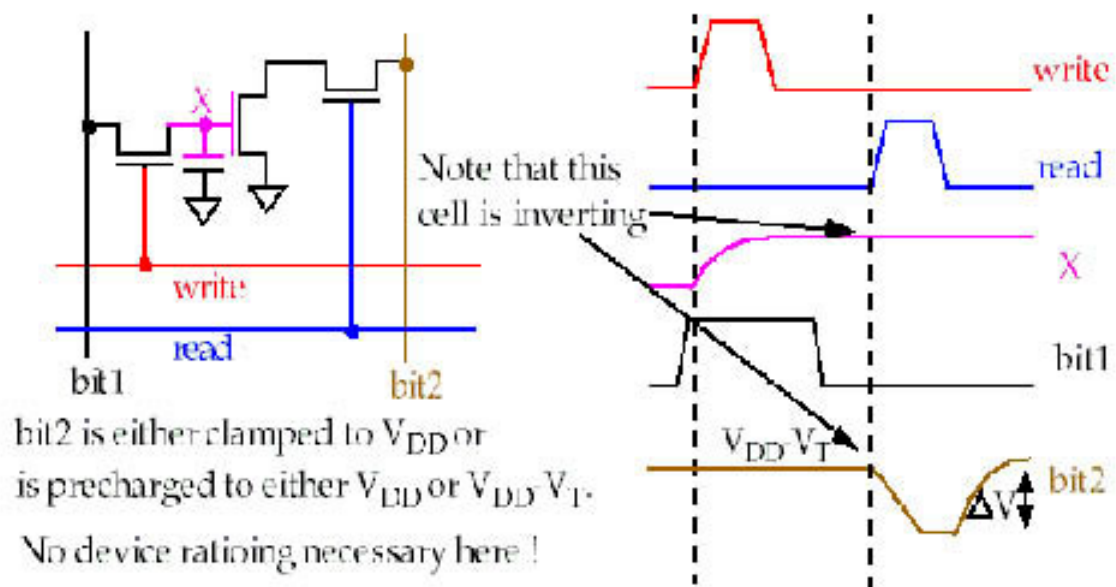
Háromtranzisztoros dinamikus memória cella.

A frissítést meg lehet valósítani a bit2 vonal olvasásával, és a fordított érték visszairásával a bit1 vonalon keresztül.

A bit2-es vonalat előtöltik olvasáskor, hogy ne legyen állandó nyugalmi áram.

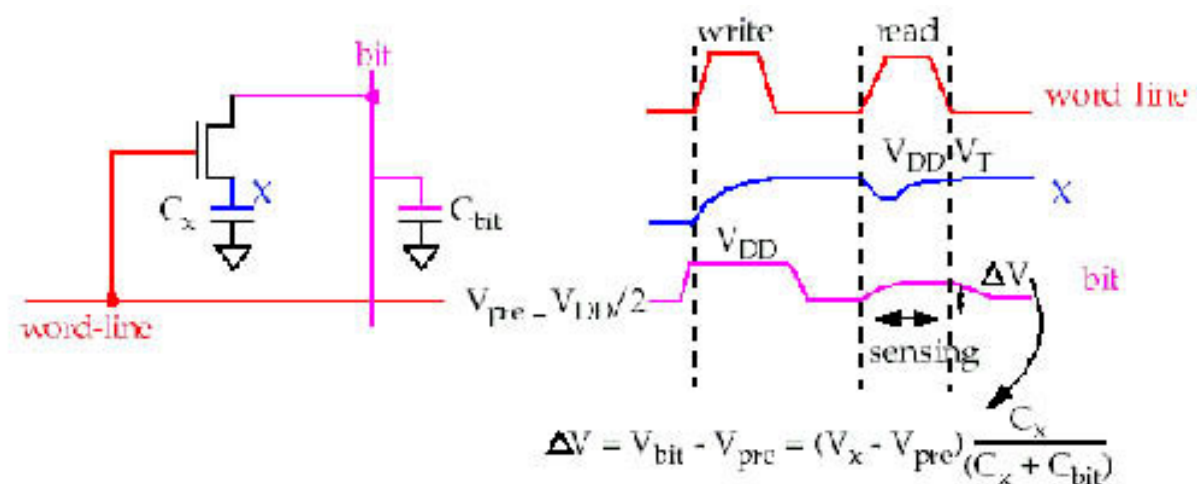
Ezt a memória felépítést néha ASIC áramkörökben használják, mivel egyszerű a tervezése és a működése is.

A tároló kondenzátor szerepét az X-el jelölt vezérlő kapu elektród parazita kapacitása tölti be. A baloldali beíró tranzisztor, amikor ki van választva, feltölti vagy kisüti az X kondenzátort, a bit1 vonal által meghatározott értékre. Olvasás előtt feltöltik a bit2 vonalat a tápfeszültség értékére. A kiválasztott cella határozza meg olvasáskor a bit2 vonal potenciálját. Ha az X kondenzátor fel volt töltve, az olvasó tranzisztor kisüti a bit2 vonalat a kiválasztó tranzisztoron keresztül. Ha nem volt feltöltve, a zárt olvasó tranzisztor nem süti ki a bit2 vonalat. A cella inversor (fordító), ha magas szintű feszültséggel írtak be a bit1 vonalon, olvasáskor alacsony feszültség jelenik meg a bit2-ön, és fordítva.



1T1R DRAM

Egy tranzisztros dinamikus memória cella.

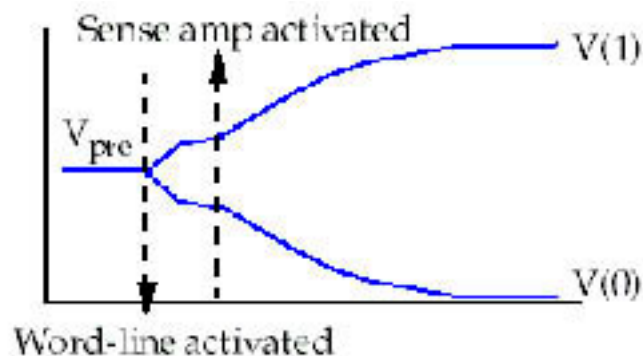


Ez a leggyakrabban használt memória cella. C_x egy különleges felépítésű kondenzátor, a cella tároló eleme. C_x értéke kb. 30 fF. Beíráskor a szókiválasztó vonal kinyitja a kiválasztó tranzisztort és a bit vonal által meghatározott értékre töltődik fel vagy sül ki a cella tároló kondenzátora.

Olvasáskor töltés átrendezés történik a C_x és a C_{bit} között. C_{bit} a beíró/kiolvasó vonal parazita kapacitása. C_x tipikusan 1 vagy 2 nagyságrenddel kisebb mint C_{bit} , így a potenciálkülönbség (ΔV) értéke kb. 250 mV.

Hogy az 1T cella működőképes legyen, az olvasáskor megjelenő ΔV -t fel kell erősíteni egy olvasó erősítővel (sense amplifier). Az egy bit vonalra kapcsolt cellák egy közös olvasó erősítőt használnak. A kiolvasó művelet a töltésátrendezéssel befolyásolja a cella tartalmát. Ezért minden olvasást ugyanabban a memóriaciklusban visszaírás követ. A visszaírás az olvasó erősítőben tárolt kiolvasott értékkel történik.

Az olvasó erősítő jelét az alábbi ábra illusztrálja.



Content Access Memory (CAM) - Tartalom-hozzáférésű memória

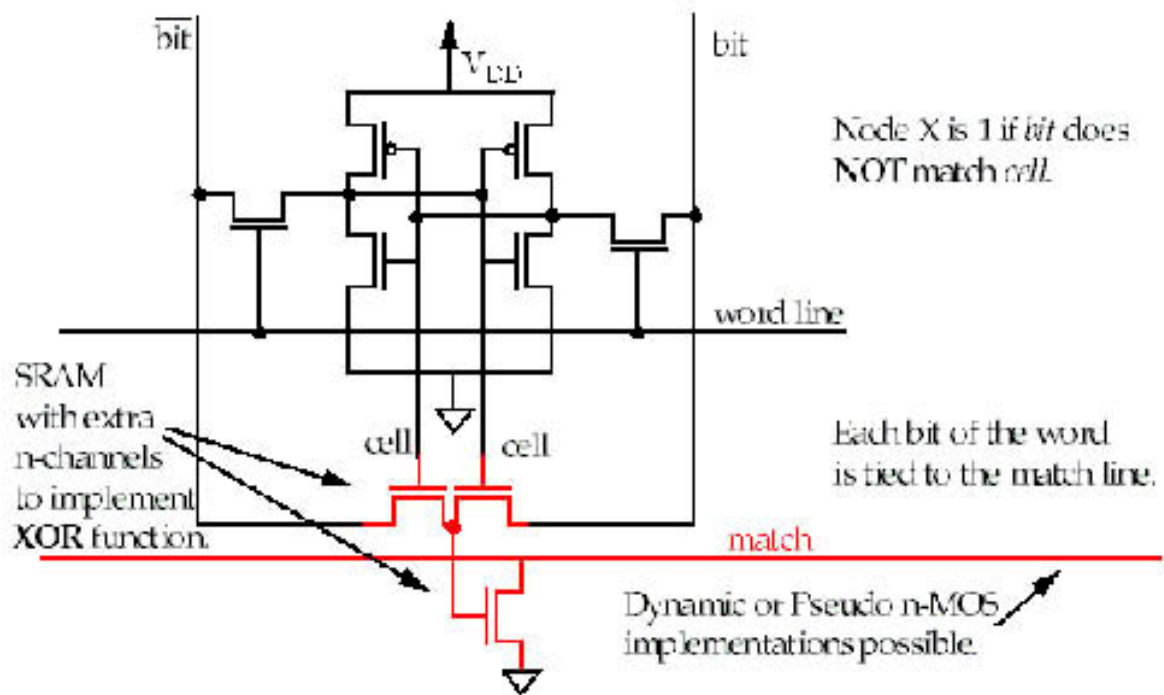
A CAM egy 6 tranzisztoros SRAM memória cellára épül.

Egyszerre összehasonlít egy bemenő adat-szót az összes tárolt adat-szóval.

Például címátalakító memóriák építésénél használják.

Beíráskor a szóvonallal kiválasztott cellákba a bit vonalakon keresztül tárolják a megfelelő adatokat.

Olvasáskor a szókiválasztó vonalakat nem használják. A bit vonalakra ráviszik a keresett adatot és a cell, /cell tranzisztorok összehasonlítják a cella tartalmát a bit vonalakkal. Egyenlőség esetén a match (egyenlő) vonal magas logikai szintre áll be.



Tartalom hozzáférésű memória.

A számítástechnika ipari alkalmazásai

Sebestyén-Pál György, adjunktus

Kolozsvári Műszaki Egyetem, Számítástechnika és Automatizálás Kar

A gyártási folyamatok számítógépes irányítása egy célt és ugyanakkor egy kihívást jelentett és jelent ma is a számítástechnika tudomány számára. Egy valós fizikai folyamat szabályozása és vezérlése különleges tervezési és megvalósítási módszereket igényel, melyek biztosítják a helyes és megbízható működést bármilyen belátható vagy kevésbé ismert környezetváltozások között. Egy helytelen számítás, vagy egy hardver-meghibásodás jelentős anyagi károkat és bizonyos esetekben emberi sérüléseket okozhat.

Általában lényeges szerkezeti különbségek vannak a vezérlési alkalmazásokban és az általános feladatokban (pl. irodai, könyvelési, vagy nyilvántartási alkalmazásokban) használt számítógéprendszerek között. A programozás szempontjából más végrehajtási elvek szükségesek, melyek biztosítják a gyors és időben határolt választ, támogatják a konkurens és párhuzamos feldolgozást, és nagyobb biztonságot nyújtanak. Ezek közül az interaktív és a valós-idejű programozási módszerek említhetők. Ezekre a különbségekre és más számítógépes vezérlésre jellemző feladatokra szeretne fényt deríteni a jelen előadás.

1. Alapfogalmak

Bármilyen előadás kezdetén szükséges meghatározni az alapfogalmakat, melyek az adott téma bemutatásához szükségesek. Egy vezérlési alkalmazásban a következő fogalmak fordulnak gyakran elő: - több művelet összeállítása amely hat és változtat valamit a paramétereken (pl. kémiai folyamatok, fizikai átalakulások, különböző fizikai paraméterek vezérlése, anyagszállítás)

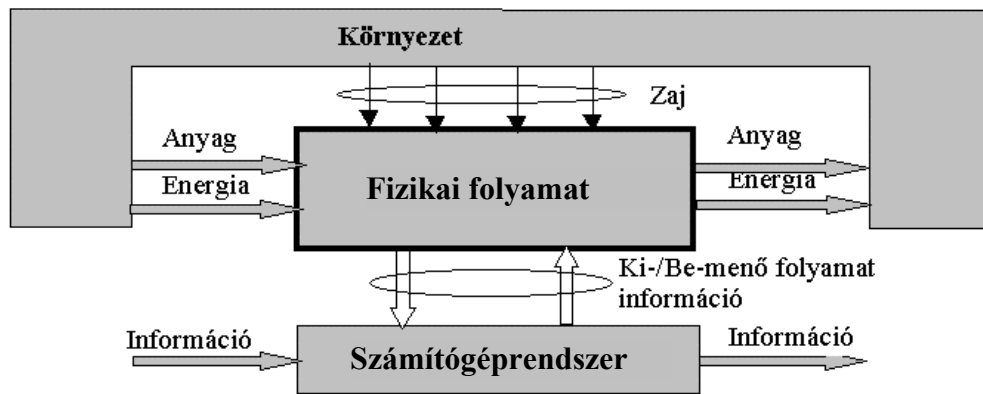
Folyamatvezérlő rendszer - egy rendszer, amely szabályozza és vezérli a folyamat bizonyos paramétereit;

Ki- /Be-menő elemek - anyag, energia és információ

A vezérlés célja - elérni és fenntartani egy előírt paraméter-értéket, egy paraméter időbeli változását okozni, vagy egy fizikai/kémiai átalakulást okozni

Számítógépes vezérlés - a számítógép alkalmazása a folyamat információ-feldolgozásában és a vezérlő jelek/paraméterek létrehozásában

Az 1. ábrán látható egy számítógéppel vezérelt folyamat elvi rajza.



1. ábra. Számítógéppel vezérelt folyamat

Beágyazott rendszer - egy számítógéprendszer, amely része egy komplex elektro-mechanikai rendszernek és felépítése egy meghatározott (szűk) célnak felel meg (célrendszerek)

Valós-idejű rendszer - amelyben egyformán fontos a feldolgozás helyessége és az előre kitűzött határidő betartása

Reaktív rendszer - egy rendszer, amely a külső és belső eseményekre/megszakításokra reagál (nem-szekvenciális rendszer)

2. Egy kis történelem

Az első vezérlő rendszerek **mechanikai** elemeket használtak. Az esetek nagy részében egyedi megoldásokat alkalmaztak és a vezérlő elemeket nem különíthették el a folyamat többi részétől. Nem lehet egy általános és egységes tervezési módszerről beszélni. Különös gondot okozott a folyamat információ-továbbítása.

A következő lépést a **hidraulikus és pneumatikus** rendszerek bevezetése jelképezte. Az e fajta megoldásokban már fel lehet ismerni a klasszikus vezérlőrendszer alapelemeit: az érzékelőt, az átalakítót, az erősítő elemet meg a végrehajtó elemet. Az információ közvetítése is könnyebben megoldható.

Később, az **elektronikus (analóg)** rendszerek biztosították a szükséges gyakorlati háttérrel az automatika és rendszerelmélet megvalósításához. Elektronikus komponensekkel bonyolultabb szabályozó függvényeket lehetett megvalósítani. A szabályozó rendszer kisebb, olcsóbb és biztonságosabb lett. Ahhoz, hogy a különböző típusú vezérlőkészülékeket könnyen lehessen összekapcsolni, **egységesített áram és feszültség jeleket** vezettek be. Ennek köszönhetően biztosított a kompatibilitás bármely automatizálási készülékek között, függetlenül a típustól vagy a gyártótól.

A **programozható automaták** voltak az első digitális vezérlőelemek. Ezek a készülékek egy logikai függvény alapján működnek, amely a legtöbb esetben az időtől is függ. A készülék szekvenciális működést biztosít a vezérelt rendszernek. A rendszer következő lépése egyaránt függ a bemenő jelektől és a rendszer állapotától.

Egy fontos lépést jelentett a **közvetlen (direkt) digitális vezérlés** (DDC - Direct Digital Control) alkalmazása. Ekkor a számítógép már elég gyors, biztonságos és olcsó volt ahhoz, hogy egy termelőfolyamat vezérlését rá lehessen bízni. Egy ilyen alkalmazásban a számítógép része egy visszacsatolt huroknak, és főleg szabályozó szerepet tölt be. Ekkor jelentek meg az

első számítógép által vezérelt **robotok** és manipulátorok. Egy pár év elteltével a robotok nélkülözhetetlen komponenseknek bizonyultak egy modern szerelővonal felépítésében.

Az egyre komplexebb folyamatok vezérléséhez **több processzoros hierarchikus vezérlőrendszereket** alkalmaztak. Egy ilyen rendszerben a vezérlési, felügyelési, és szabályozási funkciókat több logikai szintre osztják. Ezáltal a rendszer átláthatóbb, és gyorsabban válaszol a vezérelt folyamat változásaira.

Az **ipari kommunikációs hálózatok**, és az **intelligens automatizálási készülékek** megjelenése lehetővé tette az **elosztott vezérlőrendszerek** fejlesztését. Az elosztott rendszerek párhuzamos feldolgozást, rövidebb válaszidőt, és jobb hibatoleranciát biztosítanak. A következő táblázat egy pár fontos dátumot tartalmaz a számítógépes vezérlés fejlődésére vonatkozóan.

Év	Fejlődési határkövek	Észrevételek
1959-1962	Az első számítógépes vezérlések: - Egyesült Államok - a Texaco cég petrokémiai gyára - Anglia - Imperial Chemical Industry - az első DDC	- nagy költségek (> 1 millió USD) - alacsony megbízhatóság - lassú feldolgozás
1965-1970	Az Apollo űrrepülési program	- számítógépre bízott több millió \$ értékű felszerelés és emberi élet
1970-1985	A mikroprocesszor korszak	- olcsó, gyors és biztonságos digitális rendszerek
1990-2000	Szoftver fejlesztések, ipari hálózatok, osztott rendszerek	- komplex folyamatok vezérlése

3. Jellemző feladatok a számítógépes vezérlésben

A legtöbb számítógépes vezérlés **digitális jelfeldolgozáson** alapszik. A jelfeldolgozás során több fajta műveletet szükséges alkalmazni:

- mintavételezés (sample&hold) - amely periodikusan rögzíti a bemenő jel pillanatnyi értékét;
- digitális átalakítás - amely az analóg jelből digitális információt állít elő, és fordítva;
- digitális szűrés - amely hasznos frekvenciákat von ki a bemenő jelből, és ezáltal növeli a jel minőségét;
- nemlineáris feldolgozás - amely kontextustól függően jelátalakítást okoz;

Fourier, Laplace, Z átalakítások - amelyek hasznosak a feldolgozás egyszerűsítésében, vagy a bemenő jel elemzésében.

Különös figyelemmel kell kezelni a digitális mintavétel és feldolgozás során megjelenő hibákat. Számolni kell azzal, hogy a jelfeldolgozás időben nem folytonos, a jelek értékei egy diszkrét intervallumhoz tartoznak, és az esetleges túlsordulások hibás eredményt okoznak. Szükséges elemezni a lépcsőzetesen változó kimenő jel hatását, amely bizonyos esetekben rezgést okozhat. Helyesen kell kiválasztani a jelek ábrázolásához szükséges bitszámot ahhoz, hogy egy megszabott pontosságot lehessen elérni. A bitszámok növelését általában a bemenő analóg/digitális átalakító korlátozza.

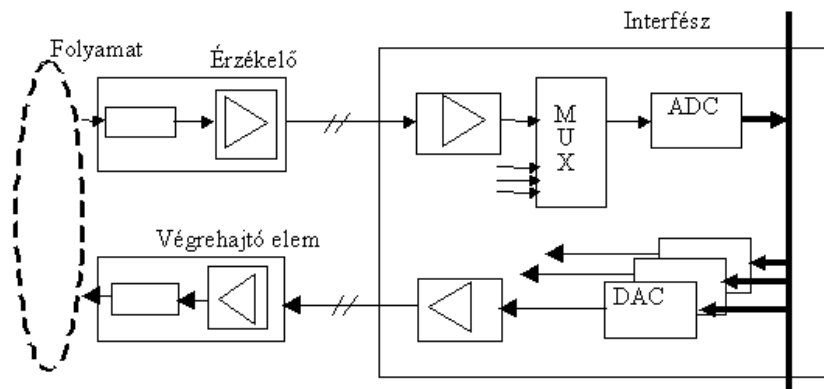
Az idő fontos szerepet játszik bármilyen vezérlő alkalmazásban. Egyrészt azért mert a ki- /bemenő jelek és a szabályozó függvények egyik fontos paramétere az idő. A számítások pontosságához szükséges a pontos időben elvégzett adatgyűjtés és feldolgozás. Másrészt, sok esetben a vezérlőrendszer határolt időn belül kell válaszoljon a külső eseményekre. Ezért szükséges a **valós-idejű programozás** alkalmazása, amely biztosítani tudja az időhatárok betartását. A különböző konkurens taszkok indítását és futtatását az időparaméterek és időhatárok függvényében kell eldönteni. Egy sajátos valós-idejű tervezési algoritmust szükséges alkalmazni, amely része lehet az operációs rendszernek, vagy be lehet építeni a vezérlési programba. Az időbeli tervezést a program futtatása előtt (**off-line**) vagy közben (**on-line**) lehet elvégezni. Az első esetben a határidők betartását garantálni lehet, viszont a rendszer csak az előre feltételezett körülmények között tud működni. Az on-line tervezés nagyobb rugalmasságot biztosít, a rendszer jobban tud alkalmazkodni a környezet változásaihoz, de nem nyújt biztosítékot az időhatárok betartására. A program szerkezete szempontjából két elvi megoldás lehetséges:

- az **idő-vezérelt modell** (time-triggered) - amelyben minden taszk indítása szigorúan az időhöz van kötve, és
- az **esemény-vezérelt modell** (event-triggered) - amelyben a külső események megjelenése határozza meg a taszkok indítását; több párhuzamos kérés esetében az indítás az időhatároktól függ

A számítógépes vezérlésben egy másik jellemző feladatcsoportot a folyamat **információ begyűjtése** és a **vezérlő jelek létrehozása** alkotja. Sajátos kimeneti és bemeneti készülékek és interfészek szükségesek a számítógép és a folyamat közti kapcsolathoz. Egyre fontosabb szerepet játszanak a kommunikációs eszközök, amelyek hibamentesen közvetítik a folyamat-adatokat.

A 2. ábrán egy analóg ki/bemenő csatorna elvi vázlata látható. A bemenő csatorna a következő elemeket tartalmazza:

- az érzékelő (szenzor) - amely átalakít egy folyamatváltozót (pl. hőmérséklet, nyomás, szint, stb.) egy villamos mennyiség változásra (pl. feszültség, áram, ellenállás, kapacitás, stb.); a kimenő jel egy szabványos tartományban változik (pld. 0-10V, 4-20 mA, stb.)
- a kommunikációs eszköz - amely továbbítja a mért jelet
- az erősítő - amely az interfészbe bemenő jelet illeszti az átalakító tartományához; az erősítő szűrési funkciókat is betölt; a szűrés célja - a zavaró frekvencia-komponensek eltávolítása (pl. magas frekvenciás zaj); bizonyos esetekben szükséges galvanikusan elszigetelni a bemenő jelet a számítógép-rendszertől; ezáltal a rendszer védve van a véletlenszerű magas feszültségektől, amelyek egy hibás kapcsolat miatt jelenhetnek veszélyt.
- a multiplexer (MUX) - amely lehetővé teszi több analóg jel beolvasását egy analóg/digitális átalakítóval
- a mintavételező (sample&hold) - amely periodikusan mintát készít a bemenő jel pillanatnyi értékéből
- az analóg/digitális átalakító (ADC)- amely digitális információt állít elő az analóg jel értékéből



2. ábra. Analóg ki/bemenő csatornák elvi vázlata

Egy kimenő csatorna a következő elemeket tartalmazza:

- a digitális/analóg átalakító - amely átalakít egy digitális értéket egy analóg jelre; a kimenő jel amplitúdóban (AM - Amplitude Modulation), frekvenciában (FM - Frequency Modulation) vagy impulzus szélességben (PWM - Puls Width Modulation) modulálható;
- az erősítő - amely előkészíti a kimenő jelet a távoli továbbításra; gyakran a kimenő jelet el kell szigetelni galvanikusan, biztonsági okokból.
- a végrehajtó elem - amely a vezérlő jel alapján változtatja a folyamat egyik paraméterét; gyakran tartalmaz egy mozgó elemet, amelynek helyzete határozza meg a vezérlés nagyságát (pl. szelepek, csapok, motorok, relék, melegítő elemek, stb.)

Újabban az adatgyűjtés és vezérlés **digitális hálózatokon** keresztül történik. A hálózatok használata több előnyt biztosít:

- nagyobb zajimmunitás
- nagyobb távolságokon lehet hibamentesen közvetíteni a jeleket
- alacsonyabb huzalozási költségek (egy szakaszra több automatizálási készülék csatlakozhat)
- strukturált adatokat lehet közvetíteni (nem csak egyszerű analóg értékeket)
- a rendszer lépésenként fejleszthető (egy új elem bekapcsolása nem követel változtatásokat a már létező elemekben)

Viszont egy ilyen megközelítés intelligens készülékeket és megfelelő kommunikációs protokollt igényel. Sajnos az általánosan használt számítógép-hálózatok nem felelnek meg ennek a célnak. Az ipari környezet sajátos tulajdonságokat igényel, mint például: determinizmus (előrelátható viselkedés), kis zajérzékenység, gyors, meghatározott időben közvetített üzenetek, magas megbízhatóság, stb. Bizonyos esetekben a hálózat különös kéréseknek kell megfeleljen: biztonságos működés robbanásra veszélyes környezetben, adatközlés a táplálási kábeleken keresztül, vagy mobil kommunikáció (fizikai kapcsolat nélkül).

4. Tipikus folyamatvezérlési feladatok

A vezérelt folyamat igényei és tulajdonságai alapján több vezérlési módszert lehet alkalmazni:

a. Szekvenciális (sorrendi) vezérlés - amely állapot automatákra épül; a kimeneti vezérlő jelek, logikai függvények eredménye, amelyekben a bemeneti jelek és az automata állapota szerepel. A rendszer működését állapot-diagramokkal vagy idő-táblázatokkal lehet leírni. Jellemző készülék az ilyen fajta vezérlésben a programozható logikai vezérlő (PLC - Programable Logical Controller).

b. Zárt hurkú (visszacsatolt) szabályozás - amelynek célja elérni és fenntartani egy előre meghatározott folyamat-paraméter értéket. A vezérlő jel értékét a hiba nagysága és időbeli változása alapján szükséges megállapítani. Az egyszerű változatban a kimenetnek csak két állapota van (igen/nem, zárt/nyitott, kikapcsolt/bekapcsolt), amely csak a pillanatnyi eltéréstől (hibától) függ. Bonyolultabb viselkedésű folyamatoknál az eltérés alakulását is figyelembe kell venni. Ilyen esetekben PID (Proportional Integral Derivative) szabályozást alkalmazunk, amelyben a vezérlő jelet a pillanatnyi hiba, a hiba integrálja, és a hiba deriváltja függvényében állítjuk elő. A szabályozás minősége az együtthatók helyes beállításától függ.

c. Több szintes szabályozás - szükséges olyan esetekben, amikor a folyamat több paramétert összhangban kell szabályozni. A feladat megoldásához több hierarchikusan rendezett zárt hurkú szabályozót alkalmazunk. Különös gondot okoz a paraméterek közti összefüggés, amelyet nem lehet mindig pontosan kiértékelni, és matematikailag leírni.

d. Alkalmazkodó (adaptive control) és előrelátó (predictive control) szabályozás - amely olyan esetekben szükséges, amikor a vezérelt folyamat vagy a környezet lényegesen megváltoztatja a dinamikus viselkedését. Ilyen esetekben szükséges, hogy a szabályozó program önállóan tudja változtatni a saját szabályozási paramétereit, a környezeti változások függvényében. A prediktív szabályozás hasznosnak bizonyul mikor a folyamatnak hosszú holt ideje van (későn érzékelhető a vezérlés hatása), vagy gyakran jelennek meg mérési hibák. A szabályozó tartalmaz egy részt, amely meg tudja jósolni a folyamat jövőbeli alakulását és ennek alapján egy pontosabb vezérlő jelet tud előállítani.

e. Optimális vezérlés - amelynek célja egy általános költség-függvény csökkentése, vagy egy cél-függvény növelése. Általában az ilyen jellegű vezérlést egy több szintes folyamatvezérlési rendszer felsőbb szintjén alkalmazzák. Az optimális vezérlés több stratégia alapján működhet: a költségek vagy a nyersanyag-fogyasztás csökkentése, a termelés vagy a nyereség növelése, stb.

5. Számítógépen alapuló felügyelő és vezérlő rendszerek felépítése

A vezérelt folyamat komplexitásának függvényében különböző felépítésű rendszereket lehet alkalmazni:

a. Központosított vezérlés - feltételez egyetlen adat-feldolgozó egységet, és minden ki- és bemenő jel egy pontban van összegyűjtve. Ezt a megoldást egyszerű folyamatoknál lehet alkalmazni. Könnyű a megvalósítása és az egész rendszer átlátható. Viszont kevésbé megbízható, költséges a huzalozás, és hosszú a feldolgozási ciklus, abban az esetben, ha több paramétert kell felügyelni. Bármilyen meghibásodás esetén az egész rendszer leáll.

b. Hierarchikus felügyelés és vezérlés - a divide-et-impera elven alapszik, ami a vezérlési funkciók több szintre való elosztását feltételezi. Ez a megoldás komplex folyamatok

vezérlésére alkalmas, gyorsabb választ és jobb átláthatóságot biztosít. A gyors választ igénylő paraméter-szabályozást az alacsonyabb szintekre szükséges elhelyezni, viszont az optimális vezérlési meg folyamat felügyelési szolgálatokat magasabb szintekre kell helyezni. A megbízhatóság szempontjából a hierarchikus megoldás is korlátozva van, az egy pontból történő felügyelés miatt.

c. Osztott felügyelés és vezérlés - feltételez egy hálózaton alapuló több feldolgozó állomásos rendszert. A vezérlési alkalmazás funkcionalitását szolgálatokra kell osztani, amelyeket szerverek látnak el. A szerverek a hálózat különböző csomópontjaiban vannak elhelyezve. Biztonsági okokból egy szolgálatot több csomópont tud ellátni. Ez a megoldás komplex folyamatokra alkalmas, ahol sok a folyamat-paraméter és kritikusak az időhatárok. Az előnyök közül a következőket lehet említeni: olcsó huzalozás, olcsó készülékek szükségesek, a rendszer folyamatosan fejleszthető és meghibásodás esetén könnyen újrakonfigurálható. Ugyanakkor új feladatok merülnek fel a tervező számára, mint például: szinkronizálás, párhuzamos futtatás, valós-idejű programozás, címzés és hozzáférés távoli adatokhoz, kommunikációs protokollok, stb.

Az osztott vezérlőrendszerek megvalósításában fontos szerepet játszanak az ipari hálózatok. Ez a hálózat kategória a vezérlő alkalmazások sajátos kéréseinek felel meg: vezérlésre alkalmas üzenet-struktúra, gyors átvitel (előre kiszámítható), megbízható és biztonságos működés és alacsony megvalósítási költségek.

Egy teljes gyártási folyamat számítógéppel való vezérlése több szervezési szintet igényel. A 3. ábrán látható a CIM modell (Computer In Manufacturing), amely piramis formájában szervezi a vezérlési szinteket. A következő szintek szükségesek:

- tervezési és szervezési (adminisztratív) szint: terméktervezés, nyilvántartás, anyagbeszerzés
- felügyelési szint: optimális vezetés és felügyelés
- folyamat szint: gyártási folyamat vezetése
- terepszint: adatgyűjtés és vezérlés
- technológiai folyamat: gyártás



3. ábra. A CIM piramis modell

*CAD - Computer Aided Design - számítógépes tervezés

CAE - Computer Aided Engineering - számítógépes gyártáselőkészítés

CAM - Computer Aided Manufacturing - számítógéppel vezérelt gyártás

SCADA - Supervision Control and Data Acquisition - felügyelési és adatgyűjtési alkalmazás

PLC - Programmable Logic Controller - programozható logikai vezérlő

PID - Proportional Integrative and Derivative control - arányos, integratív és derivatív vezérlés

A számítástechnika alkalmazása az ipari vezérlésben gyakran egy bonyolult vállalkozás, amely több tervezési terület ismeretét igényli (pl. hardver tervezés, programozás, valós-idejű rendszerek tervezése, rendszerelmélet, digitális jelfeldolgozás, ipari hálózatok stb.). Sajátos fejlesztési eszközök és módszerek szükségesek ahhoz, hogy egy adott vezérlési alkalmazás különböző jellegű kéréseit megfelelően lehessen kielégíteni.

Irodalomjegyzék

- 1] D. Gorgan, G. Sebestyén: *Structura calculatoarelor*, Ed. Albastra, 2000.
- 2] P. Deshpande: *Computer Process Control*, Prentice Hall, 1987.
- 3] G. Olsson, G. Piani.: *Computer Systems for Automation and Control*, Prentice Hall, 1992.

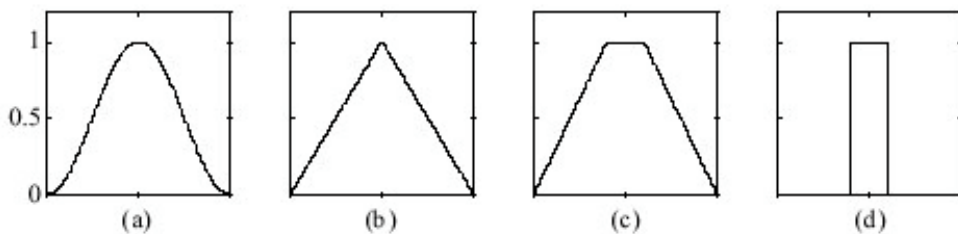
Fuzzy logika és alkalmazásai

Bíró Botond, VI. éves hallgató

Kolozsvári Műszaki Egyetem

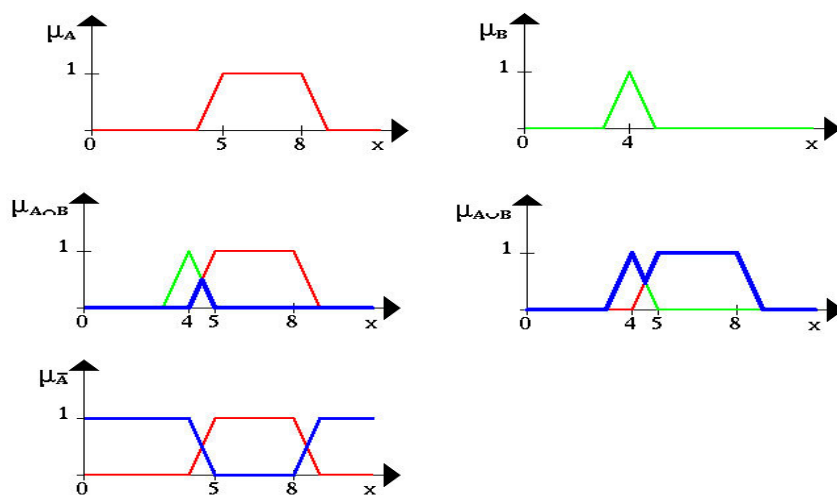
Fuzzy logika $\equiv$ minősítő logika

- Döntéshozatal - nyelvi változókkal leírt feltétel alapján történik. (pl. ha az autó *sebessége közepes és kicsit sáros* az út, akkor a féktávolság *kicsit nagy*)
- Eltérés a hagyományos logikától - az értékek nem diszkrét értékek már, hanem egy értékcsoporthoz tartoznak.
- „Hovatartozási függvények” adják meg, hogy egy hagyományos érték mennyire tartozik egy fuzzy értéktartományba.
-



Gauss görbe, b) háromszög alapú, c) trapéz alapú, d) egyedülálló (singleton)

- Fuzzy érték tartományok $\equiv$ „fuzzy halmazok”.
- Logikai műveletek - fuzzy halmazokon: és (min), vagy (max), nem.



Logikai műveletek fuzzy halmazokkal.

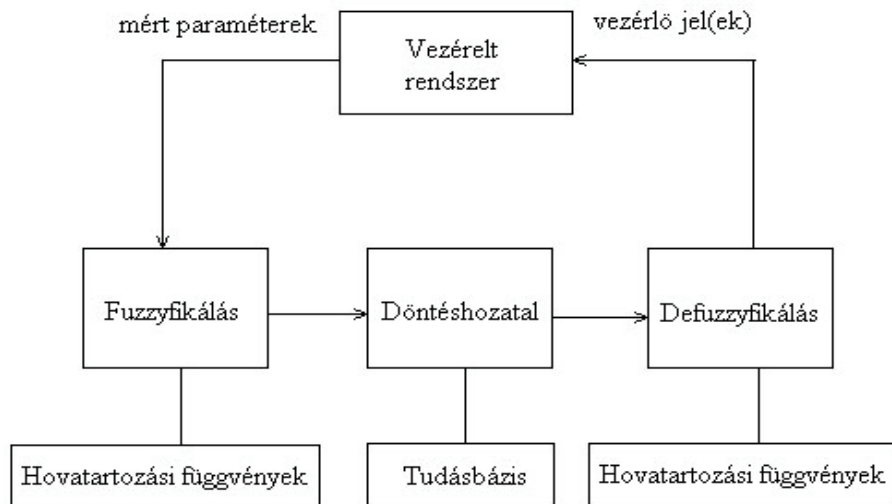
Mire jó ez?

Vezérlésre ott, ahol a szabályokat egyszerűbb verbálisan megadni, mint matematikai képleteket használni.

Egy példa fuzzy vezérlésre

A merev felfüggesztésű, fordított inga egyensúlyban tartása (*Mamdani* vezérlés).

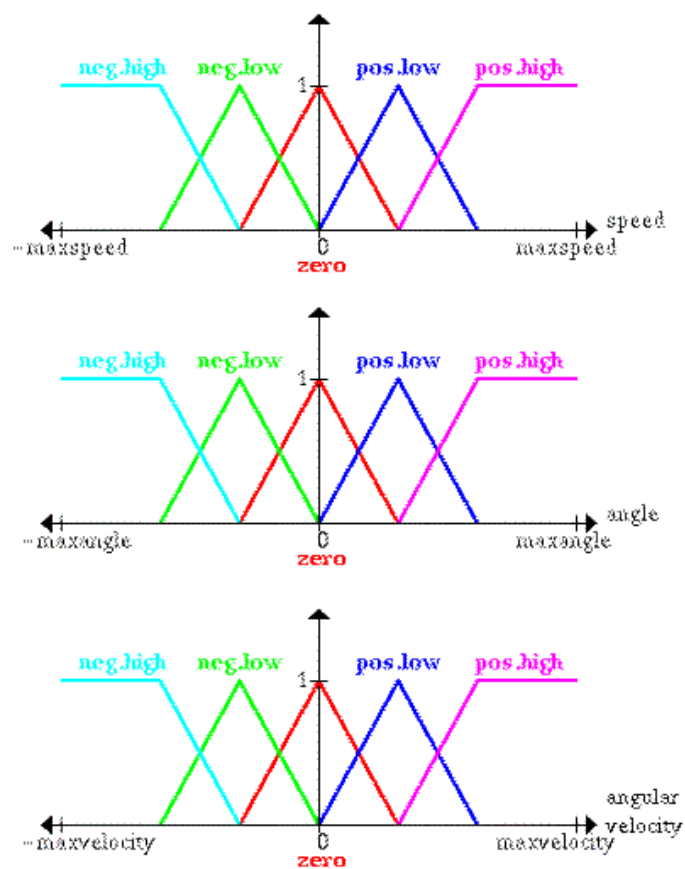
Egy fuzzy vezérlő blokk diagramja:



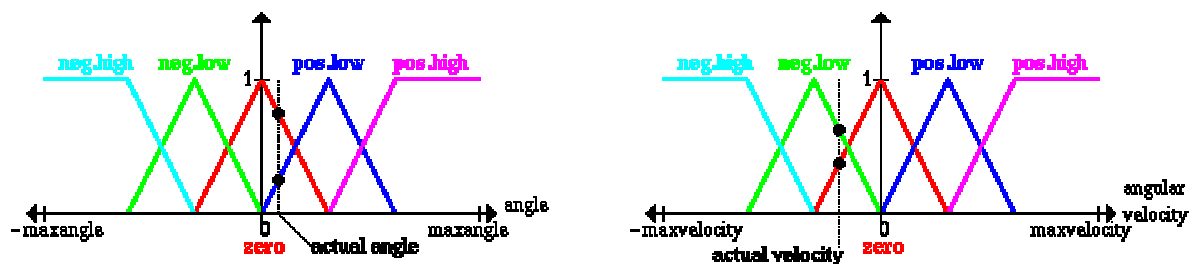
Szabályokat tartalmazó táblázat:

S z ö g s e b é s é g	Szög					
		NH	NL	Z	PL	PH
	NH			NH		
	NL			NL	Z	
	Z	NH	NL	Z	PL	PH
	PL		Z	PL		
	PH			PH		

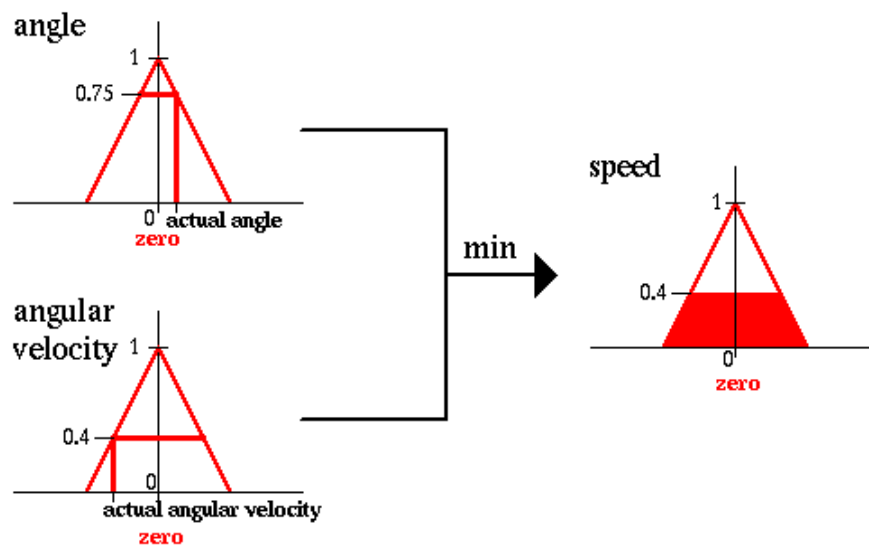
Hovatartozási függvények:



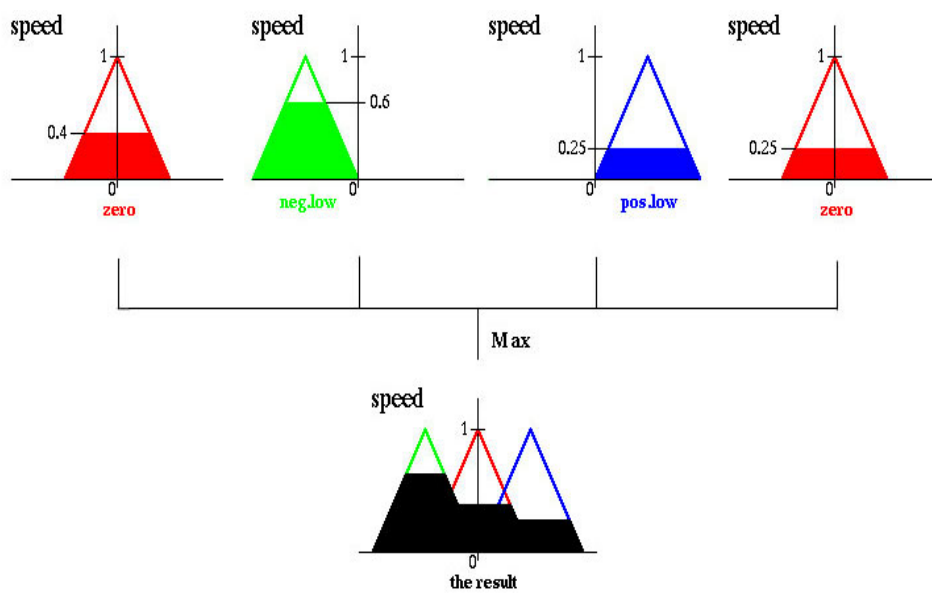
Fuzzyfikálás - eldönti, hogy a mért paraméterek (szög/szögsebesség) mely fuzzy halmazoknak az elemei:



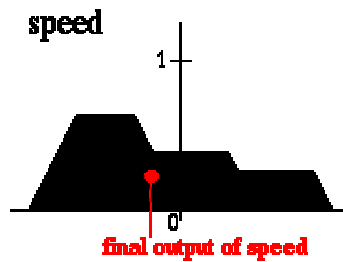
Döntéshozatal - a kapott fuzzy halmazok Descartes szorzatára alkalmazza a szabályokat (min - max módszer):



Mindegyik párra alkalmazva:



- Defuzzyfikálás: a kapott fuzzy értéket átalakítja valós értéké (súlypont módszer).
- A sebességhez hozzáadódik a kapott érték („a vezérlőjel integráló hatással dolgozik”).



Alkalmazások

- Hol ajánlott fuzzy vezérlést alkalmazni:
 - Nagyon bonyolult folyamatoknál, ahol nincs egy egyszerű matematikai modell.
 - Nagyon nem-lineáris folyamatoknál.
 - Ha lingvisztikailag meghatározott ismereteket kell feldolgozni.
- Hol nem ajánlott:
 - A hagyományos vezérlés elfogadható eredményt ad.
 - Egy könnyen megoldható és helyes matematikai modell már létezik.
 - A problémának nincs megoldása.
- Néhány valós alkalmazás:
 - Vízierőművek zsilipjeinek a vezérlésénél. (*Tokió Electric Pow.*)
 - Videokamerák fókuszálása sporteseményeknél. (*Omron*)
 - Hatékony és stabil autó-motor vezérlés. (*Nissan*)
 - Képfelismerés. (*Canon, Minolta*)
 - Rák diagnózis. (*Kawasaki Medical School*)
 - Szoftver tervezés.
 - Fogyó anyag tartalékolás. (*Hitachi*)
 - Buszmenetrend készítése. (*Toshiba*)

Irodalomjegyzék

- 1] Peter Bauer, Stephan Nouak, Roman Winkle: *A brief course in Fuzzy Logic and Fuzzy Control*
- 2] dr. Dávid László: *Logica fuzzy si controlul fuzzy*