Benesóczky Zoltán

DIGITÁLIS TERVEZÉS FUNKCIONÁLIS ELEMEKKEL ÉS MIKROPROCESSZORRAL

EGYETEMI TANKÖNYV

Műegyetemi Kiadó, 2006.

Szakmai lektor:

Dr. Selényi Endre
egyetemi tanár
BME Méréstechnika és Információs Rendszerek Tanszék

Szerkesztette, ábrákat rajzolta:

dr. Benesóczky Zoltán

Szerzők:

Az 1. melléklet kivételével: **dr.Benesóczky Zoltán**
Az 1. melléklet: **dr. Hainzmann János**

A tankönyv megjelenését a Művelődési és Közoktatási Minisztérium,
a Felsőoktatási Pályázatok Irodája által lebonyolított
felsőoktatási tankönyvtámogatási program
támogatása tette lehetővé

(Hatodik utánnyomás)

Azonosító: **55033**

ISBN 963 420 576 3

A Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
megrendelése alapján kiadja a
Műegyetemi Kiadó
www.kiado.bme.hu
Felelős vezető: Wintermantel Zsolt
Terjedelem: 18,6 (A/5) ív
Nyomdai munkák:
Műegyetemi Nyomda
Munkaszám: 5735/06

TARTALOMJEGYZÉK

ELŐSZÓ

Ez a tankönyv elsősorban a BME-n oktatott Digitális technika tantárgy II. félévi anyagának elsajátításához kíván segítséget nyújtani. Ennek megfelelően feltételezi az I. félévben tanultak (Boole algebra, kombinációs, szinkron és aszinkron hálózatok tervezése, stb.) ismeretét. Azonban ajánljuk mindenkinek, aki érdeklődik a digitális technika ill. az egyszerűbb mikroprocesszoros rendszerek tervezése iránt.

A könyv felépítésében az előadás anyagát követjük, sok helyen tervezési mintapéldákkal illusztráljuk az elmondottakat.

Az első fejezetben a kombinációs és sorrendi funkcionális elemeket ismertetjük.

A második fejezet a programozható logikákkal foglalkozik, a szakirányú digitális képzésre is gondolva, az átlagosnál nagyobb mélységben.

A harmadik fejezetben a funkcionális elemekkel való struktúrált tervezést (adatstruktúra-vezérlő felbontás, mikroprogramozott vezérlő tervezés) taglaljuk.

A negyedik fejezet az egyszerűbb mikroprocesszorok belső felépítését, működését és a mikroprocesszoros rendszer kialakítását (memória és periféra illesztés) mutatja be.

Az ötödik rész a mikrokontrollerekkel foglalkozik, elsősorban a széles körben elterjedt MCS8051 családra koncentrálva.

Az első és harmadik fejezethez a Digitális technika példatár [1] 'Tervezés MSI-vel' fejezetét ajánljuk a tanultak gyakorlására.

A könyv elkészítéséhez sok segítséget kaptam kollégáimtól. Ezért köszönet illeti Dr. Selényi Endrét, a színvonalas lektori munkáért és általános szakmai segítségéért, Dr. Horváth Gábort, a mikroprocesszorokról szóló fejezethez tett észrevételeiért, Dr. Fehér Bélát, a programozható logikákról szóló fejezethez adott előadás jegyzeteiért és szakmai javaslataiért, Dr. Hainzmann Jánost, az ABEL leírásért. Köszönöm tanszékvezetőmnek, Dr. Péceli Gábornak, hogy lehetővé tette a könyv nyugodt alkotói légkörben való megszületését. S végül, de nem utolsó sorban, köszönöm feleségemnek, Kóródi Magdolnának a sok-sok türelmet és segítséget.

1. FUNKCIONÁLIS ELEMEK

Amikor a technológia még csak a kis integráltságú SSI IC-k (Small Scale Integrated Circuits) megvalósítására volt képes, a mérnököknek a digitális tervezéshez az alapáramkörök, a kapuk és flip-flopok álltak rendelkezésre. Ezekkel az áramkörökkel megfelelő tervezési tapasztalatot szerezve kiderült, hogy bizonyos áramköri részletek azonos vagy nagyon hasonló formában újra és újra előkerülnek, ezért célszerű lenne, ha az alapáramkörökhöz hasonlóan ezek is egy IC-ben állnának rendelkezésre.

Amint a technológiai fejlődés lehetővé tette, megjelentek a különféle funkciókat egy tokban megvalósító IC-k, az MSI (Middle Scale Integrated Circuit) elemek. Ez nem egyszerűen csak a tervezési munkát könnyítette meg, de jelentős szemléletbeli változást is hozott a tervezés módszerében. Az MSI elemeket alkalmazva egyre kevésbé kellett törődni azzal, hogy milyen áramköri megoldások vannak az IC-kben elrejtve, így jobban a lényegre, a feladat funkcionális felépítésére lehetett koncentrálni. Ezen könyv írásakor már a digitális MSI IC-k nagy része is elavultnak tekinthető (a technológia túlhaladta). Ennek ellenére az egyes funkcionális elemek ismertetésénél felsorolunk néhány olyan típust a legelterjedtebb 74-es sorozatból, melyek az adott funkciót valósítják meg.

Funkcionális elemen nem csak valamely funkciót egy IC-ben megvalósító áramkört értünk, hanem egy *a tervező számára egy egységként kezelhető* blokkot, melynek részletes belső felépítését nem szükséges ismerni az alkalmazáshoz. A bonyolult LSI (Large Scale IC), VLSI (Very LSI) IC-k belsejének tervezésénél is ilyen formában használjuk a funkcionális elemeket. A mai tervező rendszerek lehetővé teszik, hogy a számítógépen a megszokott (önálló MSI IC-ként is létező) funkcionális elemeket is felhasználva tervezzük meg egy LSI IC belső kapcsolási rajzát, melyből a tervező rendszer elkészíti az áramkör létrehozásához szükséges összes dokumentációt.

A funkcionális elemeket hagyományosan két csoportba osztjuk aszerint, hogy kombinációs hálózatot, vagy sorrendi hálózatot valósítanak meg. Itt is ebben a csoportosításban szerepelnek, azonban ebből külön kiemeltük az ún. programozható logikákat és a memóriákat. Az alább ismertetett elemek sora a terjedelmi korlátok miatt nem teljes, csak a leggyakrabban alkalmazottakat tárgyaljuk.

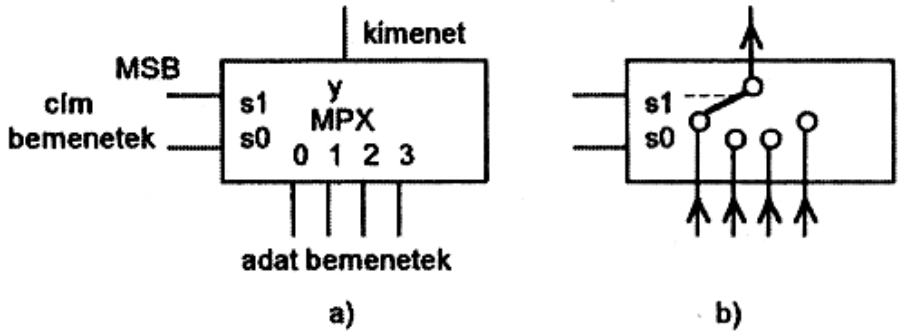
1.1. Kombinációs funkcionális elemek

1.1.1. Multiplexer

A multiplexer funkciója, hogy az n bemeneti jelből a kimenetére kapcsolja a címbemeneteken kiválasztott sorszámút. A multiplexerek méretét az (adat csatornák száma)/1 jelöléssel szokás megadni.

A multiplexernek az n sorszámozott jelbemenetből való választásához $\lceil \log_2 n \rceil$ számú cím bemenete (Sm-S0) van. A címbemenetekre adott értéket bináris számként értelmezve S0 a legkisebb helyiértékű bit. (Az ABC-t is használják a címbemenet jelölésére, ebben az esetben az A-val jelölt a legkisebb helyiértékű bit. Erre a bitre gyakran hivatkoznak az LSB jelöléssel, az angol Least Significant Bit rövidítéseként. A legnagyobb helyiértékű bit jelölése MSB, az angol Most Significant Bit alapján)

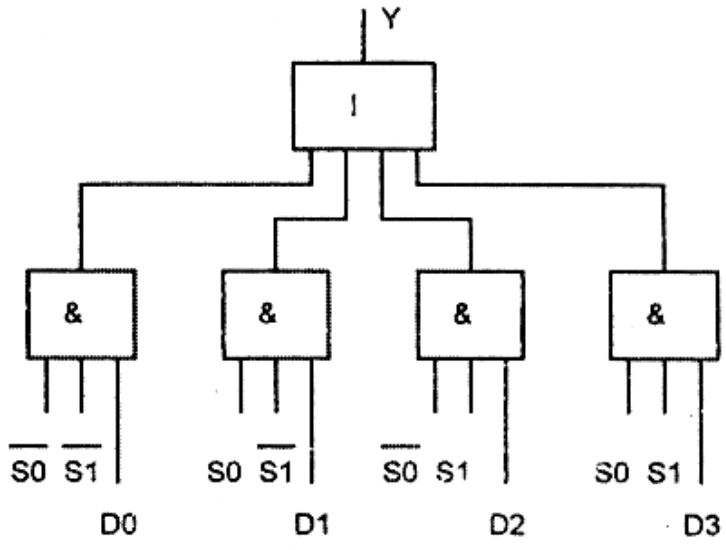
Az 1.1a ábra egy 4/1-es multiplexer jelölését mutatja. Az 1.1b ábrán a megvalósított funkciót szemléltetjük egy több állású kapcsolóval. Meg kell jegyezni, hogy a digitális multiplexer a kapcsolóval ellentétben csak egy irányban működik. (Léteznek mindkét irányban működő analóg multiplexerek, de ezekkel itt nem foglalkozunk.)



1.1. ábra. Multiplexer

Az 1.2. ábrán a multiplexer logikai vázlata látható. Ez egy kétszintű kombinációs hálózat, ahol a második szinten levő ÉS kapuk közül egy-egy címkombináció egy kaput engedélyez, s ennek az adatbemenetén levő jel az első szint VAGY kapuján keresztül eljut a kimenetre. (Az ábrákon a bemenetek negáltját előállító invertereket általában nem tüntetjük fel.)

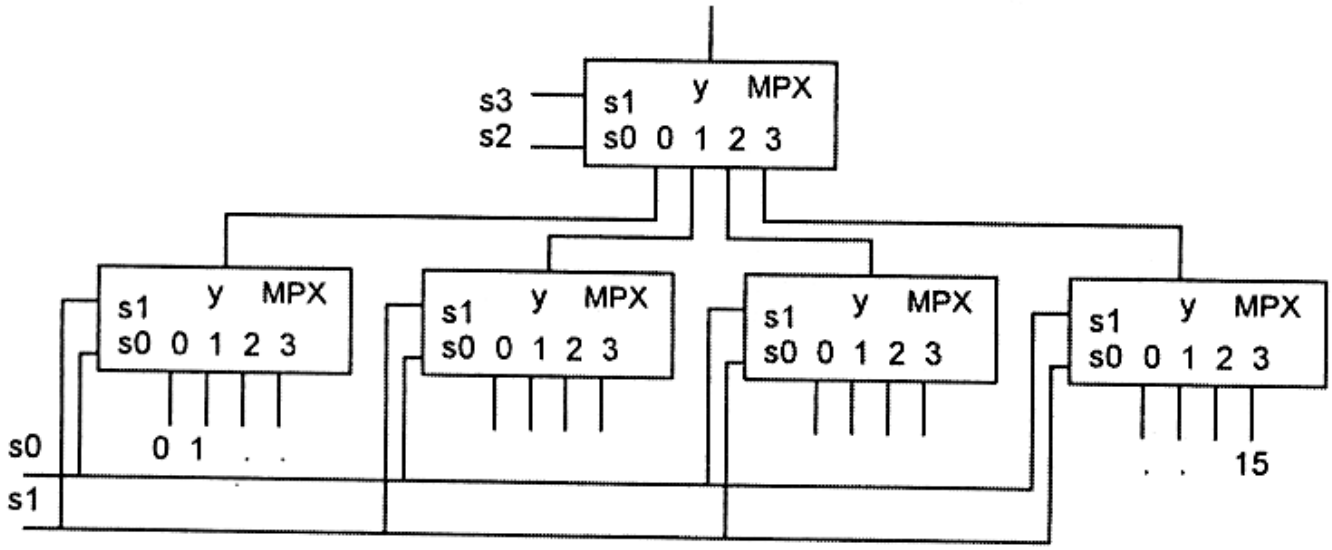
MSI elemként többnyire n/1-es multiplexert gyártanak. Ha egy IC pl. 4 két bit széles jelcsoport (busz) multiplexelésére alkalmas, azt 2x4/1-es multiplexernek nevezik (a felépítésére utalva).



1.2. ábra. A multiplexer logikai vázlata

Multiplexerek csatornaszám növelése

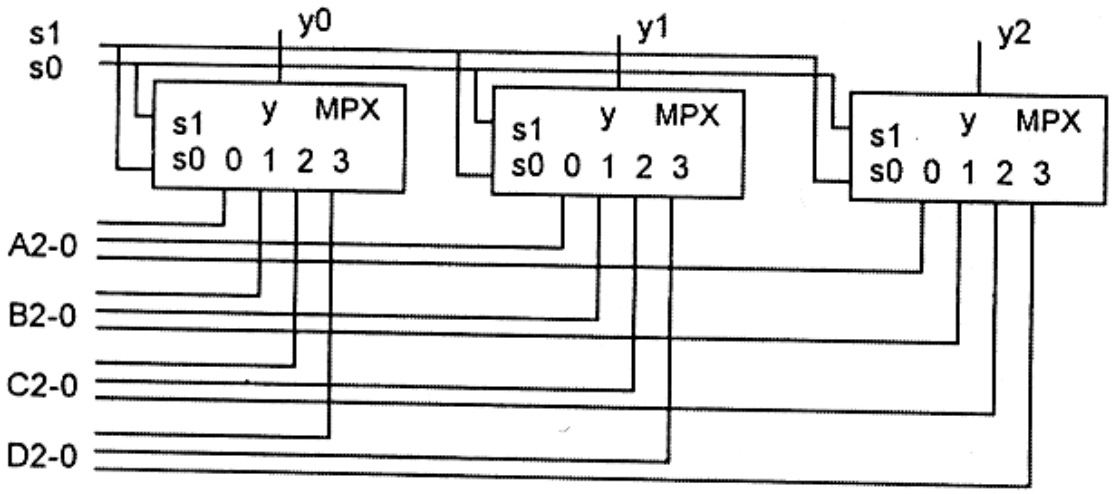
Ha a rendelkezésre álló multiplexer csatornáinak száma nem elegendő, akkor több multiplexer felhasználásával megnövelhetjük azt. Az 1.3. ábrán 4/1-es multiplexerekből csináltunk 16/1-est. Az elv az, hogy egy multiplexer bemeneteire újabbakat kapcsolunk, s az azonos szinten levő multiplexerek címbemeneteit közösítjük. Az első szinten levő multiplexer címbemenetei kiválasztják az alatta levő szint egyik multiplexerének kimenetét, ezen szint multiplexerének címe pedig kiválasztja a kívánt bemenetet (vagy egy alatta levő szint multiplexerének kimenetét). A szintek számát elvileg tetszőlegesen lehet növelni, de minden egyes szint növeli a teljes áramkör késleltetését. A bemenethez legközelebbi multiplexer legkisebb címbitje adja a teljes multiplexer legkisebb helyiértékű címbitjét (LSB).



1.3. ábra. Multiplexer csatornaszám növelése

Multiplexerek csatorna szélességének bővítése

Sokszor előfordul, hogy egyszerre több jelet (több bit széles csatornát, amit *busz*nak is szokás nevezni) kell multiplexálni. Ennek megoldására példa az 1.4. ábra.



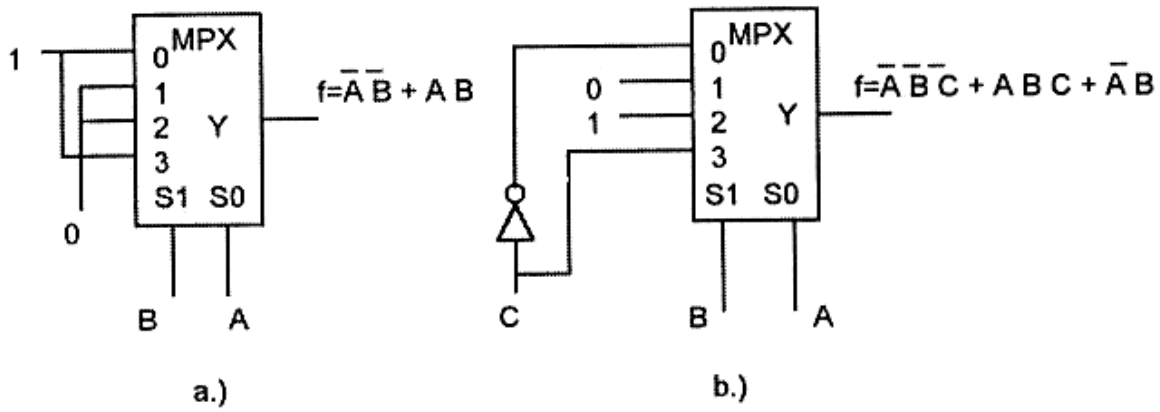
1.4. ábra. Multiplexer csatorna szélesség bővítése

(Itt 3 db 4/1-es multiplexerből készítettünk egy 4 db 3 bit széles busz (A2-0, B2-0, C2-0, D2-0) multiplexálására alkalmas áramkört). Minden egyes multiplexer *i*-edik bemenete az *i*-edik csatorna egy-egy bitjét adja, s a címbemenetek közösítve vannak. Pl. s1s0=00 esetén y2-y0=A2-A0.

Multiplexer mint univerzális kombinációs hálózat

Ha multiplexert univerzális kombinációs hálózatként akarjuk használni, akkor annak címbemeneteire a logikai változókat kapcsoljuk, az *i*-edik adatbemenetére pedig 1-et, ha a megvalósítandó logikai függvényben szerepel az *i*-edik minterm, egyébként 0-át (az igazságtáblát valósítjuk meg). Ekkor egy n címbemenetű multiplexerrel egy tetszőleges n változós függvényt realizálhatunk.

Pl. az $f = \overline{A \oplus B}$ függvény realizálása multiplexerrel (1.5a ábra).



1.5. ábra. Logikai függvény realizálása multiplexerrel

Egy n bemenetű multiplexerrel tetszőleges n+1 változós logikai függvény is realizálható, ha invertereket is felhasználhatunk. Ekkor az n+1-edik változót ponálva vagy negálva kell azoknak a termeknek megfelelő multiplexer bemenetekre kötni, amelyekben ponálva ill. negálva szerepel, ahogy az 1.5b ábra mutatja.

Multiplexert kombinációs hálózatként ma már nem alkalmazunk, azonban egyes programozható logikák belsejében az elvet ma is használják.

Néhány konkrét multiplexer a 74-es sorozatból: 74157 (4x2/1), 74150 (16/1), 74153 (2x4/1).

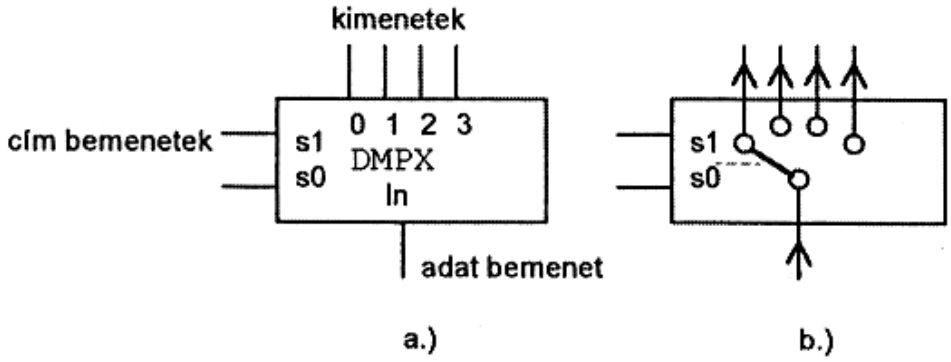
1.1.2. Dekóder/demultiplexer

A demultiplexer

A demultiplexer a multiplexerével ellentétes funkciót valósít meg. Feladata, hogy az egyetlen bemenetén levő jelet kapcsolja rá a címbemenetein megadott sorszámú kimenetére. A demultiplexer méretét a (címbemenetek száma)/(kimenetek száma)

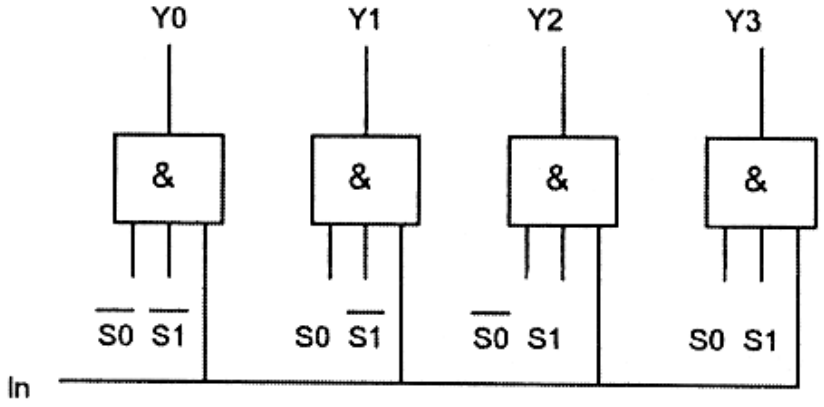
jelöléssel szokás megadni. Egy adatbemenete, n sorszámozott adatkimenete és $\lceil \log_2 n \rceil$ címbemenete van.

Az 1.6a. ábrán egy 2/4-es demultiplexer látható. Az 1.6b ábra egy többállású kapcsolóval illusztrálja a megvalósított funkciót. (Megjegyezzük, hogy a digitális demultiplexer a kapcsolóval ellentétben csak egy irányban működik.)



1.6. ábra. Demultiplexer

A demultiplexer/dekóder belső felépítését tekintve (1.7. ábra) annyi ÉS kapu, ahány kimenete van. Az egyes kapuk engedélyezése ugyanolyan, mint a multiplexer esetében.



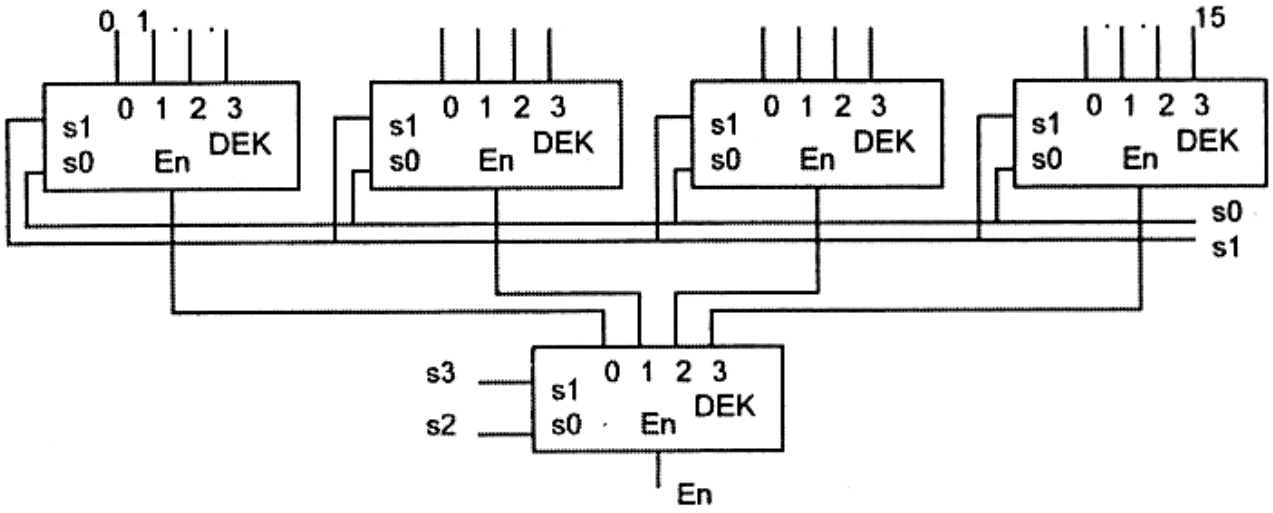
1.7. ábra. Demultiplexer logikai vázlata

A dekóder

Ha egy demultiplexer bemenetét fixen aktív szintre kötjük (ez demultiplexertől függően 0 vagy 1), akkor dekódert kapunk. Egy ilyen dekódernek mindig csak egy kimenete aktív, mégpedig a címbemenetekkel kiválasztott sorszámú. (Az egyes kimenetein dekódolja, hogy milyen bináris szám van a címbemeneteken.) Dekóder funkció esetén az adat bemenetet engedélyező bemenetnek nevezik. (Ha az engedélyező bemenet inaktív, az összes kimenet is az.) Az MSI dekóderek alacsony aktív kimenettel rendelkeznek. A gyakorlatban a demultiplexer/dekódereket többnyire dekóder funkcióban alkalmazzuk, ezért a továbbiakban röviden csak dekódernek nevezzük.

Dekóderek bővítése

A dekóderek kimenetszámának növelését mutatja az 1.8. ábra. Itt 2/4-esekből készítettünk 4/16-ost. Az elv az, hogy egy dekóder kimenetei további dekóderek engedélyező bemeneteire kapcsolódnak s az azonos szinten levő dekóderek címbemenetei közösítve vannak.

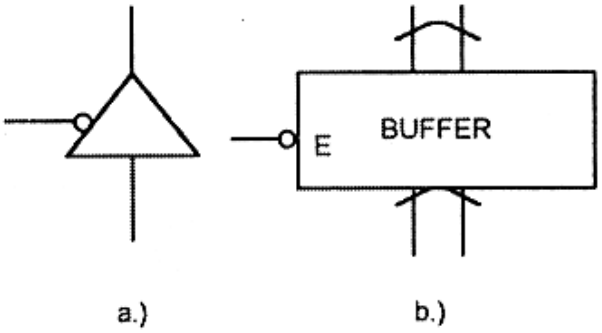


1.8. ábra. Dekóder bővítése

Az előbbi dekóder engedélyezi a címbemenetén (S3-S2) megadott sorszámú dekódert, az utóbbiak közül pedig az engedélyezettnek azon kimenete válik aktívvá, amelyet a közösített címbemeneteken megadunk. Így a kimenetet adó dekóderek legkisebb helyiértéke adja az egész egység LSB-jét. Néhány konkrét dekóder a 74-es sorozatból: 74138 (3/8), 74154 (4/16), 74154 (2x2/4).

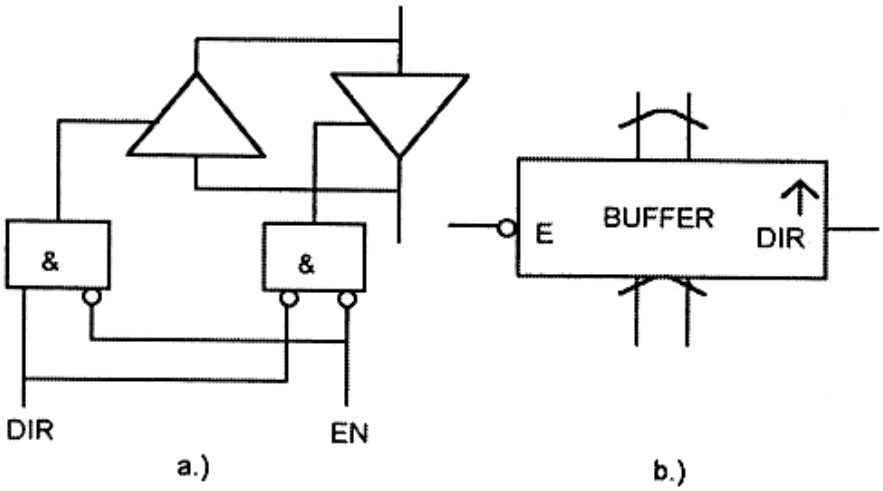
1.1.3. Három állapotú meghajtó (buffer)

A három állapotú meghajtó egy olyan áramkör, amely kimenetén vagy a bemenetének megfelelő logikai szint jelenik meg, vagy pedig a kimenete úgynevezett harmadik állapotban van, ami itt elfogadható közelítéssel olyan, mintha az egy sehova sem kapcsolódó vezeték darab lenne. A harmadik állapotba a vezérlő (engedélyező) bemenetével kapcsolható, ha az inaktív logikai szinten van. Az MSI IC-knek alacsony aktív az engedélyező bemenete.



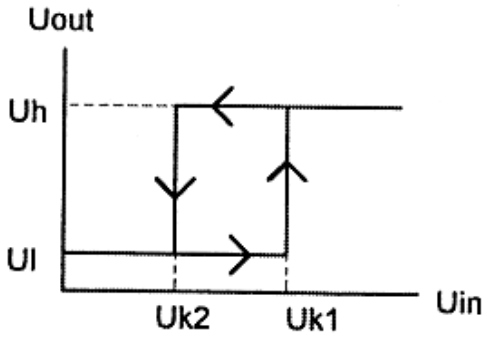
1.9. ábra. Egyirányú, három állapotú meghajtó

Mind önállóan, mind buszmeghajtóként létezik. Előbbi esetben az 1.9a ábra szerinti, utóbbi esetben az 1.9b ábra szerinti jelölést szokás használni. Kétirányú változatának jelölését mutatja az 1.10b ábra, belső felépítését pedig az 1.10a ábra.



1.10. ábra. Kétirányú, három állapotú meghajtó

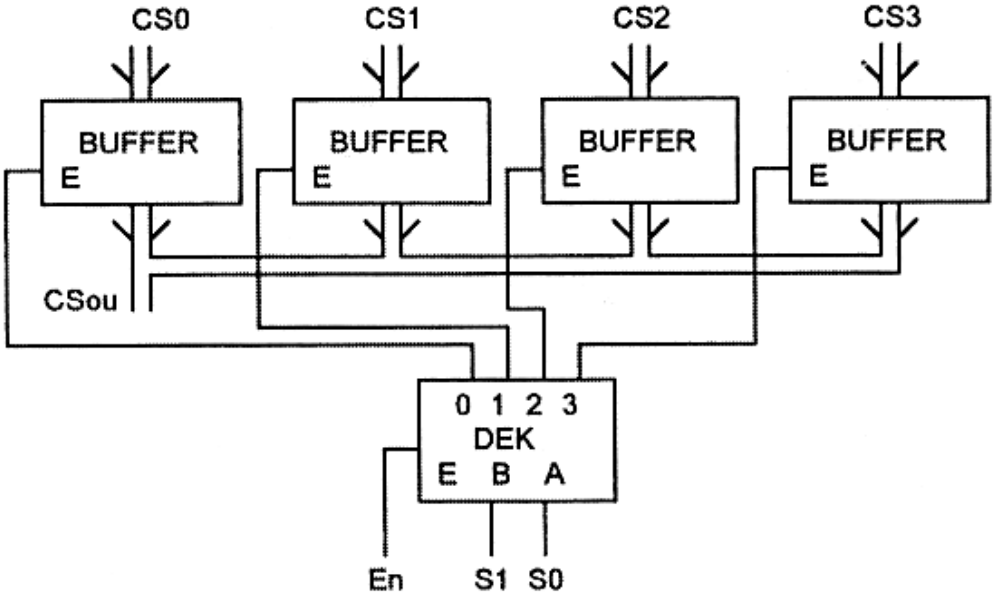
A DIR bemenettel választható ki az adatáramlás iránya, az EN bemenettel engedélyezhető a kiválasztott kimenet. Régebben csak 8 bit széles változat létezett, ma már nagyobb szószélességű változatok is vannak (a 16, 32 stb. bites processzorok elterjedése miatt). A három állapotú bufferek sok esetben ún. Schmidt-triggeres bemenetűek. Ez azt jelenti, hogy a bemenő feszültséget növelve egy Uk1 feszültségnél érzékelik a magas logikai szintet, viszont ezután csökkentve, egy az előbbinél kisebb Uk2 feszültségnél veszik észre az alacsony szintet (1.11. ábra). Így a bemenőjelen levő, az Uk1-Uk2-nél kisebb amplitúdójú zajra érzéktelenek (azonban ennek mértékét a bemenő jel ill. a zaj frekvenciája is befolyásolja).



1.11. ábra. Schmidt triggeres bemenet viselkedése

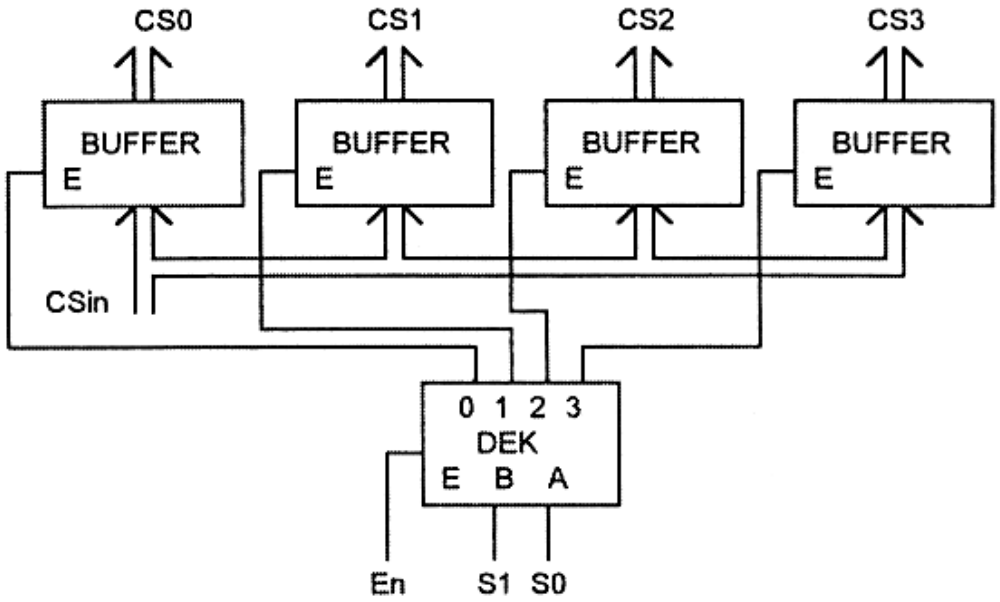
Ilyen áramkörök segítségével könnyen megvalósítható a buszok multiplexálása, demultiplexálása, mivel a normál (totem pole) kimenetű áramkörökkel ellentétben a három állapotú kimenetek összeköthetők, amennyiben biztosítjuk, hogy az összekötöttek közül egyszerre mindig csak egy kimenet aktív. Mikroprocesszoros rendszerekben előszeretettel alkalmazzuk őket, multiplexálási, demultiplexálási célra.

Az 1.12. ábrán egy három állapotú meghajtókból felépített, 4 db busz multiplexelésére képes áramkör látható. Mivel az egyes bufferek jelkésleltetése különböző, ha a engedélyezés közben váltunk címet, előfordulhat, hogy az előzőleg engedélyezett buffer még nem kapcsolódott ki, amikor az aktuálisan kiválasztott már aktív, ami a két meghajtó kimeneteinek szembekapcsolása miatt meghibásodáshoz vezethet. Ennek kiküszöbölésére a címváltások előtt célszerű letiltani a dekóder engedélyezését, s csak a tranziensek lezajlása után engedélyezni azt.



1.12. ábra.

Demultiplexáló áramkört úgy kapunk, ha a bufferek irányát megfordítjuk (1.13. ábra), ilyenkor természetesen az előbbi probléma nem jön elő.



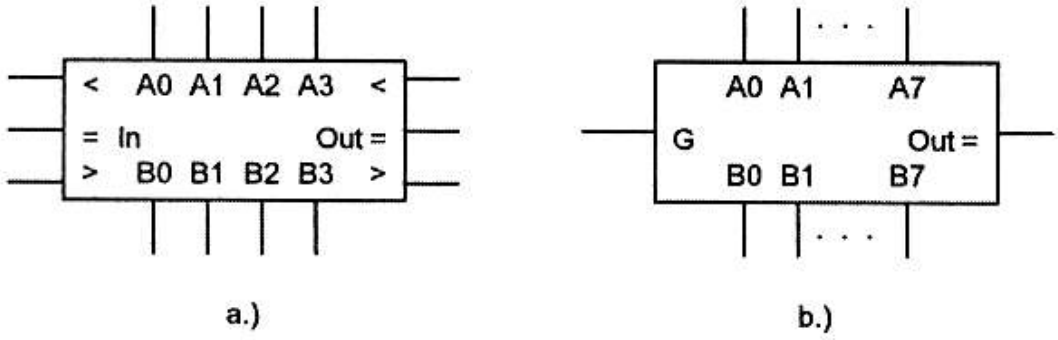
1.13. ábra.

Néhány konkrét 3 állapotú meghajtó a 74-es sorozatból: 74125 (4x1), 74LS244 (2x4), 74LS245 (1x8 kétirányú).

1.1.4. Komparátor

A komparátor funkciója, hogy jelezze a két n bites előjel nélküli abszolútértékes ábrázolású szám (A és B) közötti relációt. A komparátornak általában két n bites (MSI IC-k esetén n= 4, 8) adat bemenete (An-1-A0 és Bn-1-B0), A<B, A=B, A>B kimenete és A<B, A=B, A>B kaszkádosító (bővítő) bemenete van. Az előbbiek szerinti 4 bites komparátor jelölését mutatja az 1.14a ábra.

Létezik olyan komparátor is, amely csak az egyenlőséget figyeli. Itt az egyenlőség bemenetet sokszor G-vel jelölik. Egy ilyen 8 bites komparátort mutat az 1.14b ábra.



1.14. ábra. Komparátor

Egy kaszkádosítható 4 bites komparátor (7485) belseje az alábbi logikai függvényeket realizálja:

$$O_{A=B} = (A_3 \oplus B_3)(A_2 \oplus B_2)...(A_0 \oplus B_0)I_{A=B}$$

$$O_{A>B} = \overline{\overline{A}_3 B_3}\,\overline{(A_3 \oplus B_3)\overline{A}_2 B_2}\,\overline{(A_3 \oplus B_3)(A_2 \oplus B_2)\overline{A}_1 B_1}\,\overline{(A_3 \oplus B_3)(A_2 \oplus B_2)(A_1 \oplus B_1)\overline{A}_0 B_0}\,\overline{(A_3 \oplus B_3)(A_2 \oplus B_2)(A_1 \oplus B_1)(A_0 \oplus B_0)I_{A<B}}$$

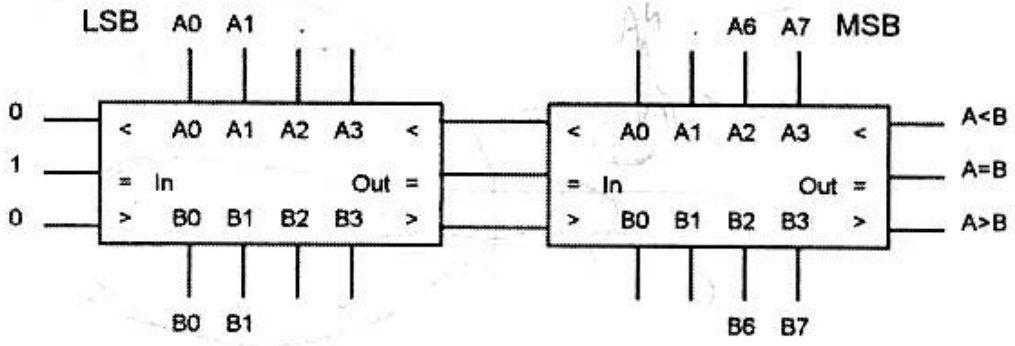
$$O_{A<B} = \overline{A_3\overline{B}_3}\,\overline{(A_3 \oplus B_3)A_2\overline{B}_2}\,\overline{(A_3 \oplus B_3)(A_2 \oplus B_2)A_1\overline{B}_1}\,\overline{(A_3 \oplus B_3)(A_2 \oplus B_2)(A_1 \oplus B_1)A_0\overline{B}_0}\,\overline{(A_3 \oplus B_3)(A_2 \oplus B_2)(A_1 \oplus B_1)(A_0 \oplus B_0)I_{A>B}}$$

A kisebb ill. nagyobb kimenet logikai megfogalmazása tagadó állítások ÉS kapcsolatából alakul ki. A>B, ha a 3. bitek nem jelzik az ellenkezőjét ÉS ha a 3. bitek egyeznek, de a 2. bitek nem jelzik az ellenkezőjét ÉS ... ÉS ha az összes bit megegyezik, de a megfelelő kaszkádosító bemenet nem jelzi az ellenkezőjét.

Az egyenlőség kimenet egyszerűen a bitenkénti egyezések ÉS kapcsolata.

Hagyományos kaszkádosítás

A kaszkádosító bemenetek segítségével az 1.15. ábrán látható módon tetszőleges számú komparátor sorba köthető, megnövelve ezzel az összehasonlítható számok szószélességét. Természetesen az ilyen módon történő kaszkádosítás megnöveli a teljes komparátor késleltetését. A komparátorok kialakítása olyan, hogy a kisebb helyiérték felől kell kaszkádosítani. *A legelső komparátor kaszkádosító bemeneteit úgy kell beállítani, mintha egy előző komparátor egyenlőséget jelezne.* Komparátorral a 2-es komplemensben ábrázolt számokat az előjel bit invertálása után tudjuk összehasonlítani (offszet kóddá transzformálás).



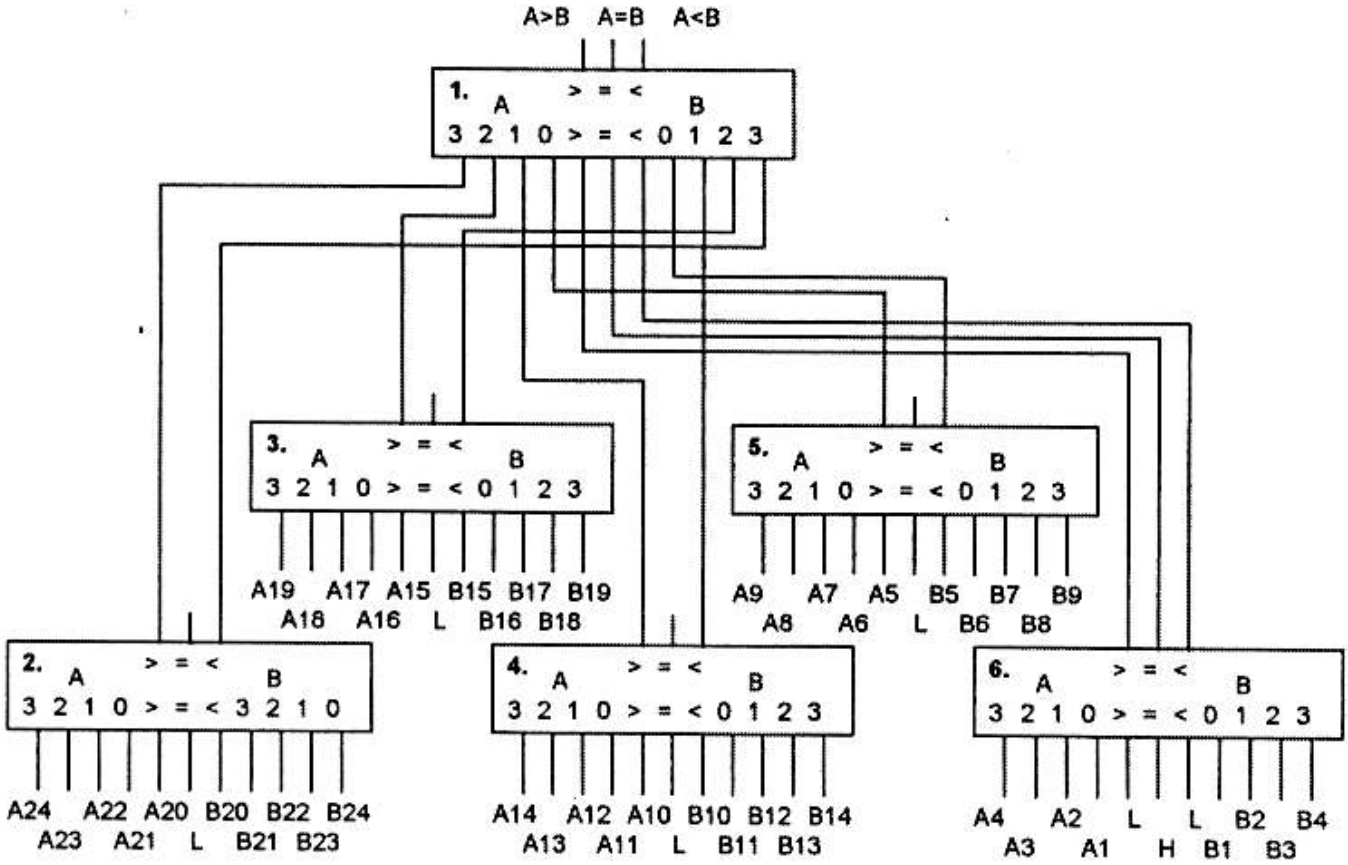
1.15. ábra. Komparátor hagyományos kaszkádosítása

Gyorsított kaszkádosítás

Az 1.15. ábra szerinti hagyományos kaszkádosítás hátránya, hogy a kaszkádosított egységek késletetési ideje összeadódik. Ezt kerüli ki az 1.16. ábrán mutatott ötletes módszer. Itt 4 bites komparátorok összekapcsolásával egy gyors 24 bites komparátort alakítottunk ki. A kapcsolás olyan komparátort tételez fel, amelynek < és > bementére egyszerre 1-et kapcsolva, annak < és > kimenete egyforma logikai értéket ad. Ilyen pl. a 7485 típusszámú.

Ha a 2. komparátor a legnagyobb helyiértékű 5 bit összehasonlításának eredményeképpen A>B-t jelez, akkor az 1. komparátor -mely az egész kapcsolás kimenetét állítja elő- szintén, mert a 2. komparátor eredményt jelző kimenetei az 1. komparátor legnagyobb helyiértékű bemeneteire vannak kötve, s komparálásnál a nagyobb helyiérték a döntő. Ha a legfelső 5 bit azonos, akkor hasonló módon az következő 5 bit összehasonlításának eredménye fog dönteni, stb. Az összes bit azonossága esetén az 1. komparátor adat bemenetire azonos érték kerül (a 2.-5. komparátorok kisebb és nagyobb kimenetei egyforma szintet jeleznek). Az 1. és 6. komparátorok hagyományos módon vannak kaszkádosítva, ami a felső bitek azonossága esetén biztosítja a legkisebb helyiértékű bitek alapján történő döntés helyességét.

Néhány konkrét komparátor a 74-es sorozatból: 7485 (4 bites, kaszkádosítható), 74LS688 (8 bites, csak egyenlőséget jelez).

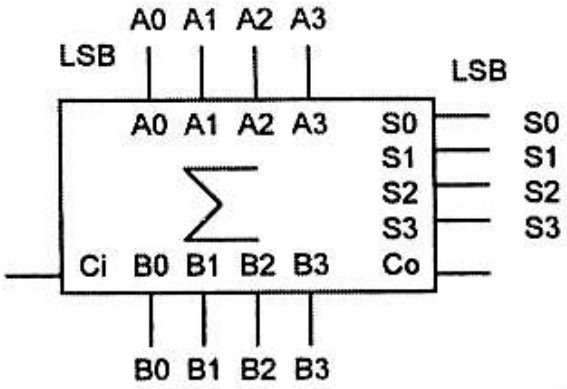


1.16. ábra. Komparátor gyorsított kaszkádosítása

1.1.5. Összeadó

Az összeadó két n bites (előjel nélküli abszolútértékes vagy 2-es komplemens ábrázolású) bináris szám összeadását végzi, ami az Sn-1-S0 (SUM) n bites kimenetén és a Co (átvitel) kimeneten jelenik meg. A kaszkádosításhoz a Co kimenet mellett a Ci (áthozatal az előző egységtől) bemenet szükséges. Egy 4 bites összeadó jelölését mutatja az 1.17. ábra.

Az összeadó által megvalósított logikai függvényeket az egyszerűség miatt egy 1 bites összeadón mutatjuk be. A bináris összeadás igazság táblázatát mutatja az 1.1. táblázat. Az összeadó működése alapján belátható, hogy az összeg kimenet akkor ad 1-et, ha az összeadandók és a carry bemenet összesen 1 vagy 3 darab 1-est tartalmaznak. Az összeg logikai függvényét mutatja az (1.1) képlet. (Az olyan függvényeket, amelyek kimenete csak a bemenetükön levő egyesek számától függ, szimmetrikus függvényeknek nevezik és $S^n_{i,j,\ldots}$-vel jelölik, ahol n a függvény változóinak száma, az alsó indexek pedig az egyesek azon számát jelölik, amelyekre a függvény 1 kimenetet ad.)



1.17. ábra. Összeadó

A	B	Ci	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

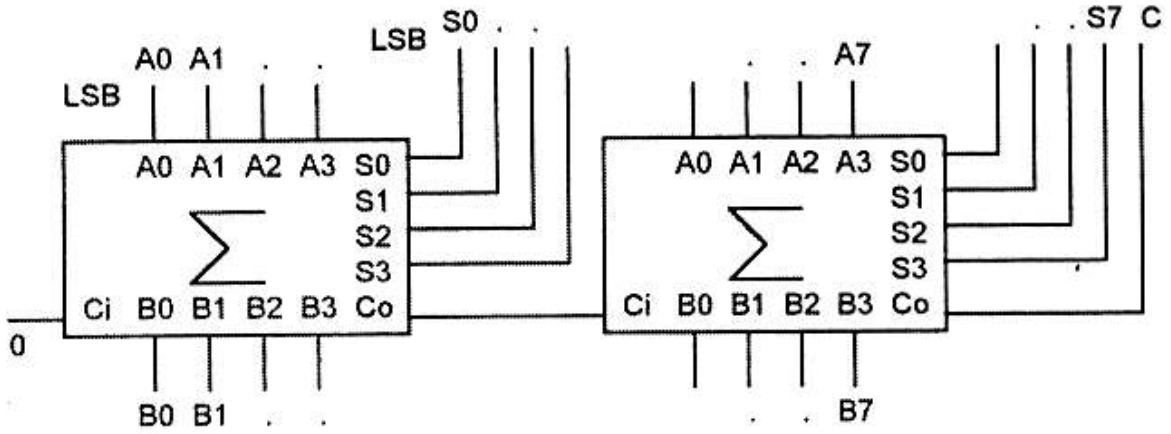
1.1. táblázat. 1 bites összeadó igazságtáblázata

$$S_0 = \overline{A}_0\overline{B}_0C_1 + \overline{A}_0B_0\overline{C}_1 + A_0\overline{B}_0\overline{C}_1 + A_0B_0C_1 = A_0 \oplus B_0 \oplus C_1 = S^3_{1,3} \qquad (1.1)$$

A carry kimenet akkor ad 1-est, ha az összeadandók és a carry bemenet összesen 2 vagy 3 darab 1-est tartalmaznak. Ennek logikai függvényét mutatja az (1.2) képlet.

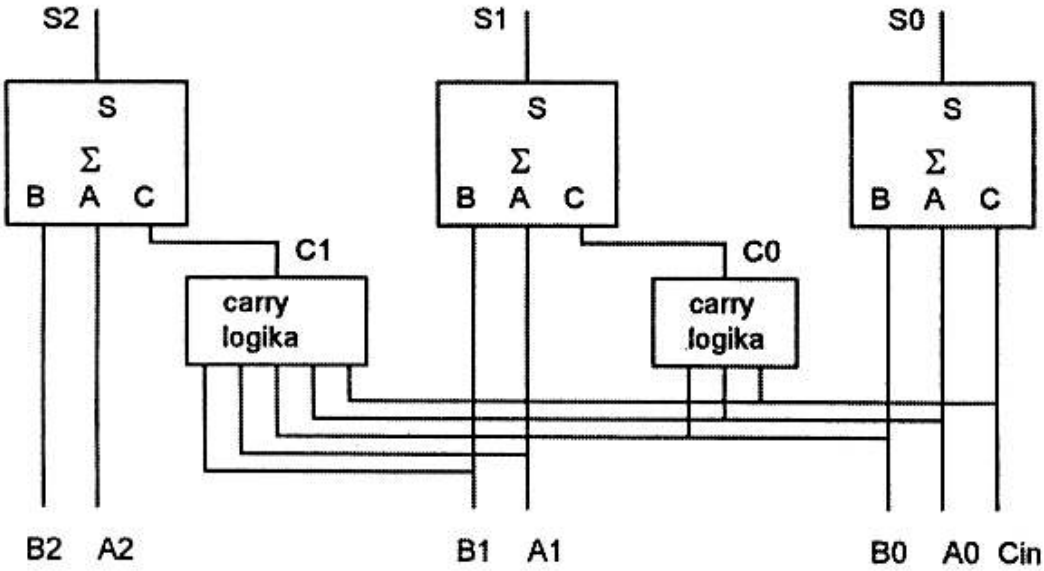
$$C = A_0B_0\overline{C}_1 + A_0C_1\overline{B}_1 + B_0C_1\overline{A}_1 + B_0C_1A_1 = A_0B_0 + A_0C_1 + B_0C_1 = S^3_{2,3} \qquad (1.2)$$

Két 4 bites összeadó kaszkádosítását mutatja az 1.18. ábra. A legelső összeadó Cin bemenetére 0-át kell kötni.



1.18. ábra. Összeadók kaszkádosítása

Az összeadóknál problémát jelent, hogy a fenti felépítés sok bit esetén nagy késleltetést okoz (az átvitel soros terjesztése miatt). Ennek elkerülésére ún. átvitel gyorsítást alkalmaznak. Ennek lényegét mutatja az 1.19. ábra. A gyorsítást egyszerűen azzal érik el, hogy a carryt minden egyes bithez önállóan állítják elő, az összes előző bit figyelembe vételével. Természetesen ez a nagyobb helyiértékek felé egyre nagyobb bemenetszámú kombinációs hálózatot igényel, ez adja a megoldás hátrányát is. Azonban így is sok késleltetési időt meg lehet spórolni, ha egy-egy néhány bites egységen belül párhuzamosan állítjuk elő a carryt, s ezeket az egységeket a szokásos módon kaszkádosítjuk.



1.19. ábra. Összeadó gyorsítása párhuzamos carry logikával

Néhány 74-es sorozatú összeadó: 7480 (1 bites teljes összeadó), 7483 (4 bites teljes összeadó).

1.2. Sorrendi funkcionális elemek

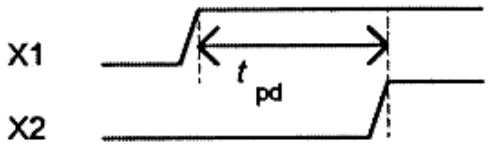
1.2.1. Időzítési alapfogalmak

A sorrendi funkcionális elemek működtetésénél nagy jelentősége van a katalógusokban előírt időzítések betartásának. Ezért az alábbiakban először a legfontosabb időzítési fogalmakat ismertetjük. Az időzítéseknél a katalógusok megadják a tipikus, minimális ill. maximális értékeket (ha van ilyen). A definiciók megtalálhatók a [3] TTL Data Bookban.

Késleltetési idő (t_{pd} propagation delay time)

Az áramkör egy megadott bemenetének megváltozását követően (amikor a bemeneti feszültség eléri az L ill. H szintet) ennyi idő múlva változik meg a megadott kimenet (éri el a definiált H ill. L szintet), amint azt az 1.20. ábra mutatja. Ez a fogalom már a

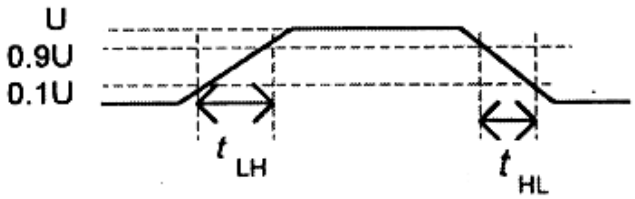
kombinációs hálózatoknál is előjön. Különböző irányú tranziensek (H-L, L-H) esetén a késleltetési idők eltérőek lehetnek.



1.20. ábra. Késleltetési idő

Fel- ill. lefutási idő

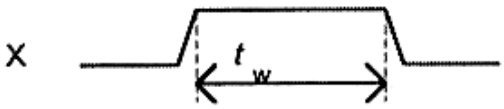
A jelek változási sebessége a valóságos áramkörökben természetesen korlátozott, így a digitális jeleknek is véges meredeksége van. A fel- ill. lefutási időn azt az időt értik, ami a jelváltozás során a 10% elérésétől a 90% eléréséig (ill. fordítva) eltelik.



1.21. ábra. Fel- és lefutási idő

Impulzus szélesség (t_W pulse duration, with)

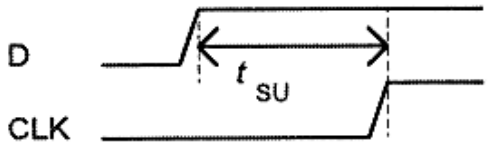
Az impulzus szélessége a jel első és hátsó éle között (pl. órajel bemenetnél).



1.22. ábra. Impulzus szélesség

Előkészítési idő (t_{SU} setup time)

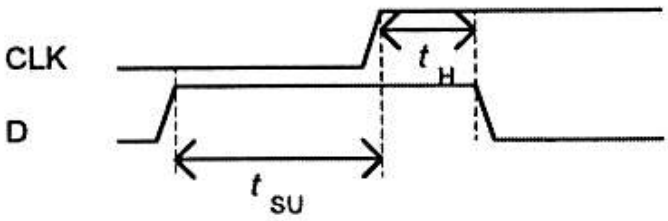
Az egyik jelnek minimálisan a megadott idővel a másik előtt már stabilnak kell lennie. (Pl. flip-flop bemenetének a megadott idővel az órajel előtt már stabilnak kell lenni, lsd 1.23. ábra.) Talán ennek be nem tartásából származnak a legnagyobb bajok. (Lásd a metastabilitást a 2.8.1. fejezetben.) Az előkészítési idő negatív is lehet, ebben az esetben a leghosszabb időt jelenti, amellyel az egyik jel a másik előtt meg kell jelenjen.



1.23. ábra. Előkészítési idő

Tartási idő (t_H hold time)

Az egyik jelet a másik jel megjelenése után még legalább a megadott ideig nem szabad megváltoztatni (pl. órajel után a flip-flop bemenetén az adatot még ennyi ideig kell stabilan tartani, 1.24. ábra). Negatív értéke azt jelenti, hogy ez a leghosszabb idő ameddig a jelet fenn szabad tartani.

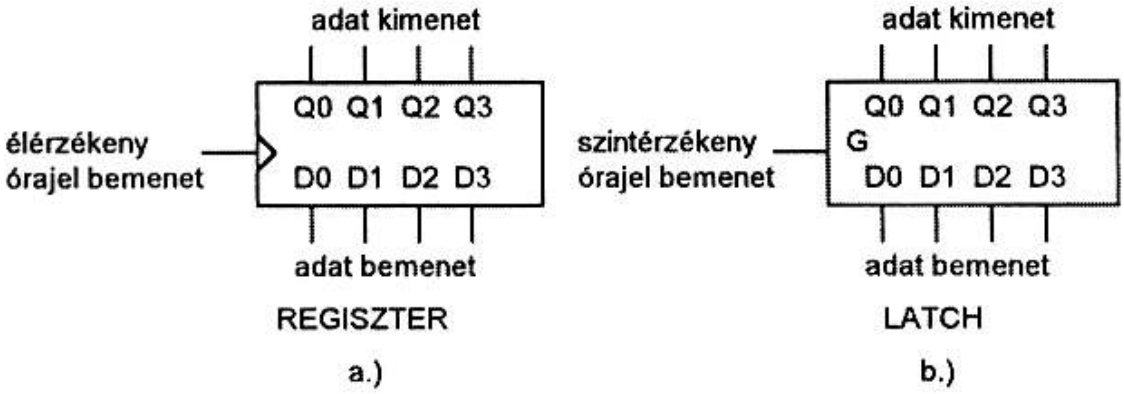


1.24. ábra. Tartási idő

A katalógusban megadott időzítési adatokat szigorúan be kell tartani, különben nem garantált a specifikáció szerinti működés.

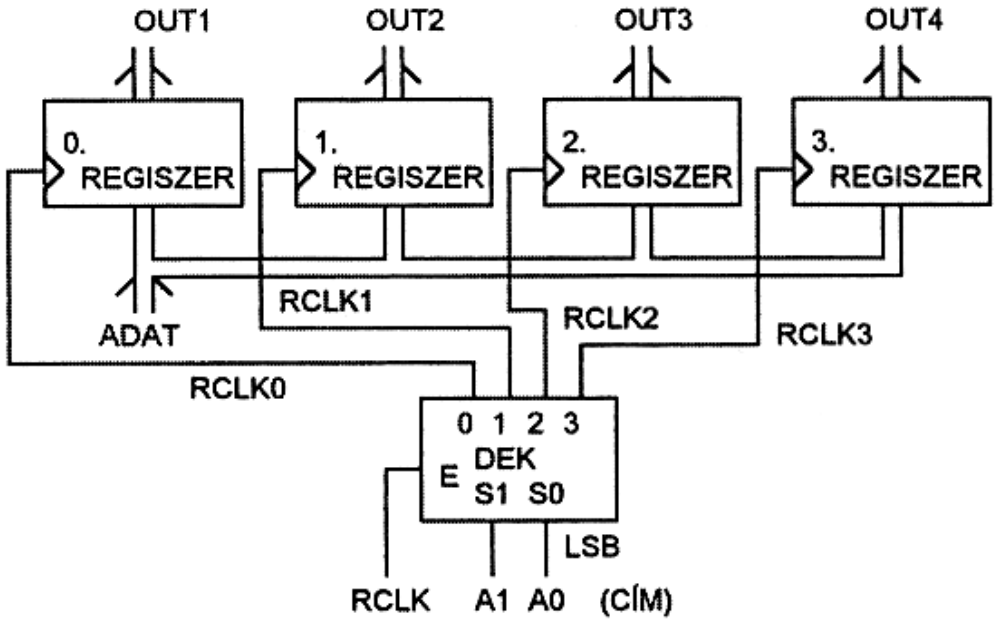
1.2.2. Regiszter, latch

A regiszterek többnyire közösített órajelű és törlő (clear) bemenetű D vagy D-G flip-flopokból (MSI IC-k esetén 4 vagy 8 darab) állnak. D flip-flop esetén regiszternek nevezzük, D-G flip-flop esetén pedig latchnek. Mindkettőt adat tárolásra használjuk, de a latch működése egy lényeges pontban eltér a regiszterétől, ugyanis G=1 esetén *átlátszó* (mintha a bemenet és a kimenet között egy huzal lenne), s ez bizonyos esetekben hasznos. Jelölésüket mutatja az 1.25. ábra.

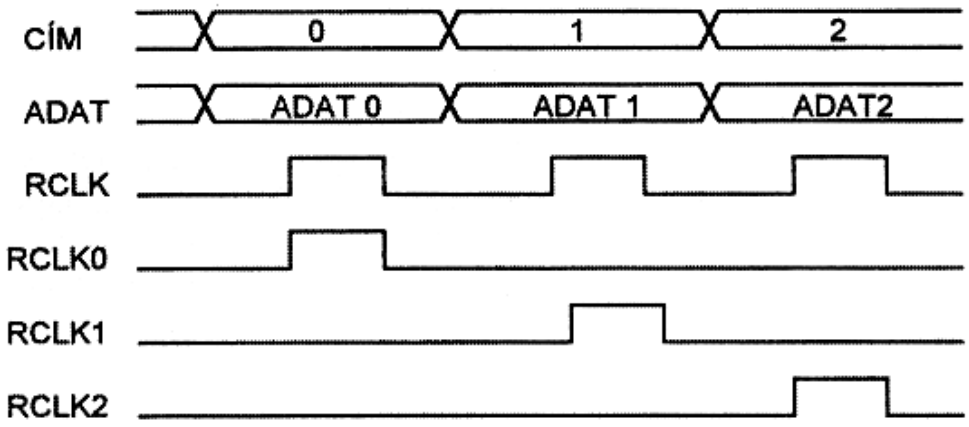


1.25. ábra. Regiszter és latch

Alkalmazási mintapéldaként az 1.26. ábrán egy olyan áramkört mutatunk be, amely egy buszról több regiszterbe képes adatokat írni. Egy regiszter beírásához először a dekóder címét és az adatot kell beállítani, majd a tranziensek lezajlása után egy hazárdmentes impulzust generálunk a dekóder engedélyező bemenetén (1.27. ábra). Ez az impulzus megjelenik a megcímzett regiszter órajel bemenetén és beírja az adatot.



1.26. ábra. Buszon megjelenő adatok beírása regiszterbe



1.27. ábra. Regiszterek beírásának idődiagramja

1.2.3. Számláló

A számlálók ciklikus működésű sorrendi hálózatok, s ha csak a számlálás üzemmódjukat tekintjük, gyűrű alakú állapotgráffal jellemezhetők. A ciklus hosszát nevezzük a számláló **modulusának**. A számláló funkciója az órajelének számolása. A véges állapotszám miatt, az (órajelek száma) modulo (állapotszám) jelenik meg a kimenetén. A számlálók órajel bemenete többnyire **élérzékeny** (fel- vagy lefutó élre vált), de létezik **impulzus érzékeny** számláló is, ilyenkor egy lefutó és egy felfutó él szükséges a számoláshoz.

A kimenet kódolását és a modulust tekintve megkülönböztetünk **bináris** (2^n modulusú, bináris kódolású), **decimális** (10-es modulusú, NBCD kódolású) és egyéb modulusú számlálókat.

Ha a számláló belsejében a flip-flopok órajele közös (a számláló *clock* bemenetéről jövő jel), akkor **szinkron számlálóról** beszélünk. Ha a számláló egy-egy

flip-flopja az órajelet az előző flip-flop kimenetéről kapja, akkor soros órajelű **aszinkron számlálónak** nevezzük. Egyéb nem szinkron esetben egyszerűen aszinkron számlálónak hívjuk. Ennek a jellemzőnek az alkalmazhatóságnál fontos szerepe van, mint később látni fogjuk.

A számlálók egy részénél vezérelhető a számlálás iránya, ezek a **fel-le számlálók** (up/down, reverzibilis számlálók).

A számlálóknak lehet betöltő (**Load**, Ld) bemenete és adat bemenete, ez lehetővé teszi, hogy a számlálóba betöltsük az adatbemeneten található értéket, vagyis a gráfon tetszőleges állapotba vezérelhetjük. Többnyire lehetőség van a számláló törlésére (**Clear**, Cl, az összes flip-flop nullázása) is. A betöltés és törlés lehet szinkron, ha a vezérlőjel aktivizálását követő órajelre történik a törlés vagy betöltés, vagy aszinkron, ha az órajeltől függetlenül, a vezérlőjel aktív szintjének hatására.

Számlálók kaszkádosítása

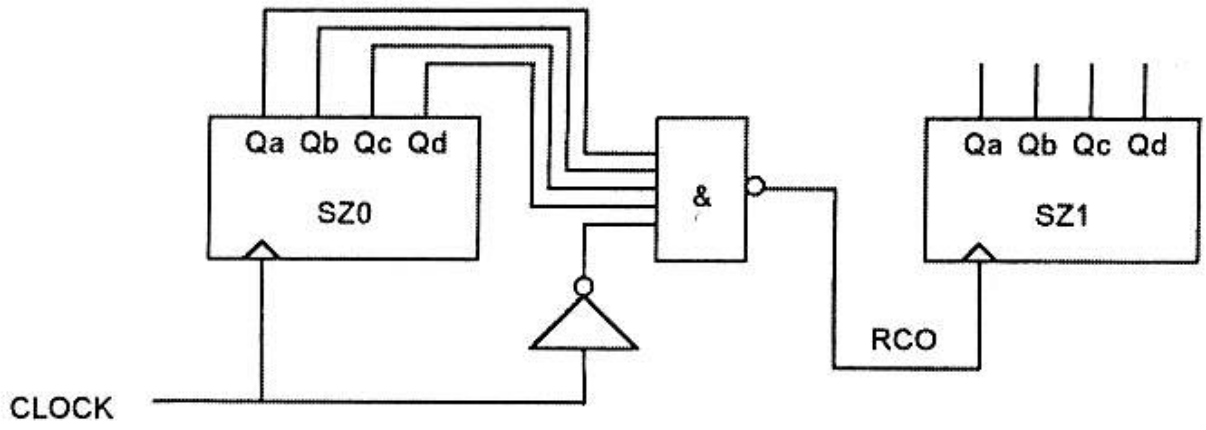
Ha nem elegendő a rendelkezésre álló számláló mérete, akkor azt megfelelő számú egység sorbakapcsolásával tetszőlegesen megnövelhetjük. A számlálók kaszkádosításával tulajdonképpen a modulust növeljük, **az eredő modulus a kaszkádosított számlálók modulusainak szorzata** lesz.

Egy következő egységnek akkor kell lépni, ha az összes előző végállapotban van (felfele számlálónál a legnagyobb, lefele számlálónál a 0). Az 1.2. táblázat mutatja, hogy mikor vált 4 darab 4 bites bináris felfele számláló utolsó egysége.

3.	2.	1.	0.
0000	1111	1111	1111
0001	0000	0000	0000

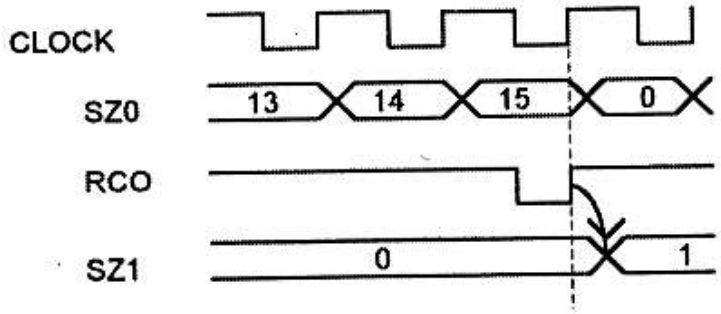
1.2. táblázat

A számlálókat aszinkron, vagy szinkron módon lehet kaszkádosítani. **Aszinkron kaszkádosítás**nak nevezzük, ha egy következő egység az órajelét egy előző egység kimenetétől kapja. Ennek órajel kapuzásos módját mutatja az 1.28. ábra.



1.28. ábra. Aszinkron számlálók kaszkádosítása órajel kapuzással

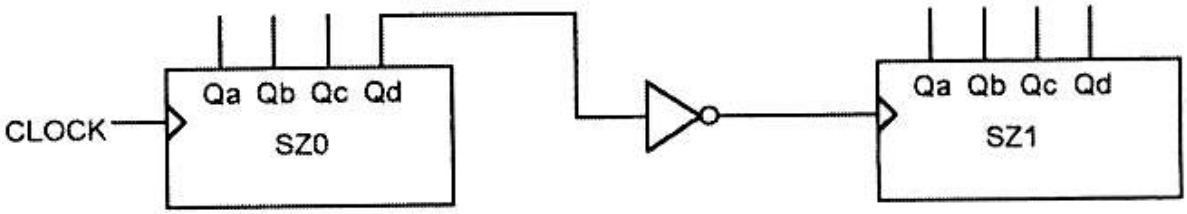
Amint az 1.29. ábra idődiagramja mutatja, a kombinációs hálózat a számláló végállapota esetén (a példában 16-os számláló) tovább engedi az órajel második felét.



1.29. ábra. Órajel kapuzásos kaszkádosítás idődiagramja

Ezt a kombinációs hálózatot eleve beépítik némely számlálóba, és **Ripple Clock**-nak nevezik (RCO). A számláló végállapotát kikódoló kapu kimenete az órajel hozzávétele nélkül nem felelne meg a célnak, mert hazárdos (funkcionális hazárd) hiszen a számláló egymást követő állapotainak kódja több Hamming távolságú!

Ha a számláló kódolása olyan, hogy a legnagyobb helyiértékű kimenete csak a végállapotban vált, (pl. bináris, vagy decimális) akkor felfele számláló esetén, a következő egység az előző egység legnagyobb helyiértékű kimenetének lefutó élére válthat. Lefele számláló esetén a felfutó élére. Két felfutó él érzékeny órajel bemenettel rendelkező előbbi tulajdonságú felfele számláló kaszkádosítása az 1.30. ábrán látható.

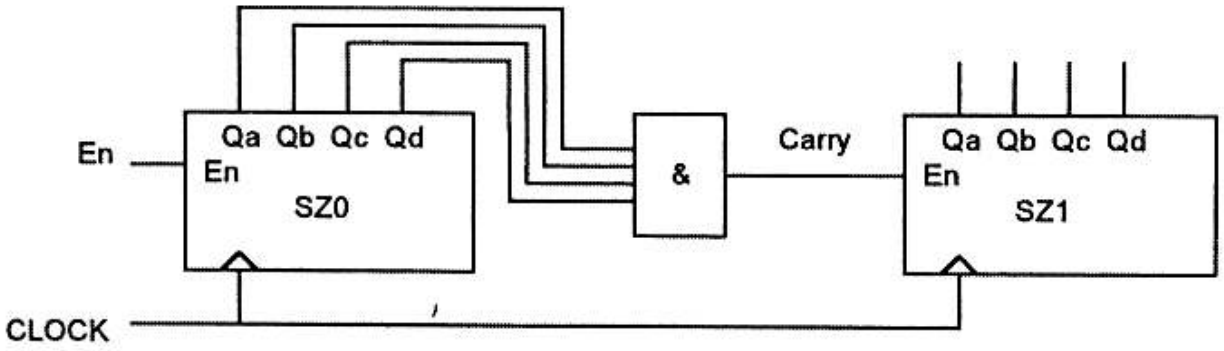


1.30. ábra. Aszinkron kaszkádosítás, az MSB-t használva órajelként

Aszinkron módon minden számláló kaszkádosítható. Azonban az aszinkron kaszkádosítás hátránya, hogy az órajel soros terjedése miatt az első és utolsó egység között jelentős késleltetés lehet (a számlálók időben eltolva váltanak). A késés a számlálók frekvenciaosztóként történő alkalmazása esetén nem okoz gondot, de bizonyos alkalmazásoknál problémát jelenthet. Ezt a késleltetést szinkron számlálók **szinkron kaszkádosítás**ával lehet kivédeni.

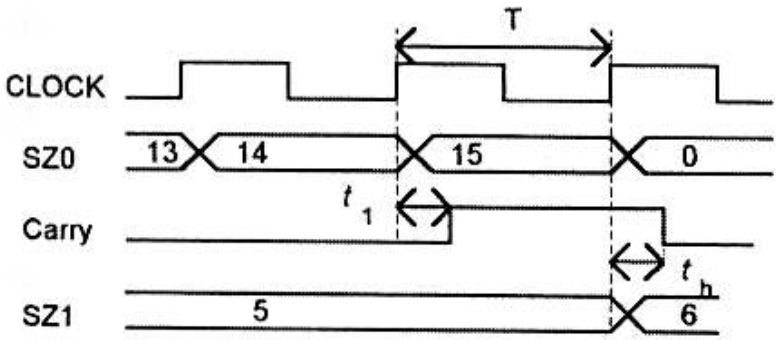
A szinkron számlálók rendelkeznek **engedélyező bemenet**tel (**En**), ami lehetővé teszi a szinkron kaszkádosítást. Egy szinkron számláló csak akkor lép, ha engedélyező bemenete aktív, egyébként marad az aktuális állapotban. Szinkron kaszkádosítás esetén az összes kaszkádosítandó számláló közös órajelet kap. Így egy következő egységet akkor kell engedélyezni, amikor az összes előző végállapotban van. A számláló végállapotának kikódoltját nevezik **Carry**nek. 16-os számláló esetén $Carry = Q_a Q_b Q_c Q_d$. Kétirányú számlálónál mindkét végértéket kikódolják:
$Max / Min = Q_a Q_b Q_c Q_d Up + \overline{Q}_a \overline{Q}_b \overline{Q}_c \overline{Q}_d Down$. (Up és Down a számlálási irányt jelző belső logikai jelek.

A szinkron kaszkádosítást mutatja az 1.31. ábra, 2 számláló esetére.



1.31. ábra. Számlálók szinkron kaszkádosítása carryvel

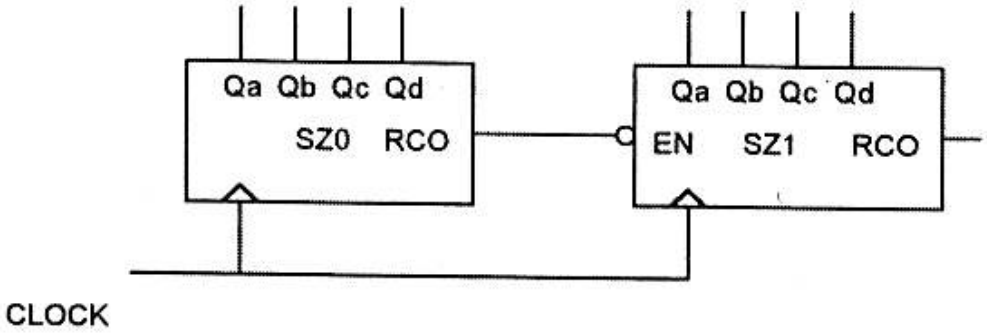
Amint az 1.32. ábra idődiagramja mutatja, a következő egységet az előző carry kimenete engedélyezi.



1.32. ábra. Szinkron kaszkádosítás idődiagramja

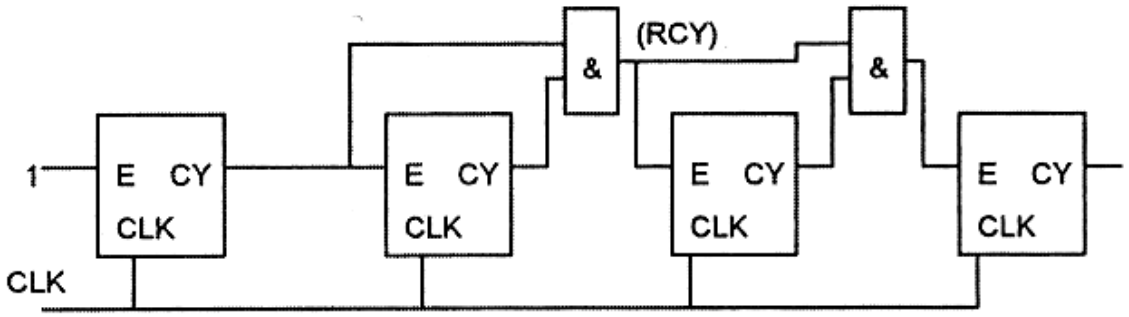
Az idődiagramból látható, hogy a carry jel a flip-flopok és az ÉS kapu késleltetése miatt késik az órajelhez képest. Így $T - t_1$ előkészítési idő és t_h tartási idő adódik az engedélyező bemenethez.

Az alapvetően aszinkron kaszkádosításra szánt RCO kimenetet is fel lehet használni szinkron kaszkádosításra (1.33. ábra), de katalógus alapján célszerű ellenőrizni az En előkészítési idejének teljesülését.



1.33. ábra Szinkron kaszkádosítás RCO-val

Több számláló kaszkádosításánál az összes előző egység carry kimenetét ÉS kapcsolatba hozva állíthatjuk elő a következő tag engedélyező jelét. (Egyes számlálóknak eleve az engedélyező jelet is magába foglaló **Rilpple Carry** (RCY) kimenete van, így a kaszkádosításhoz nincs szükség külön kapukra.) Ennek egy megoldását mutatja az 1.34. ábra:

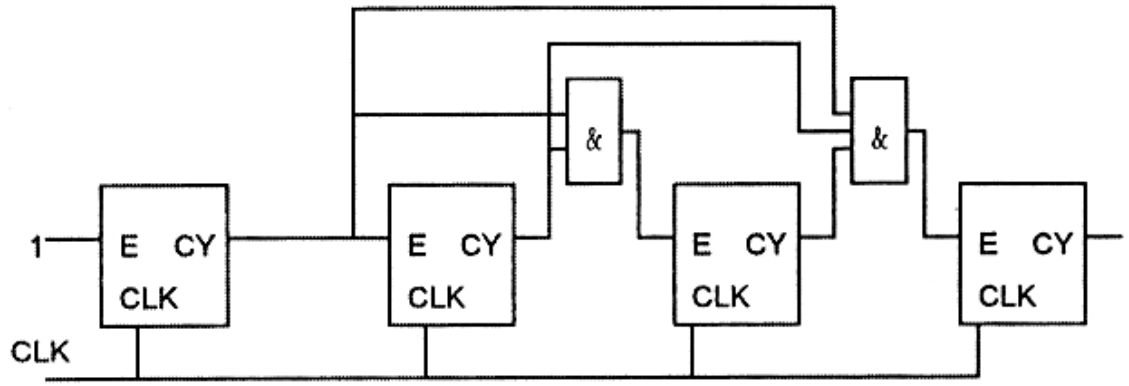


1.34. ábra. Számlálók szinkron kaszkádosítása, az engedélyezés soros előállításával

E megoldás hátránya, hogy minél távolabb van egy számláló a legelsőtől, annál több kapun keresztül, s ezáltal annál nagyobb késleltetéssel kapja meg az engedélyező jelet. Nagyobb órajel frekvenciáknál és hosszú számlánc esetén a késleltetés akkorára nőhet, hogy egy távoli egység 1 órajellel később veszi észre az engedélyezést, mint kellene.

Ilyenkor a következő megoldások közül választhatunk:

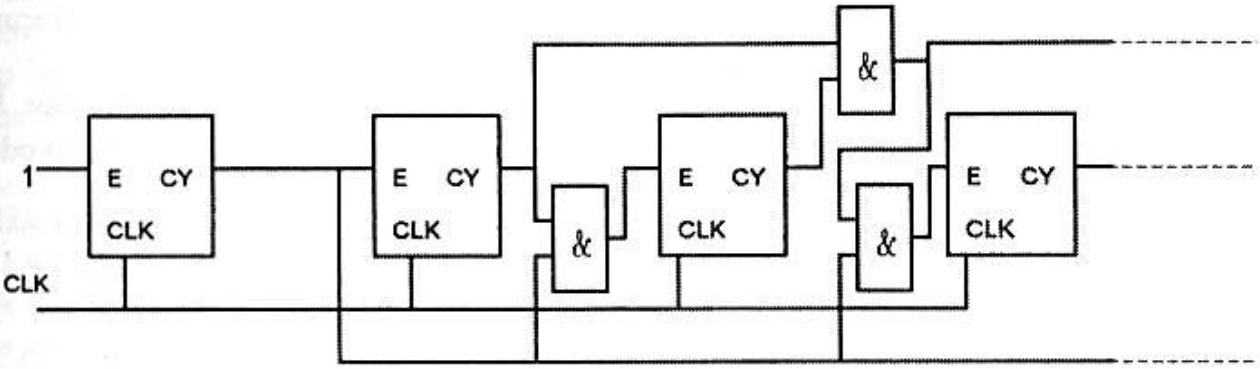
Az alábbi kapcsolásban (1.35. ábra) minden egységhez 1 kapu késleltetéssel jut el az előző egységek carry kimenete. Sajnos minél távolabb van egy egység, annál nagyobb bemenetszámú kapura van szükség.



1.35. ábra. Számlálók kaszkádosítása az engedélyezés párhuzamos előállításával

A következő megoldásban (1.36. ábra) minden újabb egységnél csak maximum 2 db 2 bemenetű kapura van szükség. Az egyik kapu azt jelzi, hogy az összes előző egység a legelső kivételével végállapotban van. Ennek az információnak egy adott egységhez való terjedésére az első számláló egy teljes ciklusideje rendelkezésre áll, így nagyobb számú egység kaszkádosítható probléma nélkül. Az engedélyezéshez szükséges másik feltétel, vagyis, hogy az első számláló is végállapotban van, viszont közvetlenül előre van csatolva, így minimális késleltetéssel jut el egy tetszőleges egységhez. Az ilyen jellegű

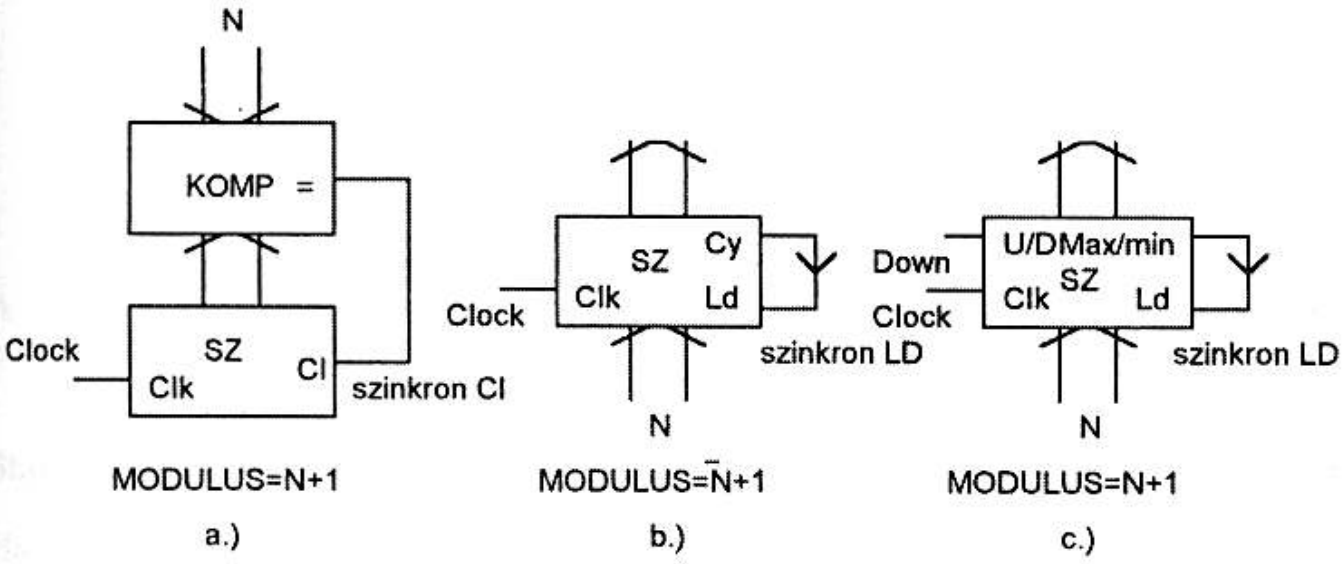
kaszkádosításhoz szükséges kapuk egynémely számlálóba be vannak építve (74LS160, 74LS163).



1.36. ábra. Gyors szinkron kaszkádosítás előrecsatolással

Számlálók modulusának csökkentése

Sokszor előfordul, hogy a rendelkezésre álló számláló vagy a kaszkádosításból kapott számláló modulusánál kisebbre van szükségünk. A modulus csökkentésének alapvető módjait mutatja az 1.37. ábra.



1.37. ábra. Modulus csökkentési módszerek

Mindhárom megoldásban *szinkron* Clear ill. Load bemenetekre érvényes a megadott modulus! Aszinkron Clear ill. Load esetén problémák lehetnek, mert a komparátor kimenete és a számlálók carry ill. Max/Min kimenete *hazárdos* lehet, ami a kívántnál korábbi törlést vagy betöltést eredményezhet. Ennek ellenére vannak esetek, amikor biztosítható a hazárdmentesség. *Ekkor a modulus az 1.37. ábrán megadottaknál 1-gyel kisebb.*

Az 1.37a ábra szerinti esetben a számláló 0-tól N-ig számol ciklikusan. Sokszor a természetes kódsorrend igénye miatt ezt a megoldást kell használni. Ha nem szükséges a

programozhatóság, akkor a komparátor helyett ÉS kaput és invertereket használhatunk a konstans N kikódolására. Többnyire nem szükséges a számláló összes kimenetét figyelni a kötött kódsorrend miatt. (Pl. ha 16-os felfele számlálóból 10-es számlálót akarunk készíteni, elég a két szélső bit figyelése, mert az 1xx1 kód itt fordul elő először.)

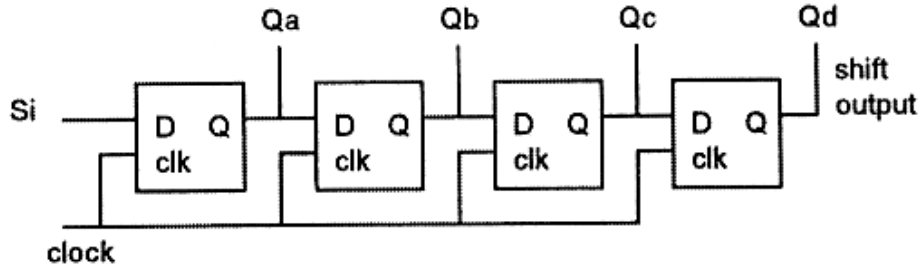
Az 1.37b ábra szerinti esetben a számláló N-től számol $2^n - 1$-ig, ciklikusan. Ez $2^n - 1 - N + 1$ állapot. Mivel $2^n - 1 - i = \bar{i}$ vagyis bináris számrendszerben gondolkodva az i bitenkénti negáltja (1-es komplemense), ezért a modulus $\overline{N} + 1$.

Az 1.37c ábra szerinti esetben a számláló N-től számol 0-ig, ciklikusan.

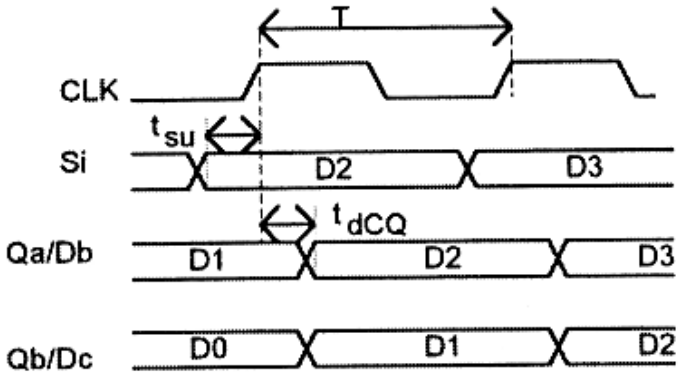
A *b* típust inkább frekvenciaosztás esetén szokás alkalmazni a kódolása miatt, míg az *a* és *c* szélesebb körben alkalmazható, közülük is az utóbbi az olcsóbb.

1.2.4. Shiftregiszter

A shiftregiszter olyan tároló (többnyire 4, 8 bites), amely minden órajelre eltolja (shifteli) a tartalmát 1-el valamelyik irányba. A legegyszerűbb shiftregiszter sorba kötött D flip-flopokból áll (1.38. ábra).



1.38. ábra. Egyszerű shiftregiszter belső felépítése

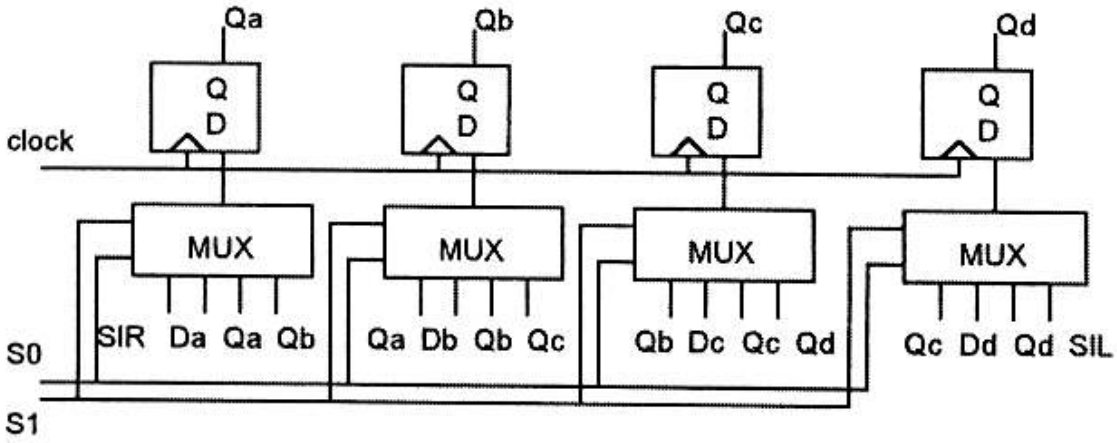


1.39. ábra. Egyszerű shiftregiszter idődiagramja

A működést idődiagramon szemlélteti az 1.39. ábra. Az összes regiszter közös órajelet kap. A bemeneti flip-flopnál a felhasználónak kell biztosítani az adat-előkészítési időt (adat stabil az órajel előtt legalább t_{SU} idővel). A többi flip-flopnál ez automatikusan adódik, mivel $T - t_{dCQ}$ idővel az órajel előtt már stabil az adat (az előző flip-flop kimenete beállt). A flip-flop által igényelt tartási idő (adat stabil az órajel után legalább t_H

ideig) pedig kevesebb, mint a saját órajelhez képest mért késleltetési ideje, t_{dCQ}. Az itt leírtakhoz hasonló történik egy tetszőleges szinkron sorrendi hálózat (pl. szinkron számláló) működése során is, mert az összes flip-flop közös órajelet kap, s egy-egy flip-flop bementére egy kombinációs hálózaton keresztül visszacsatolódik a többi flip-flop kimenetének ill. saját kimenetének változása.

A shiftregiszter bonyolultabb változatainál változtatható a shiftelés iránya, van tartási üzemmód, ill. betöltési lehetőség. Egy ilyen shiftregiszter belső felépítését mutatja az 1.40. ábra. Az üzemmódot a multiplexerek címbementével lehet kiválasztani. Ha az *i*-edik flip-flop bemenetére a tőle balra levő kimenetét választjuk, akkor jobbra shiftel, ha a tőle jobbra levőt, akkor balra shiftel, ha a saját kimenetét, akkor megtartja állapotát, ha a hozzá tartozó adat bemenetet, akkor betölt.



1.40. ábra. Többfunkciós shiftregiszter belső felépítése

A shiftregiszternek sok felhasználási módja van. Ezek közül a legfontosabbakat ismertetjük a következőkben.

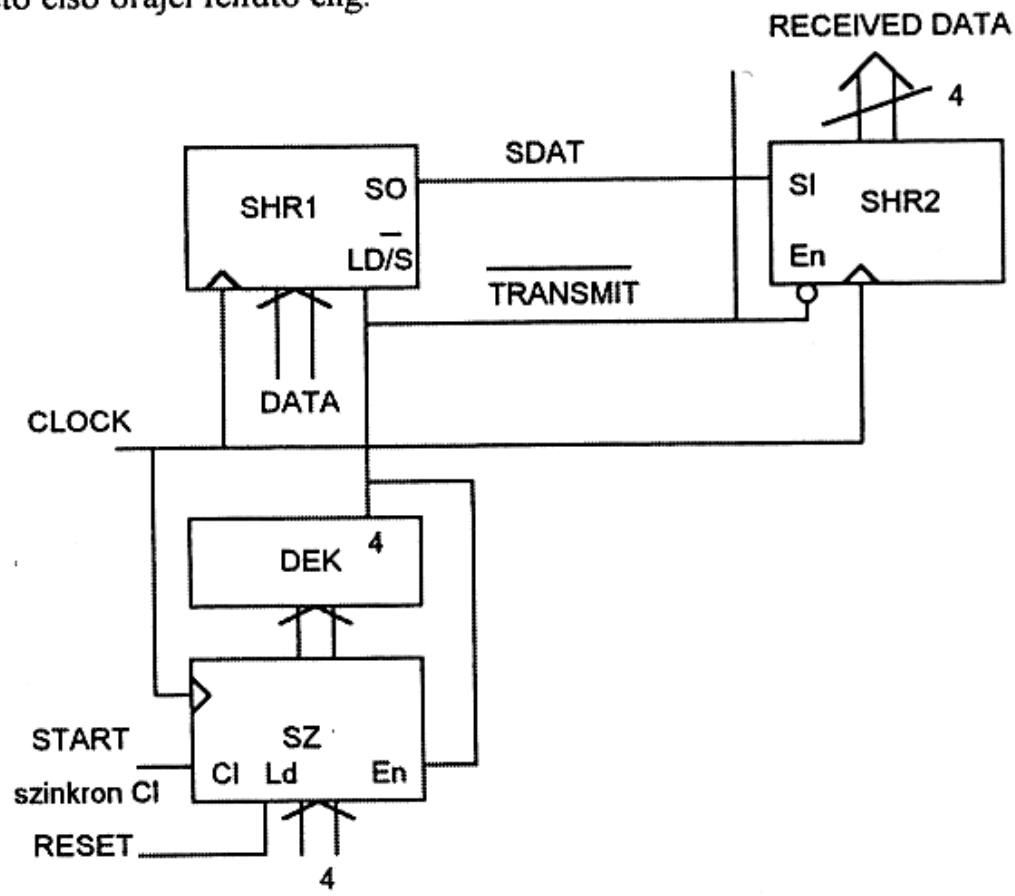
Shiftregiszter mint párhuzamos-soros, ill. soros-párhuzamos átalakító

Ha sok bites adat továbbítását kevés vezeték felhasználásával akarjuk megoldani, akkor az egyes biteket időben egymás után, sorosan kell továbbítani. Ezt a shiftregiszterbe betöltött adat kishiftelésével oldhatjuk meg. Természetesen a vételi oldalnak tudnia kell, hogy mikor értelmezze a biteket. Erre egy megoldás, ha az adatbiteken túl az órajelet és egy az adatátvitel ideje alatt aktív Transmit jelet is megkap a vevő, amely szintén egy shiftregiszter.

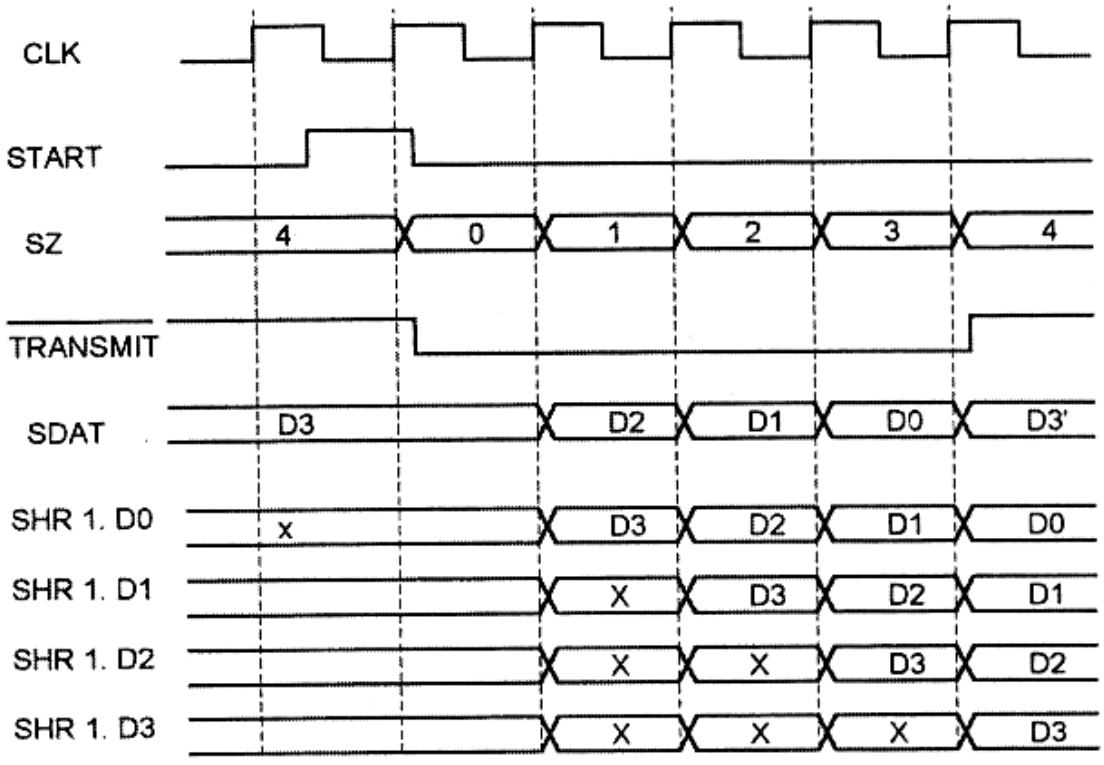
Példa:

Az 1.41. ábra egy párhuzamos-soros átalakító funkcionális blokkvázlatát mutatja, 4 bites adatok esetén. A működés az 1.42. ábra idődiagramján követhető végig. A RESET jel hatására az SZ 3 bites bináris számláló 4-et tölt be. Ennek hatására a kimenetére kapcsolt dekóder 4-es kimenete aktivizálódik, letiltja a számlálót és betöltés üzemmódba kapcsolja az adó shiftregiszterét (SHR1). Az átalakítást a START jel indítja, mely kiadása előtt az

átalakítandó 4 bites adatot a DATA bemenetre kell adni, és stabilan kell tartani a START jelet követő első órajel felfutó élig.



1.41. ábra. Egyszerű szinkron soros adó-vevő



1.42. ábra. Soros párhuzamos átalakító idődiagramja

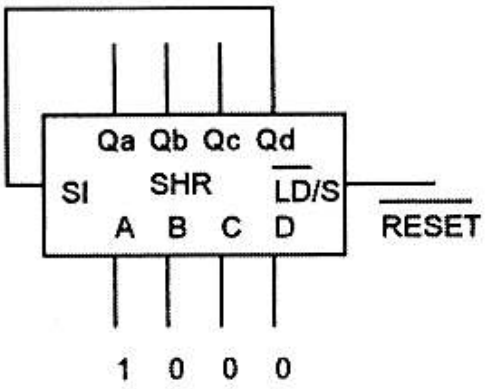
A START jel aktív szintje alatti első felfutó órajel élre a számláló törlődik, s aktivizálódik a TRANSMIT jel. (A START jelet a szinkron törlés miatt legalább egy órajel periódusnyi ideig aktivizálni kell, hogy biztosan törlődjön a számláló.) A TRANSMIT jel aktivizálása engedélyezi a számlálót, betöltés módból shiftelésbe állítja az adó shiftregiszterét, továbbá engedélyezi a vevő shiftregiszterét (SHR 2). Az ezt követő 4 órajel alatt az adó shiftregisztere egymás után az SDAT vonalra adja a D3-D0 biteket az órajellel szinkronban, melyek beshiftelődnek a vevő shiftregiszterébe. Az D0 bit átvitele után a számláló újra a 4-es állapotba jut, letiltódik a számláló, betöltés módba kapcsol SHR1, letiltódik SHR2, s így a hálózat alapállapotba kerül.

Shiftregiszter mint számláló

Shiftregiszterből sokféle módon készíthetünk számlálót, itt a 3 legegyszerűbbet ismertetjük (gyűrűs számláló, Johnson számláló, véletlenszám generátor). Mindhárom megoldás közös vonása, hogy a shittregiszter kimeneteit egy kombinációs hálózaton keresztül visszavezetjük a soros bemenetre.

Gyűrűs számláló

A gyűrűs számlálónál a visszacsatoló kombinációs hálózat egy darab drót. Így a kezdetben betöltött értéket rotálja ciklikusan. N bites shiftregiszter esetén a maximális modulus N. A modulus a kezdetben betöltött értéktől is függ. Többnyire az egyetlen 1-est tartalmazó kezdeti értéket alkalmazzuk. Az 1.43. ábra egy 4 bites gyűrűs számlálót mutat.



1.43. ábra. Gyűrűs számláló

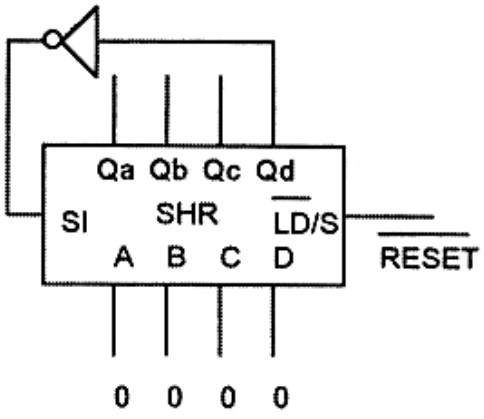
A számláló kódolása Qa, Qb, Qc, Qd:

1000, 0100, 0010, 0001

Hasonló kimeneti sorozatot egy számlálóval címzett dekóderrel is előállíthatunk, azonban ennek hátránya, hogy a dekóder bemenetén több Hamming távolságúak (több bitben eltérőek) az egymást követő címek, így ezeknél az átmeneteknél funkcionális hazárd léphet fel (tranziensnyi időre olyan kimenet aktivizálódhat, amelynek nem kellene).

Johnson (Möbius) számláló

A Johnson számlálónál a visszacsatoló hálózat egyetlen inverterből áll. Így 0 kezdetiérték esetén a számláló először feltölti magát egyesekkel, majd nullákkal. Ebből adódik, hogy a modulusa 2N. Négy bites Johnson számlálót mutat az 1.44. ábra.



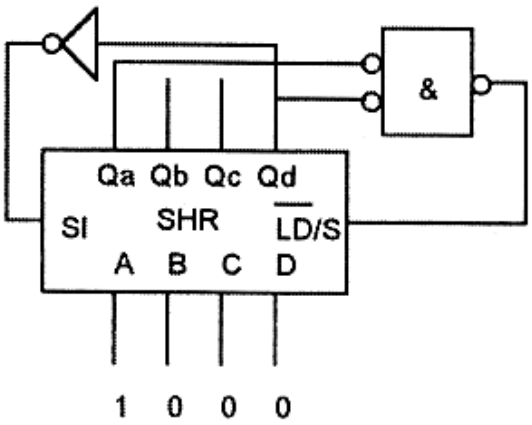
1.44. ábra. Johnson számláló

A számláló kódolása Qa, Qb, Qc, Qd: 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001

A számláló egy tetszőleges állapotát egy kétbemenetű ÉS kapu és inverterek segítségével dekódolni lehet. A kapuval a 0-1 átmenetet, vagy ha ilyen nincs, a két szélső bitet kell figyelni. (Pl. 0000 $\overline{Qa}\,\overline{Qd}$, 1000 $Qa\,\overline{Qb}$) Előnye, hogy az ily módon előállított kimenet nem hazárdos, mivel az egymást követő kódok szomszédosak.

Önkorrigáló Johnson számláló

Létezik a Johnson számlálónak egy olyan változata, amely tetszőleges kezdőállapotból beletalál a normál ciklusba. Az áramkör azt használja ki, hogy tetszőleges állapotból a Johnson számláló előbb-utóbb előállít egy 0xx0 állapotot. Így egy kapuval ezt figyelve, betölthető a normál üzemmód valamely ettől különböző állapota, s így nincs szükség egy külön RESET jelre a bekapcsolás után. 4 bites önkorrigáló Johnson számlálót mutat az 1.45. ábra.



1.45. ábra. Önkorrigáló Johnson számláló

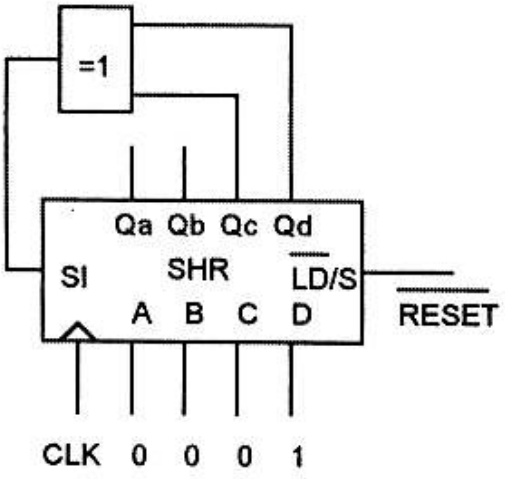
Véletlenszám generátor

Ha a shiftregiszter visszacsatoló hálózata XOR kapukból áll, akkor megfelelő kimenetekről történő visszacsatolás esetén az így kapott számláló modulusa $2^n - 1$.

A modulus mindenképpen kisebb 1-el, mint az n biten ábrázolható bináris számok számossága, mert csupa 0-ból álló kezdeti érték esetén a következő állapot is ugyanez maradna. Azt belátni, hogy a megadott modulus valóban elérhető, már bonyolultabb, ezért ettől itt eltekintünk. Az ily módon előálló számláló fontos tulajdonsága, hogy az egymást követő kódok eléggé véletlenszerűek. Egy 4 bites shiftregiszter esetén maximális ciklushosszt eredményez az 1.46. ábrán látható visszacsatolás.

A számláló kódolása Qa, Qb, Qc, Qd:

0001, 1000, 0100, 0010, 1001, 1100, 0110, 1011, 0101, 1010, 1101, 1110, 1111, 0111, 0011



1.46. ábra. Véletlenszám generátor

Néhány esetre megadjuk a szükséges visszacsatolások helyét az állapotszám függvényében:

shiftregiszter mérete	állapotszám	visszacsatolandó bitek sorszáma
3	7	3, 2
4	15	4, 3 (Ez látható az 1.43. ábrán.)
5	31	5, 3
6	63	6, 5
7	127	7, 6
8	255	8, 6, 5, 4.

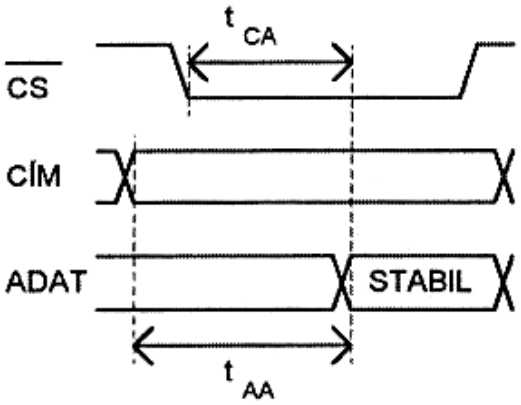
1.3. Memória elemek

A memóriák sok adat egyidejű tárolására és gyors kiolvasására alkalmas elemek.
A memóriákon előforduló tipikus jelek:

Adat és cím:
- cím bemenetek (An-A0), az adat címének kijelölésére,
- adat ki/bemenetek (Dm-D0), az adatok beírására, kiolvasására.

Vezérlőjelek:
- CS (chip select) kiválasztó bemenet, a memória engedélyezésére,
- RD (read) vagy OE (output enable) olvasás engedélyező bemenet,
- WR írás engedélyező bemenet.

Attól függően, hogy a memória csak olvasható, vagy írható és olvasható is, megkülönböztetünk **ROM**-okat (Read Only Memory) és **RAM**-okat (Random Access Memory). A memória elemek egyik legfontosabb jellemzője a tárolható adatok mennyisége. Ezt (szószám) x (szószélességben), vagy byte-ban (1 byte=8 bit), kilobyte-ban (1 kbyte=1024 byte), megabyte-ban (1 Mbyte=1024 kbyte) szokás megadni. (Pl. 256x4 bites, 32 byte-os, 256 kbyte-os.) A memóriák másik fontos jellemzője a hozzáférési idő (access time). Egy memóriához különféle hozzáférési időket szoktak megadni. CS-hez képest t_{CA}, címhez képest t_{AA} idővel stabilizálódik az adat. Azonban ha külön nem jelölik meg, akkor a címhez képesti hozzáférési időre kell gondolni. Az 1.47. ábrán látható az értelmezésük. Mindkét hozzáférési idő esetében feltételezzük, hogy az OE bemenet folyamatosan engedélyezve van.

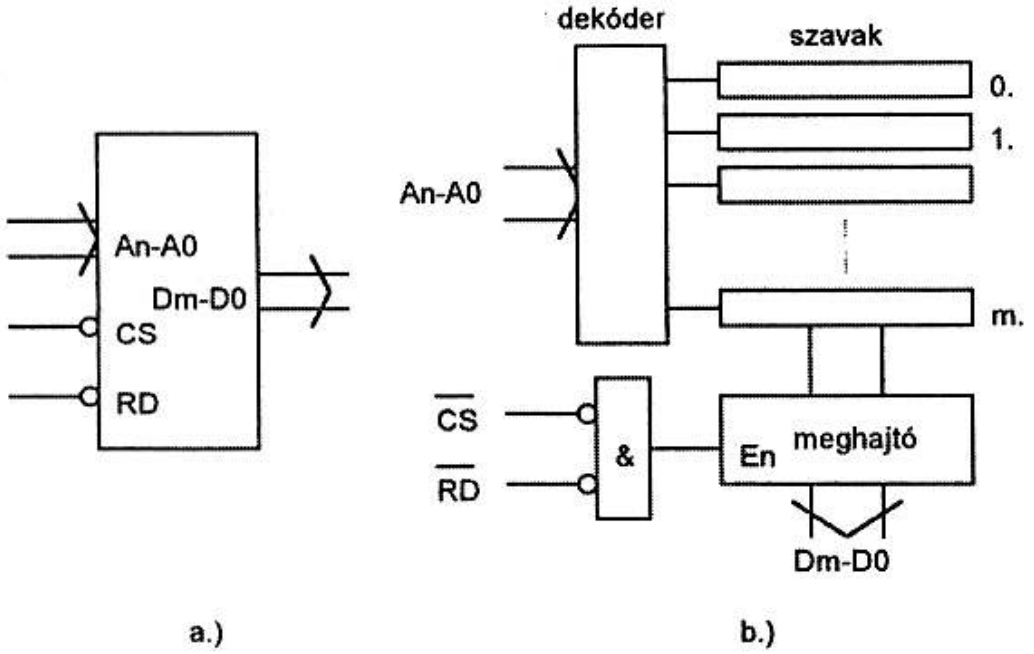


1.47. ábra. Hozzáférési idők

1.3.1. ROM

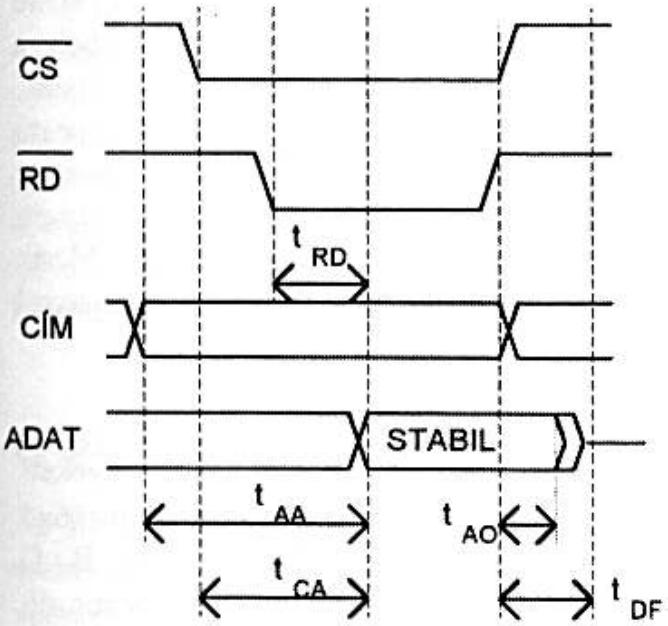
A ROM-ok ún. csak olvasható memóriák. Leggyakrabban mikroprocesszoros rendszerekben nem megváltoztatható programok (pl. PC BIOS) és konstans adatok tárolására, ill. univerzális kombinációs hálózatként alkalmazzák.

A ROM jelölését mutatja az 1.48a ábra, leegyszerűsített blokksémáját pedig az 1.48b ábra. Itt a beíráshoz (beégetéshez) szükséges blokkokat nem tüntettük fel.



1.48. ábra. ROM egyszerűsített blokkvázlata

Az An-A0 címbemenetekkel címezhető meg a ROM egy-egy szava. A megcímzett szó a kimeneten akkor jelenik meg, ha a CS bemenet, és az RD bemenet egyszerre aktív (1.49. ábra).



1.49. ábra. Memória olvasás idődiagramja

A ROM-ok kimenete többnyire három állapotú, de léteznek open collectorosak is (némely PROM). Az utóbbiaknál nem szabad megfeledkezni arról, hogy a működésükhöz felhúzó ellenállásokra is szükség van.

Az 1.49. ábra idődiagramján látható a ROM-ok, EPROM-ok katalógusokban előforduló legfontosabb időadatai.

t_{CA}, t_{AA}: a már ismertetett hozzáférési idők,
t_{RD}: az olvasás engedélyezés jelhez képesti adat késleltetés,
t_{AO}: a cím megváltozásához képest az adat megváltozása,
t_{DF}: a olvasás engedélyezés megszüntetéséhez képest a kimenet 3. állapotba megy.

A memóriák alkalmazásakor figyelembe kell venni a katalógusban megadott időadatokat!

A ROM-okba speciális programozó ("égető") készülékkel vihetők be az adatok, melyek kikapcsolás után is megmaradnak. Egyes ROM-okba a gyártó viszi be az adatokat a chip készítésének utolsó lépéseként, ún. maszkprogramozással.

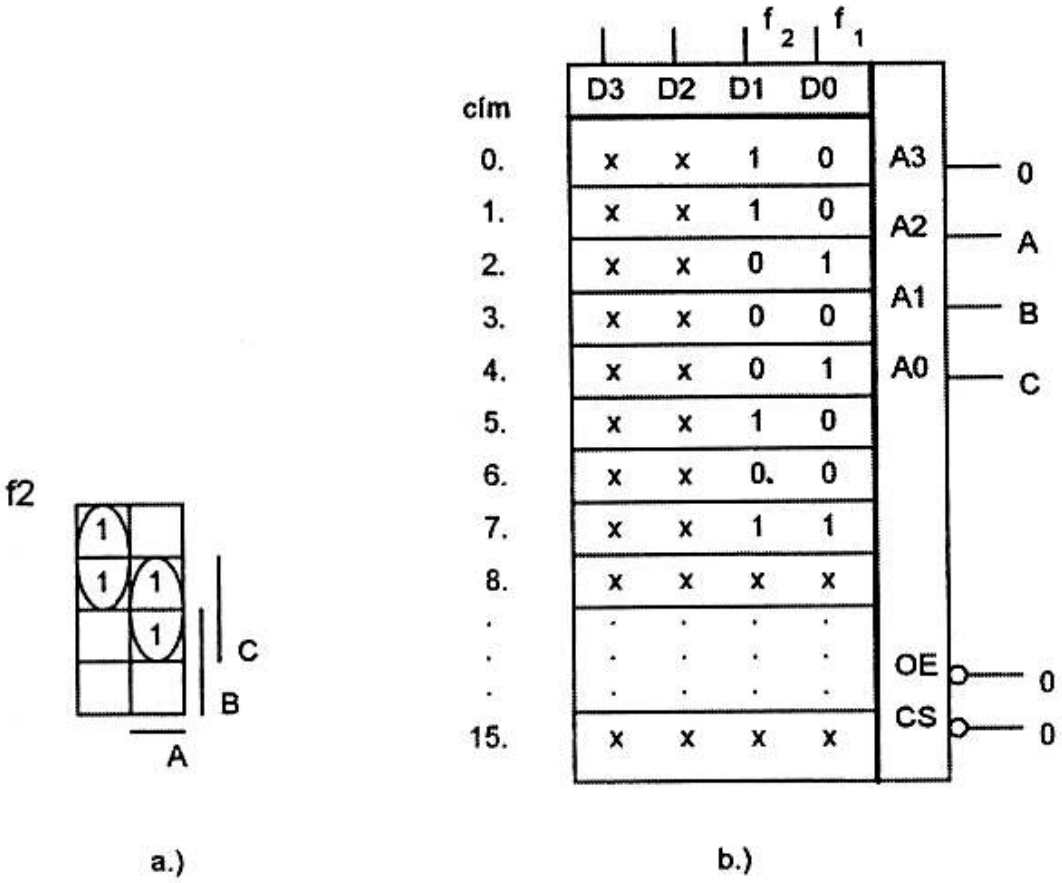
Az első PROM-ok bipoláris tranzisztorokból készültek. Programozáskor a bipoláris PROM-ban biztosítékokat (fuse) égetnek ki (rövidzárakat szüntetnek meg), ezért irreverzibilis az eljárás. Az égetés bitenként történik, a viszonylag nagy áram okozta hőfejlődés korlátozása végett. Ezek az áramkörök gyorsak, (néhány nsec-os késleltetésűek is léteznek) azonban szintén a technológiából adódóan nagy az egységnyi chipfelületre jutó disszipációjuk (hőtermelésük), ezért csak viszonylag kis tárolókapacitásúakat (~2 kbyte) gyártanak. Ma már inkább MOS és CMOS változatok léteznek. Ezeknél programozáskor inkább biztosítékokat aktivizálnak (anti fuse), ami tkp. a már meglévő, de még nem vezető antifuse vezetővé tételét jelenti.

ROM, mint univerzális kombinációs hálózat

A ROM-ot, mint univerzális (programozható) kombinációs hálózatot is gyakran alkalmazzák (pl. mikroprocesszoros rendszerek címdekódere). Ebben az esetben a kombinációs hálózat bemenetei a ROM címbemenetei, kimenetei pedig az adatvonalak. A ROM-ba a kombinációs hálózat igazságtábláját kell "beégetni". Így egy n címbemenetű ROM-mal tetszőleges n bemenetű kombinációs hálózat megvalósítható. Mivel a bemenetek számával az igényelt ROM mérete exponenciálisan nő, ezért 8-10 bemenetű kombinációs hálózatnál nagyobbat nem célszerű ezzel a módszerrel megvalósítani. Másik hátránya, hogy az így előállított kombinációs hálózat hazárdos, és a szokásos módszerrel - minthogy nem kapuk valósítják meg a termeket - nem hazárdmentesíthető.

Példa:

Valósítsuk meg ROM-mal az $f_1 = ABC + \overline{A}B\overline{C} + A\overline{B}\overline{C}$ és $f_2 = \overline{A}\overline{B} + AC$ függvényeket! A megvalósításhoz egy 8x2 bites ROM-ra van szükség. Ilyen nem kapható, ezért 16x4 bitesből készítjük el. A ROM A2, A1, A0 címbemeneteire kapcsoljuk az A, B, C változókat, a D0 kimenethez rendeljük az f_1, D1-hez pedig az f_2 függvényt. Először elő kell állítani mindkét függvény igazságtábláját. Az f_1 igazságtáblája rögtön adódik, mert az diszjunktív normál alakban adott. Az f_2 igazságtábláját legegyszerűbben Karnaugh-táblában ábrázolva olvashatjuk ki (1.50a ábra).



1.50. ábra. Logikai függvény megvalósítása ROM-mal

$$f_2 = \overline{A}\,\overline{B} + AC = \overline{A}\,\overline{B}\,\overline{C} + ABC + A\overline{B}C + \overline{A}\,\overline{B}C$$

A függvények ROM-os megvalósítását mutatja az 1.50b ábra. Itt a ROM tartalmát a szemléletesség miatt táblázatos formában ábrázoltuk. Az A3 címbitet fixen 0 szintre kötöttük, mert 3 változós a függvény, így csak a ROM alsó 8 szavát használjuk ki, a többi területre közömbös, hogy milyen érték kerül. A memória szavak D3 és D2 bitjeire szintén közömbös, hogy mit írunk, mert csak 2 függvényt kell megvalósítani. A ROM-ot folyamatosan engedélyeznünk kell, ezért a CS és OE bemeneteket fixen aktív szintre kötöttük.

EPROM

Az EPROM-ból (Erasable PROM) mint a neve is mutatja, az adat kitörölhető, s az eszköz újraprogramozható. Ezekben az áramkörökben MOS tranzisztorok jól elszigetelt vezérlő elektródájára felvitt töltések tárolják az információt. A programozáshoz a TTL áramköröknél megszokott 5V-nál nagyobb (12.5V-21V) feszültségre van szükség, "beégetésüket" külön erre a célra kifejlesztett EPROM programozóval lehet elvégezni. Csak nagyon speciális esetben fordul elő, hogy az EPROM égetésére képes áramkört az EPROM felhasználási helyére is beépítik (pl. pénztárak "fekete doboza"). Jelenleg a gyártók kb. 5 évet garantálnak a felvitt információ megőrzésére, de a gyakorlatban ez hosszabb idő. Az eszköz típustól függően kb. 1000-10000-szer égethető újra. A törlés a chipen lévő kvarcüveg ablakocskán keresztül történő UV besugárzással lehetséges (min.

5-10 perc). Az UV sugárzás hatására az előzőleg felvitt töltések eltávoznak. Az EPROM-ok a RAM-okhoz képest lassú eszközök, hozzáférési idejük 70-300 nsec nagyságrendű. A könyv írásakor Megabit-es nagyságrendben van a legnagyobb EPROM-ok kapacitása.

EEPROM

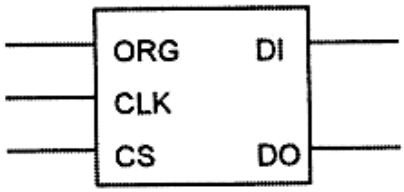
Ez az eszköz az EPROM-hoz hasonló, de elektromosan törölhető (a megfelelő lábakra adott vezérléssel), s ez az eljárás lényegesen gyorsabb, mint az UV-s törlés, de azért még mindig elég hosszú (néhány msec). Sajnos a törlési ciklusok száma korlátozott (1000-10000). Az újabb típusokhoz nem kell külön programozó készülék, mert külön törlő és a RAM-okhoz hasonló beírási funkciójuk van, így ezek a funkciók a mikroprocesszoros rendszerekbe illesztett EEPROM-ok normál működésének részei. A beírási ill. törlési folyamatot egy az IC-be beépített vezérlő végzi. Mivel törlés ill. a beírás lassú, és újabb adatot csak az előző beégetése után lehet beírni, ezért a beírási vagy törlési folyamat végét egy READY jellel jelzi az áramkör, ezt kell figyelnie a beírást végző egységnek.

A legnagyobb méretű EEPROM-oknál már hibajavítást is alkalmaznak. Pl. az AT28C256-ban egy adatbyte beírásakor még 4 speciálisan előállított bitet tárolnak, ami lehetővé teszi a hibajavítást.

A külön programozót nem igénylő EEPROM-okat olyan információk tárolására szokás alkalmazni, melyeket a készülék kikapcsolása után meg kell őrizni, de viszonylag ritkán szükség van az adatok megváltoztatására is.

Soros EEPROM

EEPROM-okból soros hozzáférésű változatok is léteznek. Ezeket olyan célra alkalmazzák, mint az EEPROM-okat, de olyan helyeken, ahol fontos a kevés jelvezeték, és nem okoz problémát az adatok lassabb elérhetősége (pl. mikrokontrolleres alkalmazások, telefonkártya stb.).
Ilyen soros EEPROM-ra példa a 93LC46 (1.51. ábra).



1.51. ábra. 93LC46 soros EEPROM

Az ORG bemenet alacsony ill. magas szintre kapcsolásával beállítható az EPROM szervezése, vagyis, hogy egy olvasás vagy írás parancs hatására 8 vagy 16 bitesen kell értelmezni az adatokat. Szétválasztott adat be- és kimenettel rendelkezik, amelyek azonban az alkalmazások többségénél összeköthetők.

A CS és DI egyszerre magas szintje alatt érkező CLK felfutó éle jelzi az áramkörnek egy átviteli ciklus kezdetét. Ezután egy utasítást (opcode) és ha kell, adatbiteket vár a chip, a CLK felfutó élével szinkronban, ezek határozzák meg a további működést.

7- féle utasítás létezik:

READ: Az opcode után a címbiteket kell beléptetni, majd a chip sorosan kiadja az adatbiteket, az órajel felfutó éleinek hatására.

EWEN: A törlés és írás engedélyezése.

ERASE: A beléptetett címen levő tartalom törlése (FFH byte-tal feltöltése).

ERAL: Az egész EEPROM törlése.

WRITE: A beléptetett címre beírja a beléptetett adatot.

WRALL: Az egész chipet feltölti a beléptetett adattal.

EWDS: Letiltja a törlést és az írást.

Az írás jellegű utasítások viszonylag lassan hajtódnak végre (4msec).
Az EEPROM tartalmát megváltoztató parancsok előtt EWEN parancsot kell kiadni, majd a tartalom megváltoztatása után az EWDS paranccsal kell zárni az eljárást.

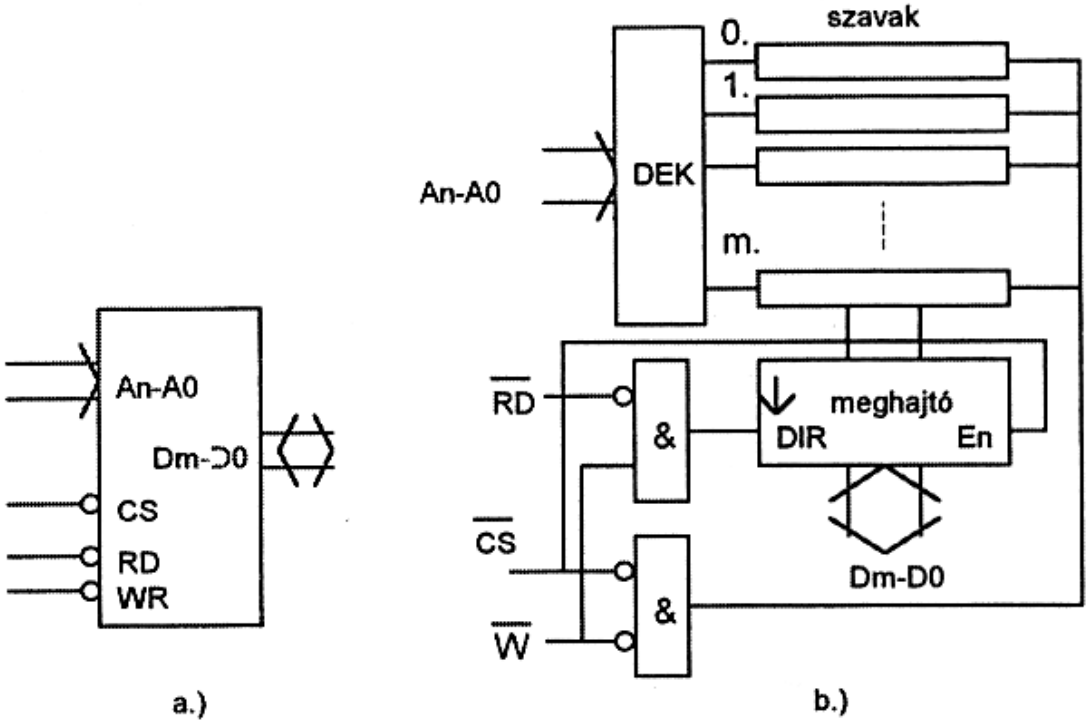
1.3.2. RAM

A RAM-ok, írható olvasható tárolók, melyek azonban a ROM-okkal ellentétben a tápfeszültség megszűnésekor elfelejtik tartalmukat. RAM-okat alkalmaznak a mikroprocesszoros rendszerekben változók és a működés közben lecserélhető programok tárolására, video kártyákban a karakteres és grafikus információk tárolására stb. A RAM-ok egy beíró (WR) jellel is rendelkeznek. Az erre a bemenetre adott megfelelő szélességű impulzussal írható be az adat a megcímzett rekeszbe.

Statikus RAM

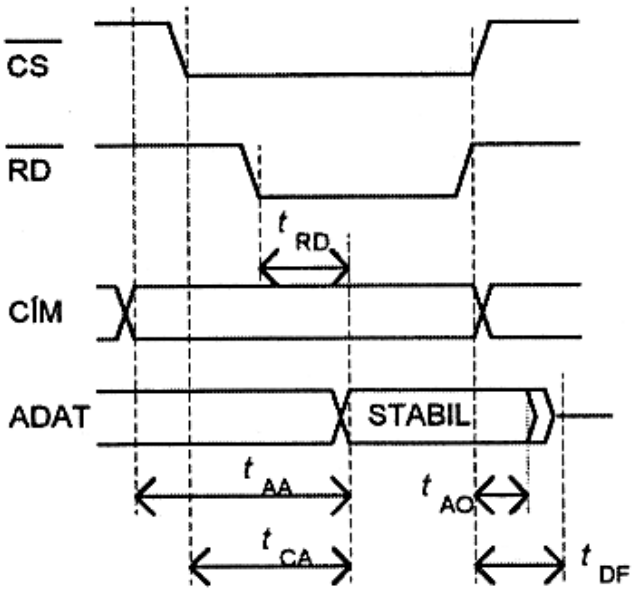
A statikus RAM-okban az információt bistabil (FET) tranzisztoros áramkörök tárolják, s egy bit tárolásához több tranzisztor szükséges. A tárolás elvéből adódóan nincs szükség frissítő áramkörökre mint a dinamikus RAM-ok esetében, így a tervezés ezekkel az áramkörökkel egyszerűbb. Sajnos a nagyobb chip felület igény miatt, csak kisebb méretű statikus RAM-ok készíthetők, mint DRAM-ok. E könyv írásakor a statikus RAM-ok kapacitása 4 Mbites nagyságrendű. A hozzáférési idejük kb. 5-150 nsec. A statikus RAM-ok fogyasztása rendkívül kicsi. Ezért léteznek olyan típusok is, ahol az IC tokba lítium elemet is beépítenek, s az így kapott nem felejtő RAM-ra 10 évet garantálnak. Léteznek olyan kvázi statikus RAM-ok is, amelyek belül dinamikus RAM-ként vannak felépítve, de a frissítő áramkört is tartalmazza a chip. Azonban ezek fogyasztása jelentősen nagyobb a valódi statikus RAM-okénál.

A statikus RAM jelölését mutatja az 1.52a ábra, durva blokkvázlatát pedig az 1.52b ábra.



1.52. ábra. Statikus RAM

A statikus RAM olvasási ciklusának idődiagramját és a katalógusokban szereplő jellemző időadatokat mutatja az 1.53. ábra.



1.53. ábra. Statikus RAM olvasási ciklusa

Az olvasási időadatok hasonlóak a ROM-oknál ismertetettekhez, ezért azokat nem részletezzük.

A statikus RAM írási ciklusának idődiagramja látható az 1.54. ábrán. A legfontosabb katalógusbeli időzítések:

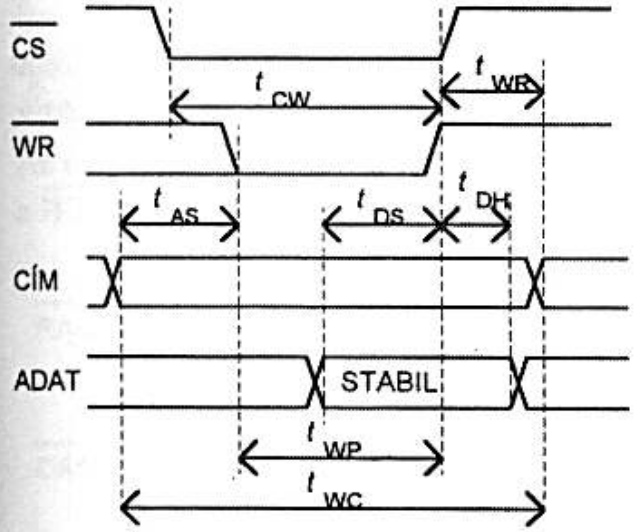
t_{AS}: cím előkészítési idő, a címnek legalább ennyi idővel a WR aktivizálása előtt meg kell jelennie,

t_{DS}: adat előkészítési idő, az adatnak legalább ennyi idővel a WR megszűnése előtt meg kell jelennie,

t_{DH}: adat tartási idő, az adatnak legalább ennyi ideig a WR megszűnése után még stabilan kell maradnia,

t_{WR}: cím tartási idő, legalább ennyi ideig nem szabad megváltoztatni a címet a WR megszűnése után,

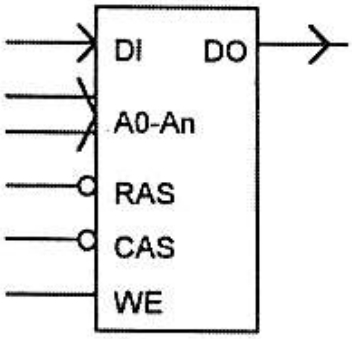
t_{WC}: írási ciklus idő, ciklikusan ennyi időnként lehet újabb adatot a memóriába írni.



1.54. ábra. Statikus RAM írási ciklusa

Dinamikus RAM

A dinamikus RAM-okban egy bit információt egy FET tranzisztor vezérlő elektródjának felületére felvitt töltések tárolnak. Ez a kis kapacitás viszonylag rövid idő alatt kisül, ha magára hagyják, ezért ciklikus időközönként a töltés felfrissítésére van szükség. Erre külön kiegészítő áramköröket tartalmaz a DRAM, de a **frissítés** vezérlését kívülről, a felhasználónak kell megoldania. Erre speciális frissítő áramkörök kaphatók. A dinamikus RAM jelölését mutatja az 1.55. ábra.

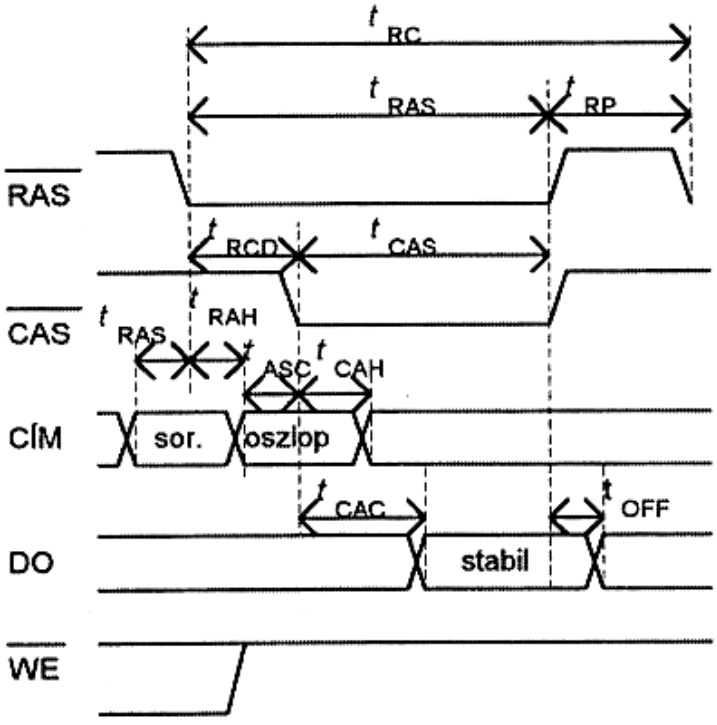


1.55. ábra. Dinamikus RAM jelölése

A dinamikus RAM-ok előnye, hogy kisebb chip felületen tárolható ugyanannyi információ mint a statikus RAM-okban. A könyv írásakor a dinamikus RAM-ok kapacitása 16 Mbit-es nagyságrendű. Hozzáférési idejük kb. 15-120 nsec.

A dinamikus RAM-ok címzése ill. az adat írása és olvasása eltér a statikus RAM-tól. A dinamikus RAM-nak 3 vezérlőjele van: $\overline{RAS}$, $\overline{CAS}$, $\overline{WE}$. Ráadásul szétválasztott adat be- és kimenettel rendelkeznek, amelyeket azonban sok alkalmazásban egyszerűen össze lehet kötni. Mivel a dinamikus RAM-ok kapacitása igen nagy a szokásos statikus RAM-okhoz képest, a címzésükhöz sok lábra lenne szükség a statikus RAM-okhoz hasonló kialakítás esetén, ami nagyon megnövelné a költségeket. A problémát úgy oldják meg, hogy **a címzés multiplexálva**, két részletben történik. Ugyanazon címbemenetekre kell ráadni a cím alsó (sor cím), majd a felső (oszlop cím) részét. A **sor cím** a $\overline{RAS}$ (Row Address Select), az **oszlop cím** pedig a $\overline{CAS}$ (Column Address Select) jel első élének hatására íródik be a DRAM címregiszterébe.

Az olvasási ciklus során be kell írni az említett módon a teljes címet, s az jelenti az olvasás engedélyezését, amikor a $\overline{WE}$ inaktív, a $\overline{RAS}$ és $\overline{CAS}$ pedig egyszerre aktív, amint az 1.56. ábrán látható.



1.56. ábra. Dinamikus RAM olvasási ciklusa

A DRAM olvasás legfontosabb katalógusbeli időadatai:

t_{RC}: Az egymást követő olvasások ciklusideje

t_{ASR}: A sor cím RAS lefutó éléhez viszonyított setup time-ja,

t_{RAH}: A sor cím RAS lefutó éléhez viszonyított hold time-ja

t_{ASC}: Az oszlop cím CAS lefutó éléhez viszonyított setup time-ja,

t_{CAH}: Az oszlop cím CAS lefutó éléhez viszonyított hold time-ja,

t_{RCD}: A CAS a RAS-hoz képest maximum ennyit késhet, hogy olvasás esetén az adat a RAS lefutó éle után t_{RAC} idővel megjelenjen, egyébként t_{RAC}-nál később jelenik meg az adat annyi idővel, amennyivel t_{RCD}-t túlléptük,

t_{CAS}: A CAS-ra előírt impulzusszélesség tartomány

t_{RAS}: A RAS-ra előírt impulzusszélesség tartomány

t_{RP}: Két RAS között legalább ennyi időnek el kell telni,

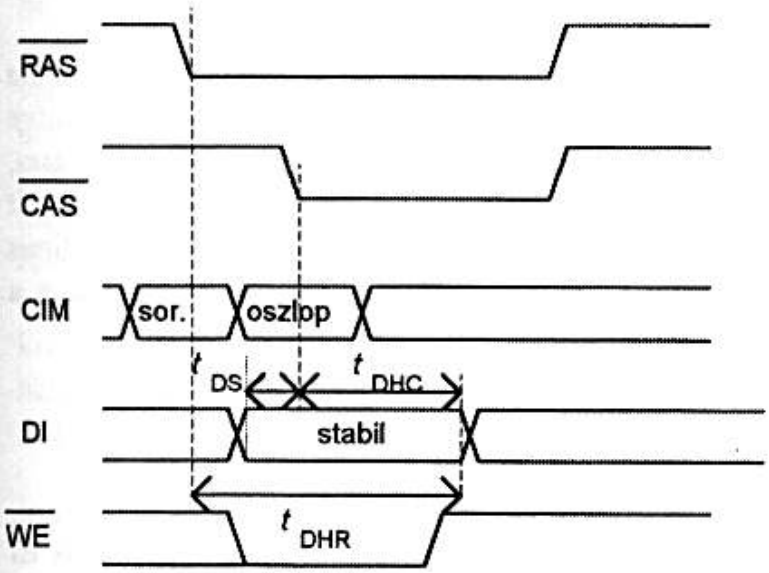
t_{CAC}: A CAS-hoz képest ennyi idővel jelenik meg az adat (CAS-hoz képesti hozzáférési idő),

t_{RAC}: A RAS-hoz képesti hozzáférési idő,

t_{OFF}: A CAS után ennyi idővel szűnik meg az adat,

t_{RC}: Az olvasás ciklusideje véletlenszerű hozzáférés esetén,

Az írás esetén a címzés hasonló az olvasásnál ismertetetthez, de a $\overline{CAS}$ megjelenése előtt a $\overline{WE}$ jelet aktivizálni kell.



1.57. ábra. Dinamikus RAM írási ciklusa

Az adat a $\overline{CAS}$ lefutó élére íródik be a DRAM-ba (1.57. ábra). Természetesen a CAS lefutó éle előtt előírt idővel az oszlop cím már nem változhat.

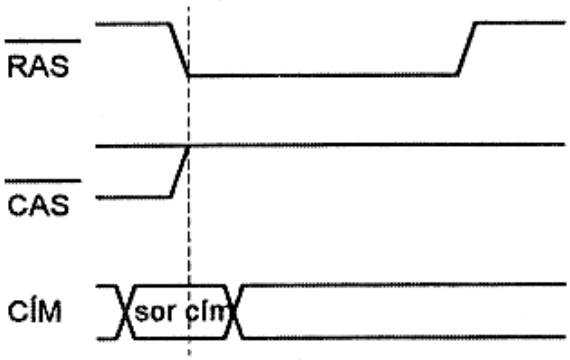
A DRAM írás legfontosabb időadatai hasonlóak az olvasásnál ismertetettekhez. Kiemelve a legfontosabb hozzájött adatokat:

t_{DS}: Az adat CAS lefutó éléhez képesti setup time-ja.

t_{DHC}: Az adat CAS lefutó éléhez képesti hold time-ja.

t_{DHR}: Az adat RAS lefutó éléhez képesti hold time-ja.

Ha a dinamikus RAM-hoz hosszabb ideig nem fordulnak, akkor egy idő után elfelejti az adatokat, ezért ilyenkor **frissítésre** van szüksége, amit egy kiegészítő áramkör végez (ezt egy chipbe szokás tenni, a cím multiplexálást végző áramkörrel). A frissítési ciklus során csak sorcímet kell adni a DRAM-nak, s végig kell címezni a tartományt. A frissítési ciklusok közötti idő (amíg a teljes tartományt felfrissítjük) kisebb kell, hogy legyen egy a katalógusban megadott értéknél (4-64 msec nagyságrendű). A frissítési ciklust mutatja az 1.58. ábra.



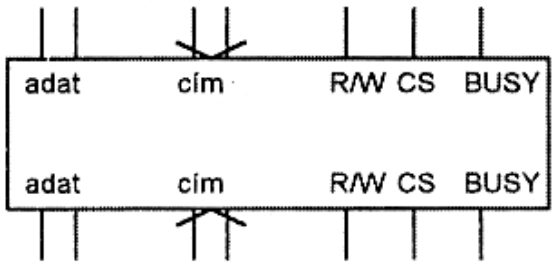
1.58. ábra. Dinamikus RAM frissítési ciklusa

NOVRAM (Non Volatile RAM)

Ez nem felejtő RAM, amely az EEPROM-ból és a RAM-ból létrehozott öszvér. Benne egy-egy RAM cella és EEPROM cella van összekapcsolva. Funkcióját tekintve közönséges RAM-hoz hasonló. A RAM cellákat a szokásos módon lehet írni és olvasni, de a RAM és EEPROM között egy átírási folyamatot is el lehet indítani (mindkét irányban), ami 10 msec nagyságrendű ideig tart. Ezzel egyesítették a két memória típus előnyeit, a RAM gyors hozzáférési idejű, de kikapcsolás előtt elmenthető a tartalma a nem felejtő EEPROM-ba.

Dual port RAM

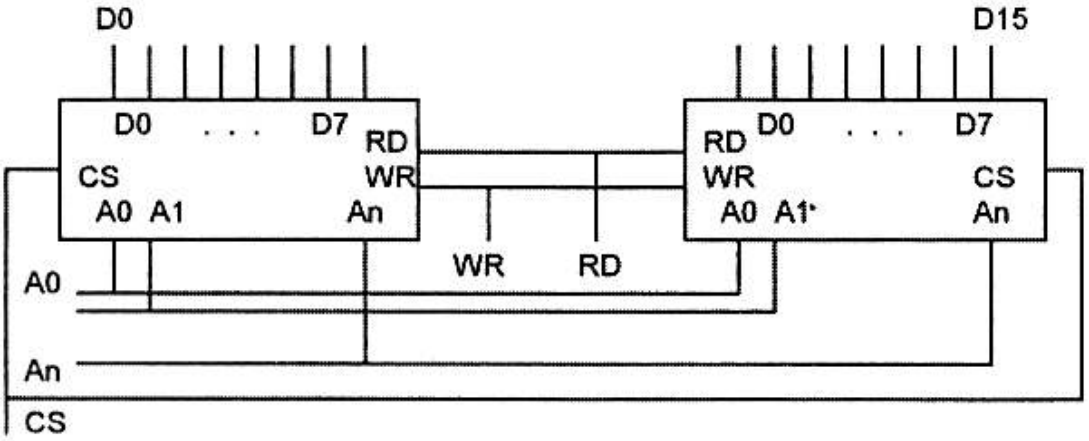
A dual port RAM olyan RAM, amelyet egyszerre két helyről lehet megcímezni, írni, olvasni, ennek megfelelően mindkét hozzáférési felületen megtalálhatók a címvonalak és vezérlő vonalak. Akkor van probléma, ha mindkét oldalról ugyanazon című rekeszbe akarnak írni. Ekkor egy belső arbiter (döntési) logika dönti el, hogy kié az elsőség, s addig a másik oldal foglalt jelzést kap a chip BUSYi (i=1,2) kimenetén. A dual port RAM jelölését mutatja az 1.59. ábra.



1.59. ábra. Dual port RAM

Memóriák szószélességének növelése

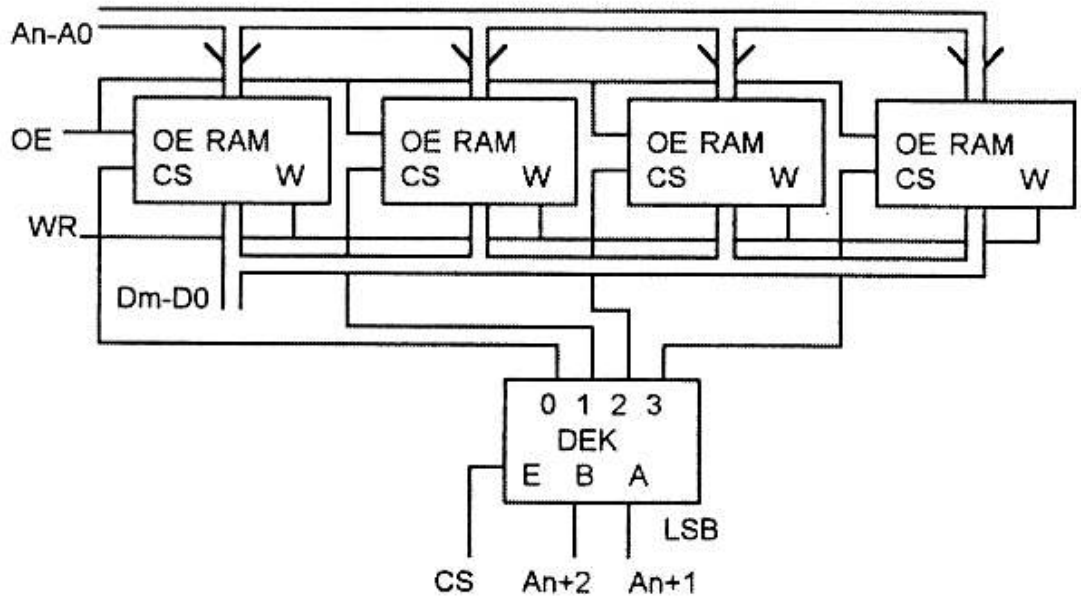
Ha egy memória szószélessége nem elegendő, akkor egyszerűen megnövelhetjük azt, csak a címeket és a vezérlőjeleket kell párhuzamosan kapcsolnunk. Az 1.60. ábra két 8 bites szószélességű RAM-ból készített 16 bit szószélességű RAM-ot mutat.



1.60. ábra. Memóriák szószélességének növelése

Memóriák kapacitásának növelése

Ha egy memória kapacitása nem elég, akkor több egységből kell összerakni a kívánt méretet. Az 1.61. ábrán mutatott kapcsolással megnégyszerezzük a rendelkezésre álló RAM memória méretét. Egy dekódert címeznek a felső bitek, mely kimenetei közül a megcímzett a megfelelő RAM-ot engedélyezi, a vezérlőjelek csak erre lesznek hatásosak. A dekóder engedélyező bemenetét mint globális chip select jelet használhatjuk. A memóriák megfelelő vezérlő, adat és cím vonalait egyszerűen párhuzamosan kötjük. A kiválasztott chipen belüli címet a párhuzamosan kötött címvezetékekre adott cím határozza meg. A kiválasztott chippel lehet a vezérlőjelek által meghatározott műveletet (írás vagy olvasás) végezni.



1.61. ábra. Memóriák kapacitásának növelése

2. PROGRAMOZHATÓ LOGIKÁK

Ez a fejezet, a többitől kicsit elkülönülve, önálló egységet alkot. Részletes tanulmányozása nem szükséges a továbbiak megértéséhez. Főként azoknak az olvasóknak ajánljuk, akik mélyebben érdeklődnek a programozható logikák iránt.

Az áramkör technológia egy IC-n belül egyre nagyobb alkatrészsűrűséget tesz lehetővé. Így a tendencia az, hogy az egyre bonyolultabb logikákat egyre kisebb helyen lehet elhelyezni. A régen több kártyából álló készüléket ma már egyetlen kártyán, esetleg egyetlen IC-ben meg lehet valósítani.

A nagybonyolultságú IC-k alkalmazása a digitális tervezés módszereit is jelentősen befolyásolja, mivel az ezekkel való tervezés lehetetlen számítógépes segítség, egyre nagyobb tudású CAD (Computer Aided Design) rendszerek nélkül.

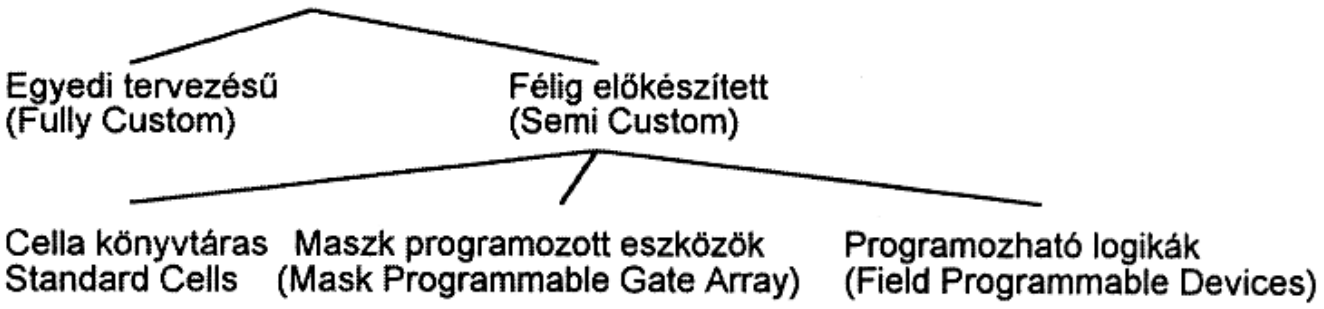
Különösen nagy jelentősége van a programozható logikák elterjedésének. Ezekkel az eszközökkel jelentősen lerövidíthető a tervezési ciklus, ami nagymértékű anyagi megtakarítást jelent, s a későbbi változtatások jó része is a NYÁK (nyomtatott áramkör) terv módosítása nélkül lehetséges.

E fejezet célja, hogy rövid áttekintést adjon a fenti áramkörökről, és tervezési módszerekről. Az áramkörök ismertetésénél csak a programozható logikákra koncentrálunk, azoknak a prototípus tervezésben való nagy jelentősége miatt.

A fejezetben gyakran használjuk a pterm rövidítést. Ez az ÉS kapcsolattal megvalósított termekre utal (product term).

2.1. Felhasználó által specifikált ill. programozható eszközök csoportosítása

A nagy integráltságú, felhasználó által specifikált és/vagy felhasználó által programozható eszközöket az előkészítettségük alapján az alábbiak szerint szokás felosztani :



2.1. ábra. Felhasználó által specifikált ill. programozható eszközök csoportosítása

Egyedi tervezésű (Fully Custom)

A felhasználót csak a technológia köti, bármit elkészíthet, amit a technológia lehetővé tesz. Főként nagyon egyedi áramkörök megvalósítására alkalmazzák. Ilyen módon nagy sebességű és jó helykihasználású (nagy alkatrész-sűrűségű) tervek készíthetők, viszont a prototípus elkészítése rendkívül drága.

Félig előkészített (Semi Custom)

Meg kell különböztetnünk, hogy magát az IC-t készítik-e félig elő a tervezéshez (MPGA, FPLD), vagy csak a tervező rendszer nyújt segítséget (cellakönyvtáras).

Cellakönyvtáras (Standard Cells)

Itt a tervező rendszer ún. cellakönyvtárakat ad a felhasználó kezébe. A cellakönyvtárak különféle digitális alapelemek és funkcionális egységek előre megtervezett lay-outjait (geometriai kialakítását) tartalmazzák. Így a tervező nem kell, hogy IC szinten jártas legyen a tervezésben (ellentétben a fully custom megközelítésű tervezéssel), hanem a cellakönyvtárakból választott elemekből építheti fel a tervet. A tervező rendszer feladata az egységek elhelyezése és huzalozásának megtervezése.

A cellakönyvtáras tervezés előnyei:
- a cellakönyvtárak használata nagyon megkönnyíti és meggyorsítja a tervezést,
- a prototípus elkészítése viszonylag gyors és valamivel olcsóbb mint egyedi tervezés esetén, de még mindig drága.

A módszer hátrányai:
- az előre tervezett könyvtári elemek miatt nagyobbak a kötöttségek, és nagyobb a terv redundanciája, kisebb alkatrészsűrűség érhető el, mint egyedi tervezés esetén.

Maszk programozott (MPGA)

Az MPGA (Mask Programmable Gate Array) IC-k előre elkészített, általánosan használható tranzisztor tömböket tartalmaznak, melyek lay-outját az utólag megtervezendő egy- vagy többrétegű fémezéses összeköttetés megkönnyítésére optimalizálták. A tervező rendszerek a terv bevitele után, az IC-ben rendelkezésre álló tranzisztorok összeköttetésének megtervezésével automatikusan létrehozzák az utolsó technológiai lépésben felviendő fémezési maszk tervét.

Legfontosabb tulajdonságai:
- mivel csak a fémezési maszkot kell az IC-re felvinni, az elkészítés sokkal olcsóbb, mint az előzőekben, de még mindig sokkal drágább, mint a következőkben ismertetendő felhasználó által programozható áramkörök esetén,
- a tervezés és a prototípus elkészítése gyors,
- az elérhető alkatrészsűrűség kisebb, mint a cellakönyvtáras esetében.

Felhasználó által programozható eszközök (Field Programmable Devices)

Ezek az eszközök a felhasználó által programozhatók, vagyis ahhoz, hogy a megvásárolt általános célú áramkörből a felhasználó által specifikált áramkör fizikailag létrejöjjön, nem kell az IC gyártóhoz fordulni, az labor körülmények között, speciális programozó készülék segítségével elkészíthető (beprogramozható).

Ez a tulajdonságuk forradalmian megnöveli a hardver fejlesztők lehetőségeit:

- alkalmazás specifikus IC-k (ASIC) elkészítésére nagyon alkalmas,
- a prototípus elkészítése nagyon gyors és nagyon olcsó, a többi lehetőséghez képest,
- a terv módosítása is egyszerűbb,
- kis sorozatszámú és nagy integráltságú, speciális áramköröket igénylő hardver fejlesztése, gyártása gazdaságosabb.

A felhasználó által programozható logikák csoportosítása

A felhasználó által programozható eszközöket bonyolultságuk és felépítésük szerint az alábbi csoportokra oszthatjuk, a tárgyalásuk is ezt követi:

Kisebb bonyolultságú eszközök (PROM-ok, bipoláris, ECL és CMOS technológia):
Programmable Logic Devices (PLD-k, AND-OR struktúra, főként CMOS technológia),

PAL (AND mátrix programozható),
PLA (AND és OR mátrix programozható).

Közepes és nagy bonyolultságú eszközök (CMOS technológia):

CPLD (komplex PLD-k, egy eszközben több PLD struktúrájú áramkör),
FPGA (Field Programmable Gate Array, egy eszközben rendkívül sok, viszonylag egyszerű logikai tömb, programozható összeköttetésekkel).

A felsorolt és egyéb különleges tulajdonságuk miatt, a továbbiakban ezeket az eszközöket ismertetjük nagyobb részletességgel, az egyszerűbb, kisebb integráltságú eszközöktől kezdve, egészen a legnagyobb integráltságú IC-kig.

2.2. A felhasználó által programozható logikák programozásának technológiái

A programozható logikák előre elkészített különféle architektúrájú logikai egységeket tartalmaznak. Az egységek konfigurációja, ill. az egységek összeköttetése programozható, ily módon valósítva meg a célnak megfelelő logikát. A programozásra több, tulajdonságaiban jelentősen eltérő technológiát dolgoztak ki.
A legfontosabb szempontok:
- egyszer vagy többször programozható,
- a tápfeszültség kikapcsolásakor felejt vagy nem (az egyszer programozható eszközök természetesen nem felejtenek),
- a megvalósításhoz szükséges chip terület.

2.2.1. Egyszer programozható eszközökben használatos programozási technológiák

Programozás biztosítékkal (fuse)

A biztosítékot alkalmazták legelőször programozási célra. Ez egy speciálisan kialakított kisméretű fém összeköttetés, melyet a programozás során megfelelő áramú programozó impulzus alkalmazásával elégethetnek, így megszakítható az összeköttetés. Kisebb integráltságú eszközöknél, PROM-ok és PLD-k esetében alkalmazzák.

Programozás anti-fuse-zal

Ez egy olyan technológia, amellyel egy alapvetően jó szigetelő polykristályos szilícium vagy speciális dielektrikum vezetővé tehető 10-18V-os programozó feszültség alkalmazásával. Nagyon kicsi a felület igénye más programozási eljárásokhoz képest, de 3 extra maszk szükséges a CMOS technológia hagyományos maszkjain kívül, és nagyobb feszültségű tranzisztorok kialakítására is szükség van a programozó impulzus miatt, amelyek viszont nagy helyigényűek. A létrehozott összeköttetés ellenállása 50-500 ohm, járulékos kapacitása kicsi a nem biztosíték jellegű technológiákhoz képest, ami nagyobb sebességű eszközök létrehozását teszi lehetővé.

2.2.2. Többször programozható eszközöknél alkalmazott technológiák

Programozás EPROM és EEPROM segítségével

Mindkét esetben egy lebegő gate-es FET tranzisztor lebegő elektródjára felvitt töltéssel lehet a tranzisztor állapotát (be- vagy kikapcsolt) programozni. EPROM esetében a töltések eltávolítása (az eszköz törlése) UV fénnyel lehetséges, EEPROM esetében pedig elektromos úton. Az EEPROM azonban kb. kétszeres chip területet igényel, mint az EPROM.

Megjegyezzük, hogy az EPROM-os technológiát oly módon is alkalmazzák, hogy nem készítenek kvarc ablakot a chipre, így az törölhetetlen, vagyis egyszer programozható lesz.

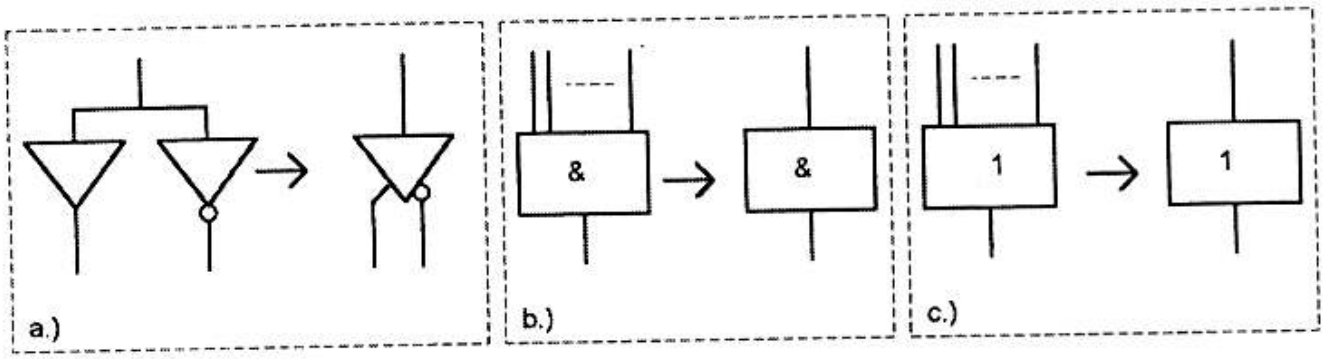
Programozás statikus RAM-al

A statikus RAM nagy előnye, hogy az így programozott eszköz gyorsan átkonfigurálható (sokkal gyorsabban, mint az EEPROM). Hátránya, hogy egy-egy SRAM cella nagy chip területet igényel, s minden bekapcsolásnál meg kell oldani a konfigurálást. Ezt vagy a RAM-oknál megszokott címzés kialakításával oldják meg, vagy shiftregiszterszerű kialakítással. Azonban a felprogramozáshoz járulékos áramkörök szükségesek, továbbá a konfigurációs adatokat tároló más eszköz (ROM, diszk).

2.3. Egyszerű PLD-k

A PLD-k olyan kis bonyolultságú kétszintű AND-OR struktúrájú áramkörök, melyek lehetővé teszik, hogy a felhasználó maga alakítson ki belőlük - típustól függő korlátok között - tetszőleges logikai (kombinációs ill. sorrendi) kapcsolást. Az SRAM-os programozási technológia kivételével az összes többi előfordul a különféle PLD típusoknál. Az áramköröket speciális programozó készülékkel lehet programozni, a konkrét égetési algoritmus szinte típusról típusra változik. Az alábbiakban először az egyszerű PLD-kkel (PAL, PLA, PLS, konfigurálható makrocellás PLD-k) foglalkozunk. Főként egyszerű SSI logikák vagy MSI IC kiváltására alkalmazzuk. (Pl. mikroprocesszoros rendszer címdekódere, egyszerű vezérlők, speciális kódolású vagy nem standard modulusú számlálók stb.)

A PLD-k rajzolásánál a rajzok egyszerűsítése végett a 2.2 ábra szerinti jelöléseket vezetjük be.



2.2. ábra. Jelölések

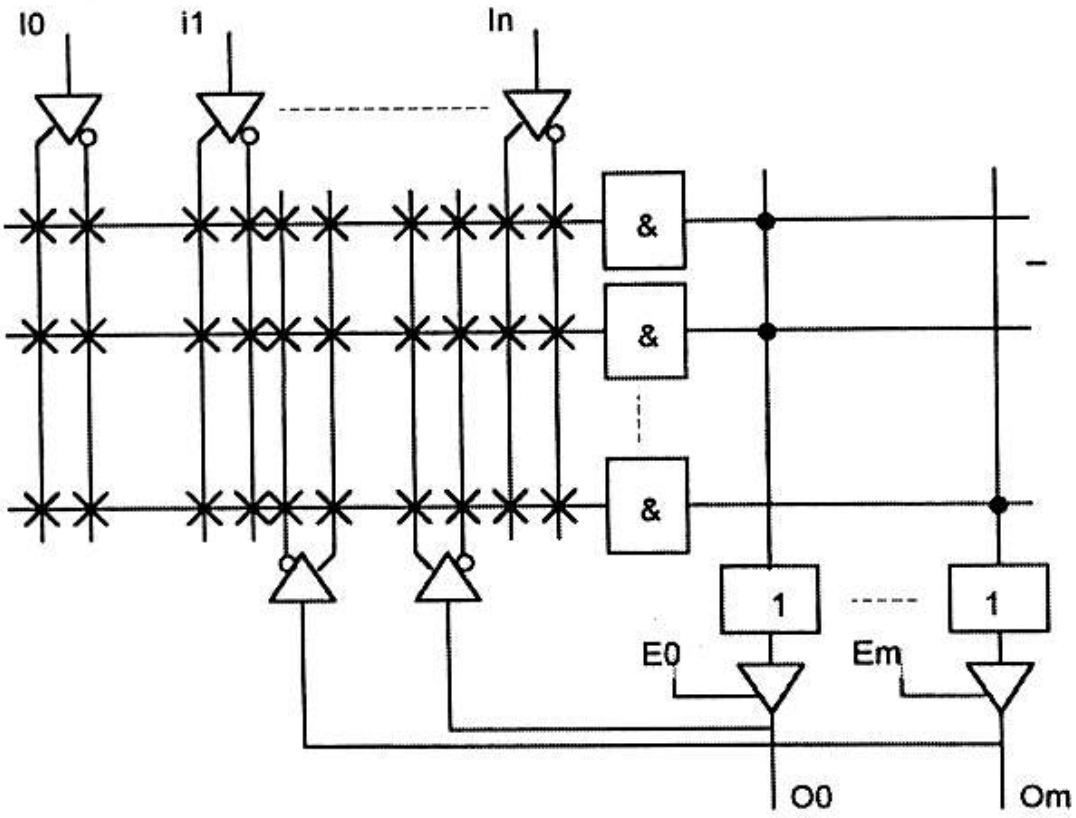
Az invertált és ponált jel előállítását a 2.2a ábra szerint jelöljük. A több bementű kapuk bemeneteit csak egy vonal jelöli (2.2b és 2.2c ábrák).

2.3.1. PAL (Programmable Array Logic)

A PAL-nál *az összes bemeneti változó ponálva és negálva is rákapcsolódhat minden egyes AND kapura*, egy-egy biztosítékon keresztül. Az AND kapuk csoportonként rá vannak kötve a kimeneteket előállító egy-egy OR kapura. Így a struktúra által korlátozott term méreten és term számon belül tetszőleges logikai függvény előállítható az áramkörrel, de jelentős korlát, hogy *csak az AND mátrix programozható*. Összetettebb függvény is megvalósítható, ha többszintű logikát alakítunk ki a kimenetek visszavezetésével, azonban a jelterjedési idő ekkor a szintek számától függően megnő.

A 2.3. ábra egy fuse-os, programozatlan PAL-t mutat. A vonalak kereszteződéseibe (mátrix pontokba) tett x azt jelenti, hogy az adott bemenet (ponáltja ill. negáltja) egy soros biztosítékon keresztül rákapcsolódik a kijelölt kapu egy bemenetére. Beprogramozás után a kiégetett biztosítékok helyén nincs kereszt. A kimeneti OR kapukra csak a kereszteződésben ponttal jelölt AND kapuk kapcsolódnak. Ez azt jelenti, hogy csak maximum ennyi termet tartalmazhat egy logikai függvény. A PAL-ok

kimenete három állapotú, az engedélyezést vagy közvetlenül egy erre dedikált bemenet, vagy kimenetenként egy logikai függvény (többnyire egyetlen term) végzi.



2.3. ábra. PAL blokkvázlata

Kétirányú I/O

Azok a kimenetek, amelyek vissza vannak csatolva az AND mátrixba, bemenetként is használhatók (*kétirányú I/O*), így az IC lábak a bemenetek és kimenetek között a feladathoz alkalmazkodva megoszthatók. Ekkor a buffert természetesen le kell tiltani (fixen, ha csak bemenetként használjuk, vagy egy termmel, ha kétirányú portként).

Invertálható kimenet

Egyes PAL-okban a kimeneti buffer előtt egy EXOR kapu van, melynek másik bemenete programozható logikai szintet kap. Ennek segítségével a kimenet programozáskor invertálható (az EXOR-on 1 szint). Így ha a komplemens függvény tartalmaz kevesebb termet, akkor azt lehet megvalósítani, majd a kimenetét invertálni (pl. AmPAL18P8B az AMD-től).

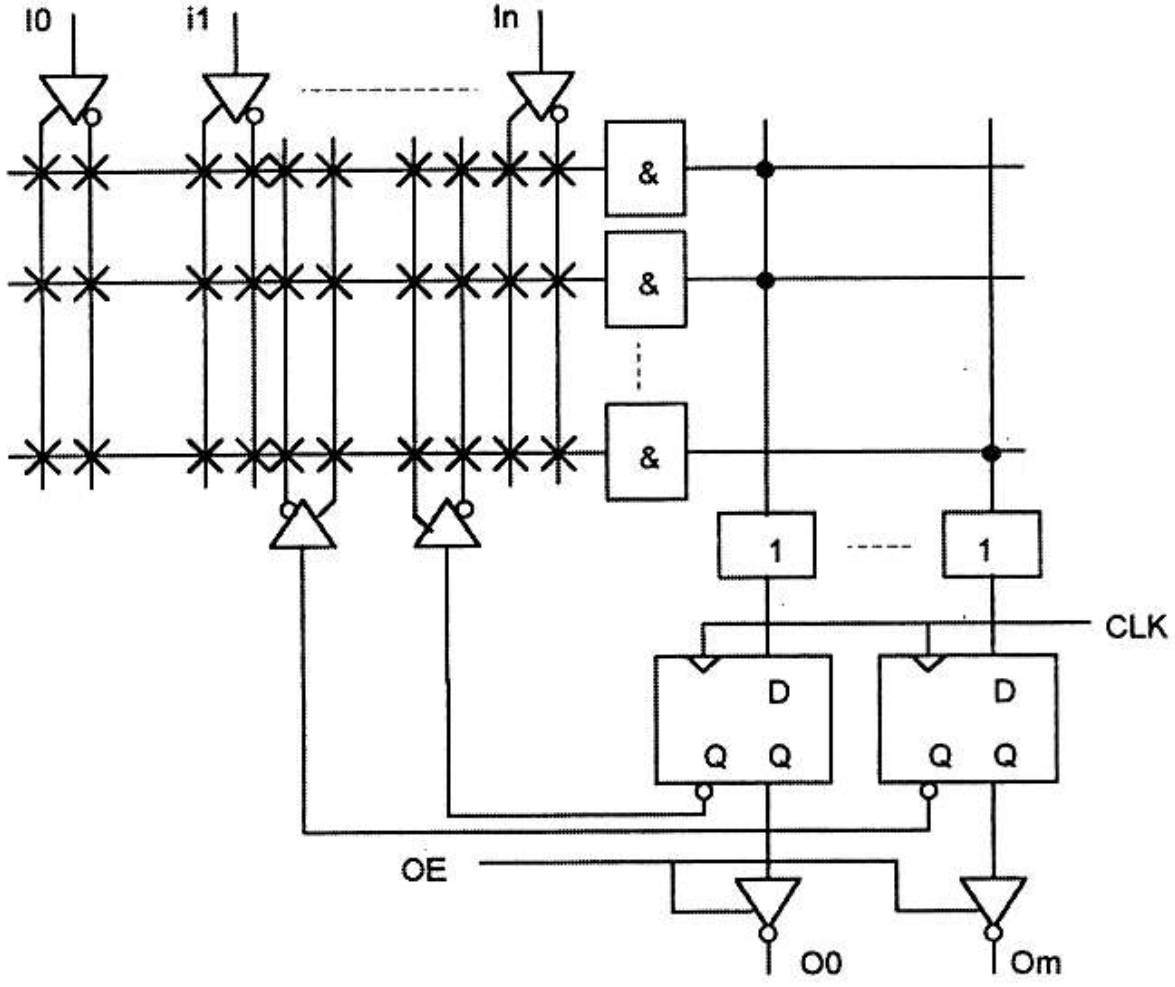
Security fuse

A PLD-k konfigurációs tartalma (kiégetett biztosítékok helye) a programozás után ellenőrzés céljából kiolvasható. Azonban tartalmaznak egy olyan biztosítékot, melynek kiégetése után már nem lehetséges a kiolvasás. Ha másolás ellen védeni akarjuk az áramkörünket, akkor ezt a biztosítékot ki kell égetni. Persze ennek igazán csak sorrendi áramkörök esetén van értelme, hiszen a kombinációs hálózatok igazságtáblája könnyen kideríthető.

Tipikus kombinációs PAL-ok: PAL 16L8, PAL20L8. A számok közötti L betű a kombinációs jellegre utal.

Regiszteres PAL

Az előbbi PAL struktúrát kiegészítették kimeneti regiszterekkel (D flip-flop). A flip-flopok kimenete a chipen belül visszacsatolható, így a szokásos szinkron sorrendi hálózat struktúra alakul ki (Moore modell, a kimenet csak az állapottól függ). Egyes típusokban a szokásos kombinációs kimenet is megtalálható, ami Mealy modell kialakítását is lehetővé teszi. A standard regiszteres PAL-okban a flip-flop kimenete és a kimeneti láb között invertálás történik. A regiszteres PAL blokkvázlatát mutatja a 2.4. ábra.



2.4. ábra. Regiszteres PAL blokkvázlata

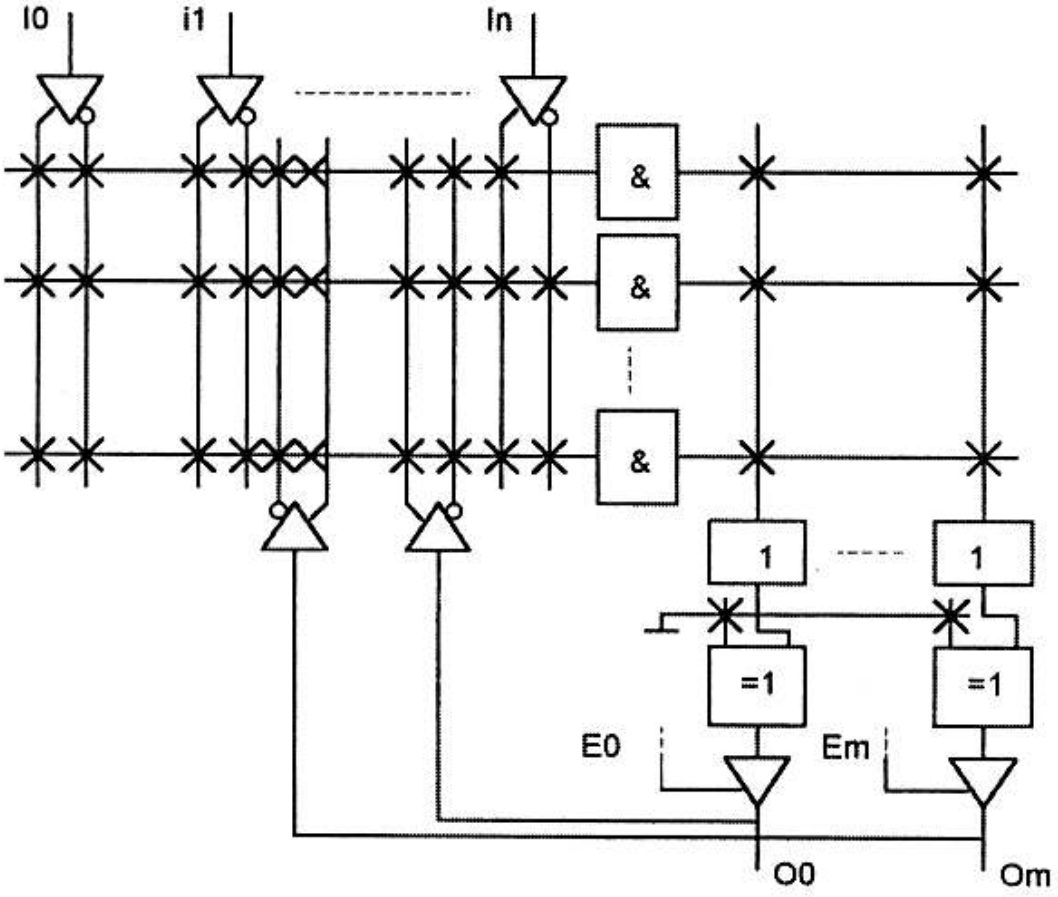
Bekapcsolási reset

A regiszteres PAL-ok regisztereinek a tápfeszültség megjelenése után 0 a kezdőállapota, a bekapcsolási reset (power up reset) áramkörnek köszönhetően. Ez megkönnyíti az inicializálást, mert a vezérlő függvénnyel már csak az 1 kezdeti értékű regiszterek beállítását kell megoldani. Viszont figyelemmel kell lenni arra, hogy a regiszterek törlése a kimenetet 1-be állítja, a kimeneti invertálás miatt.

Tipikus regiszteres PAL-ok: 16R4, 16R6, 16R8, 20R4, 20R6, 20R8. A számok közötti R betű a regiszterre utal.

2.3.2. PLA (Programmable Logic Array)

A PLA-k annyiban különböznek a PAL struktúrától, hogy *az AND és az OR mátrix is programozható* (2.5. ábra).



2.5. ábra. PLA blokkvázlata

Ez nyilvánvalóan kedvező, mivel a termek megoszthatók a kimeneti függvények között, míg a PAL esetében, ha egy függvényhez kevesebb term kell, mint amennyi az OR kapuhoz hozzá van rendelve, a maradékok elvesznek. A kombinációs kimenetek polaritása mindig programozható. (Pl. PLS 153, 8 bemenet, 10 I/O, 32 AND, 10 OR.)

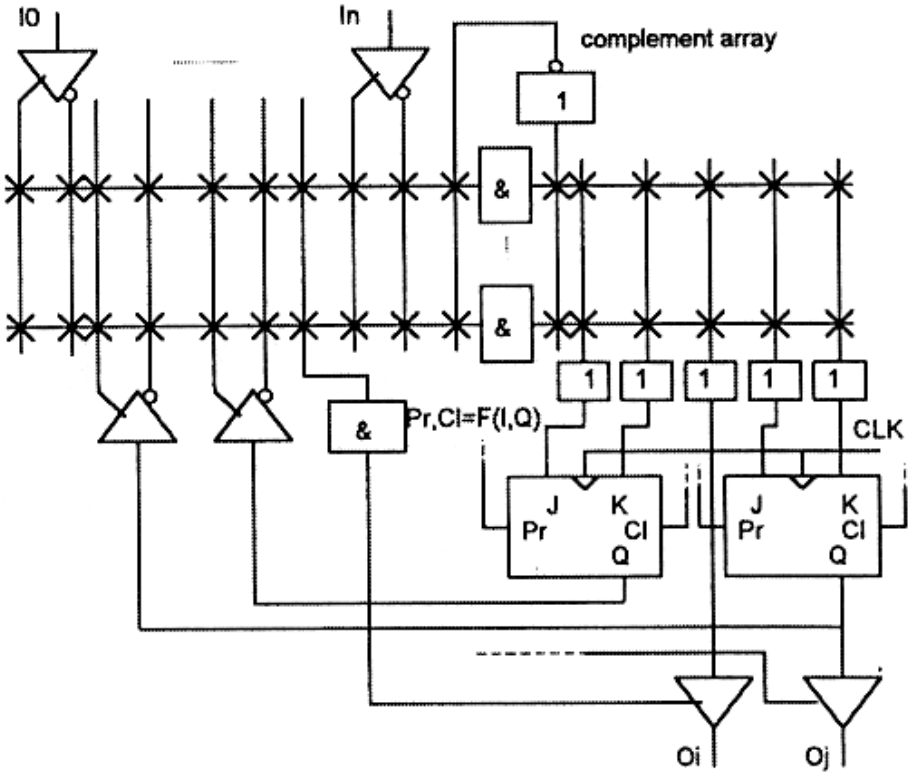
2.3.3. PLS (Programmable Logic Sequencer)

A PLA-k estében a regiszteres PAL-hoz képest a kimeneti regiszteren túl még egyéb, a szinkron sorrendi hálózatok tervezését megkönnyítő módosításokat is végeztek, s a kialakított struktúrát elnevezték PLS-nek.

Komplemens term (Complement array)

A módosítások közül talán a legjelentősebb, az ún. complemens term megjelenése, ami a 2. szinten jelenlévő OR kapuk mellett egyetlen NOR kapu, melynek kimenete visszavezetődik az AND kapukhoz, amint a 2.6. ábra mutatja. Ezt a struktúrát az állapotgráf ELSE típusú elágazásainak egyszerűbb megvalósítására lehet felhasználni.

Egy szinkron sorrendi hálózat állapotgráfjában egy Pi állapotból a bemenettől függően különböző állapotok következhetnek. A legtöbb esetben csak néhány, de nem az összes bemeneti kombináció érdekes, mert pl. a többi bemenet normál működés során nem léphet fel (nem teljesen specifikált hálózat). Azonban ilyenkor célszerű, ha egy előre definiált állapotba visszük az automatát, mert pl. ekkor képes lesz detektálni a hibás bemenetet. A feladatból is következhet ELSE típusú elágazás. Ennek a megvalósításakor használható ki az említett NOR kapu.



2.6. ábra. PLS blokkvázlata

Nézzük a 2.7. ábrán szereplő állapotgráf részletet:

Az S1 és S2 utáni állapotot előíró termek:

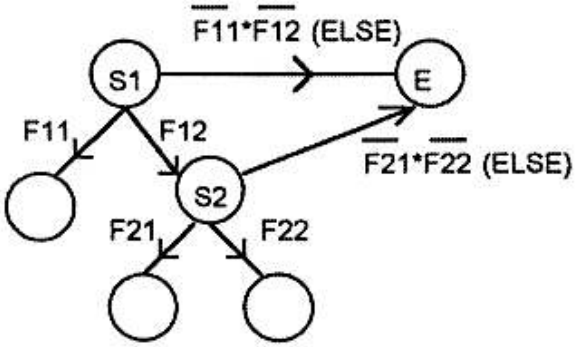
S_1F_{11}, S_1F_{12}, $S_1\overline{F}_{11}\overline{F}_{12}$

S_2F_{21}, S_2F_{22}, $S_2\overline{F}_{21}\overline{F}_{22}$

Az ELSE ágat kifejező rész átalakítható:

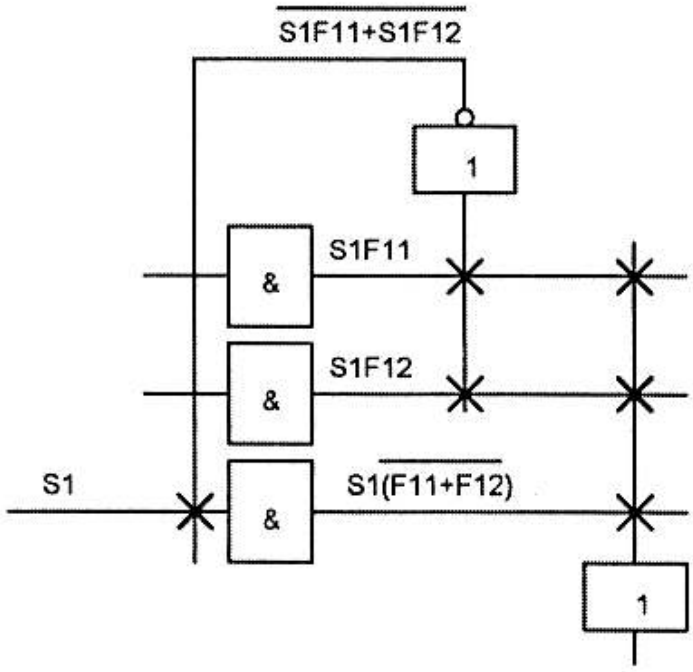
$$S_1\overline{F}_{11}\overline{F}_{12} = S_1\overline{F_{11}+F_{12}} = S_1\left(\overline{S_1}+\overline{F_{11}+F_{12}}\right) = S_1\overline{S_1\left(F_{11}+F_{12}\right)} = S_1\overline{S_1F_{11}+S_1F_{12}}$$

$$S_2\overline{F}_{21}\overline{F}_{22} = S_2\overline{S_2F_{21}+S_2F_{22}}$$



2.7. ábra. ELSE típusú elágazás egy állapotgráfban

Egy ELSE ág megvalósításához tehát a NOR kapun kívül szekunder változónként csak egy AND kapu szükséges, amint a 2.8. ábra mutatja:

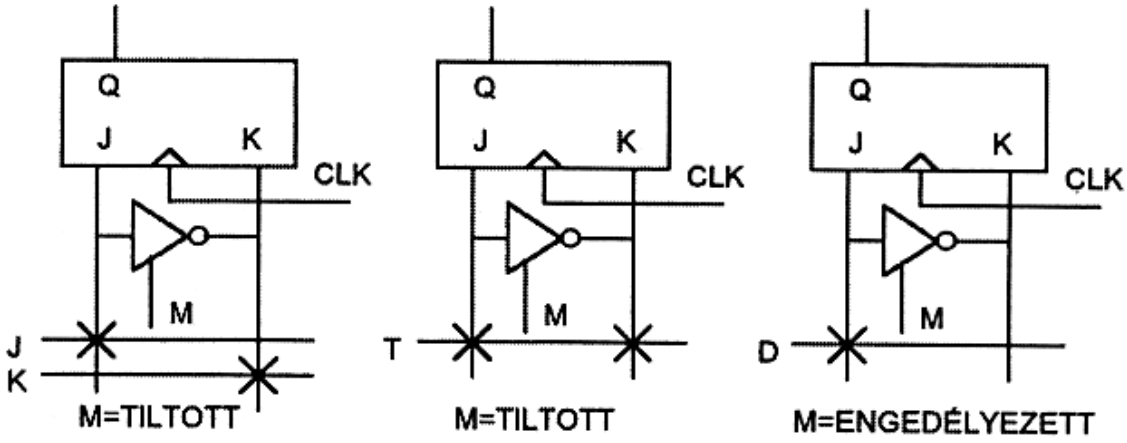


2.8. ábra. Az ELSE ág megvalósítása a PLS-ekben

Mivel a NOR kapura kapcsolódó függvényben az Si állapot AND kapcsolatban szerepel, *egyetlen NOR kapu tetszőleges számú állapotra vonatkozó ELSE ág megvalósításában részt vehet.* Azonban meg kell jegyezni, hogy a fenti struktúrát csak J-K flip-flopos realizálásokban lehet igazán kihasználni. Ezért a PLS-ek szinkron S-R, J-K, vagy konfigurálható (J-K/T/D) flip-flopot tartalmaznak.

Konfigurálható flip-flopok

A PLS-eknél megjelennek a konfigurálható flip-flopok. Így mindig a feladathoz legjobban alkalmazkodó flip-flopot lehet választani.
A programozható J-K/T/D flop-flopot J-K-ból szokták kialakítani. Egy lehetséges megvalósítást mutat a 2.9. ábra:



2.9. ábra. Konfigurálható flip-flop megvalósítása

A flip-flop J és K bemenetét egy három állapotú kimenettel rendelkező inverter köti össze. Alapállapotban a flip-flop J-K típusú. Ha mindkét bemenetre ugyanazt a termet kötjük J=K, akkor T flip-flopot kapunk. Ha az invertert aktivizáljuk, akkor a J bemenetet használhatjuk és D flip-flopot kapunk.

Kimeneti regiszter és "eltemetett" regiszter

A regiszteres PAL-okban kevés a kombinációs kimenet, így nagyon korlátozott a Mealy-szerű kialakítás lehetősége, a kimenet alapján történő állapotkódolás viszont jelentős megkötést jelent. Ezért a bonyolultabb PLS-ekben a regisztereket két részre osztották, s ezek közül csak az egyik kimeneteit vezették ki a chip I/O lábaira. Így kb. dupla annyi flip-flop áll a tervező rendelkezésére. Az "eltemetett" flip-flopok csak szekunder változókként, a többi pedig típustól függően csak kimeneti regiszterként használható (mert a visszacsatolás hiányzik), vagy szekunder változóként is. Hátránya a kialakításnak, hogy az állapot nem látszik a kimeneten, így nehezebb a beültetés utáni tesztelés. A PLS-ek egy része a PLA-knál megszokott felépítésű kombinációs kimenetekkel is rendelkezik.

Flip-flopok inicializálása

Az első generációs PLD-kben a kialakított sorrendi hálózat kezdőállapotának beállítását csak a vezérlő függvény megfelelő kialakításával ill. a bekapcsolási resettel lehetett megoldani. A modernebb programozható logikákban már a flip-flopok külön set és - típustól függően - reset bemenettel is rendelkeznek. Ezek a bemenetek aszinkron, szinkron esetleg vegyes (aszinkron set, szinkron reset) kialakításúak. A bemenetek vezérlését típustól függően egyetlen term vagy egy általános logikai függvény végezheti. Ezzel tehermentesítették a vezérlő függvényt előállító hálózatot, hiszen pl. egy flip-flop 1-be írása egy teljes AND kaput leköt.

Példa PLS-re: PLS 167, 6 kimeneti regiszter, 6 eltemetett regiszter (szinkron S-R), 14 bemenet, (aszinkron preset)/(output enable) bemenet. Speciális módon, az I/0 lábra 10V-ot adva, lehetőség van az eltemetett regiszter tartalmának megfigyelésére. Ezt a lehetőséget a programozó készülék használhatja ki, a beégetett áramkör tesztelésére.

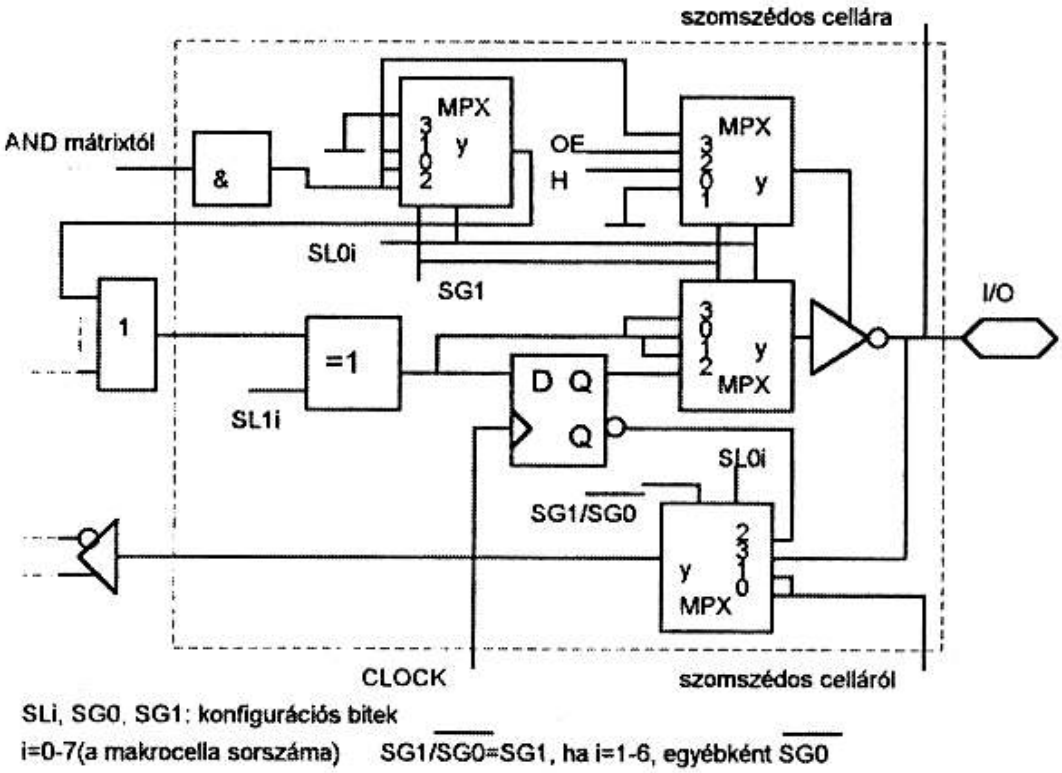
Illegális állapotba lépés aszinkron bemenet esetén

Egyes PLS-eknél az egy-egy OR kapura kapcsolódó termek számának függvényében változhat a késleltetési idő (akár 10 ns-ot is). Így az egyes flip-flopokhoz ugyanaz a logikai jel eltérő időpontban ér el. Ezért a setup time betartásánál a leghosszabb késleltetésre kell számítani. A setup time be nem tartása esetén (ez történik, ha a bemenet az órajelhez képest aszinkron módon változik), egy az órajel környezetében történő változást esetleg nem minden flip-flop vesz észre, ami illegális állapotba lépést eredményezhet! Ez mindig előfordulhat, de az eltérő setup time-ok ennek esélyét megnövelik. (Lásd a metastabilitást a 2.8.1. fejezetben.)

2.3.4. Konfigurálható makrocellás PLD-k

Az eddig ismertetett PLD struktúráknál egy-egy I/O láb dedikáltan kombinációs kimenet, regiszteres kimenet, vagy bemenet, esetleg kétirányú. Ugyanazt a kimenetet nem lehet egyik alkalmazásban kombinációsként, másikban regisztereseként programozni. Ez a kiosztás sokszor nem teszi lehetővé a PLD kihasználását, s ezen az eltérő kombinációs/regiszteres kimenet arányú típusválaszték csak keveset segít. Ezért a szokásos PLD struktúra kimentére olyan egységet tettek, mely lehetővé teszi, hogy a felhasználó az egység kimenetére csatlakozó IC lábat programozhatóan vagy kombinációs kimenetnek, vagy regiszteres kimenetnek, vagy bemenetnek, esetleg kétirányú I/O-ként használja. Természetesen kombinációs és regiszteres esetben is megvan a lehetőség a visszacsatolásra, így aszinkron és szinkron sorrendi hálózat is kialakítható. Az így létrejött, típusonként és gyártónként némileg különböző struktúrát nevezik *makrocellának*. A nagy változatosság miatt általános struktúrát nehéz lenne rajzolni, ezért inkább néhány konkrét áramkört ismertetünk.

A GAL16V8 makrocelláját mutatja a 2.10. ábra.



2.10. ábra. A GAL 16V8 makrocellája

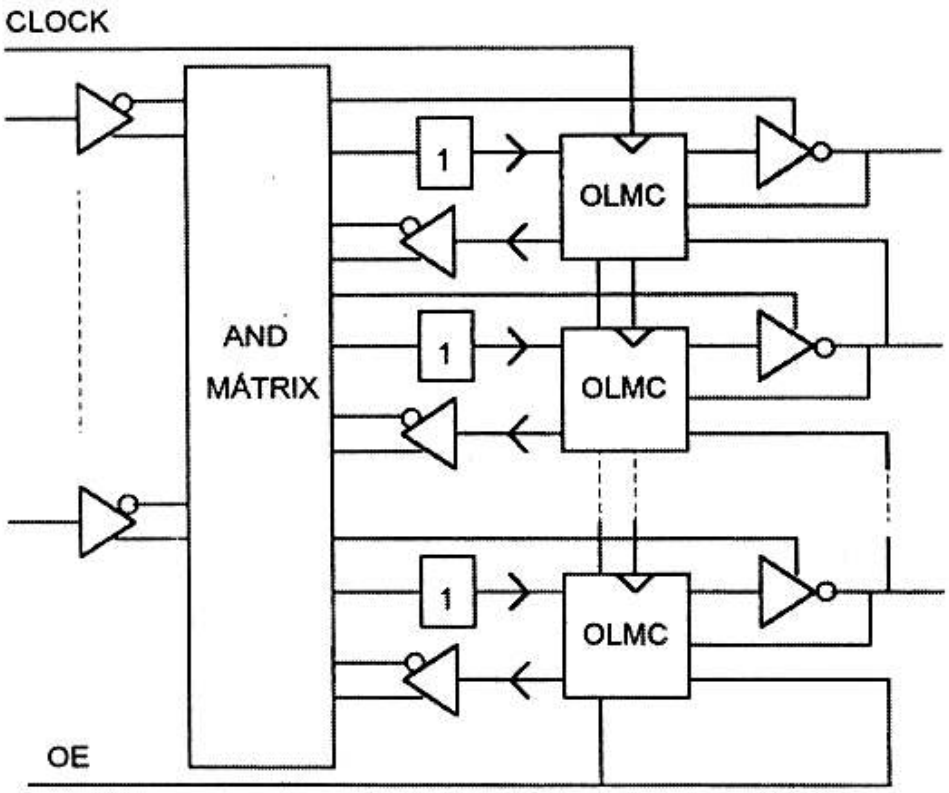
Ez a struktúra a következő konfigurációkra képes:

1. Regiszteres kimenet (8 pterm) aktív alacsony szinttel, kimenet vezérlés OE-vel.
2. Regiszteres kimenet (8 pterm) aktív magas szinttel (EXOR-nál és a kimeneten invertálás), kimenet vezérlés OE-vel.
3. Kombinációs kimenet (7 pterm) aktív alacsony szinttel, kimenet engedélyezés ptermmel.
4. Kombinációs kimenet (7 pterm) aktív magas szinttel, kimenet engedélyezés ptermmel.
5. Kombinációs kimenet (8 pterm) aktív alacsony szinttel, kimenet engedélyezés állandóan.
6. Kombinációs kimenet (8 pterm) aktív magas szinttel, kimenet engedélyezés állandóan.
7. Bemenet.

A GAL16V8 8db függetlenül konfigurálható makrocellát tartalmaz, 10 általános és 2 dedikált bemenettel (CLOCK, OE) rendelkezik.

Sajnos, a GAL16v8 makrocella vezérlésénél spóroltak a konfigurációs tárolóelemekkel, ezért közösítették a multiplexerek címzését, aminek többek között az lett a következménye, hogy regiszteres esetben a kimenet csak az OE-vel vezérelhető, ptermmel nem. Másik kellemetlen tulajdonsága, hogy az invertálási lehetőség a flip-flop bemeneténél van, ezért amikor a bekapcsolási reset a flip-flopokat törli, a kimenet 1-be áll. Ráadásul a flip-flopoknak nincs külön set ill. reset bemenetük. A chip EEPROM cellákban tárolja a konfigurációs információkat. Fontos tudni, hogy a hasonló elnevezésű típusok nem egészen ugyanolyanok.

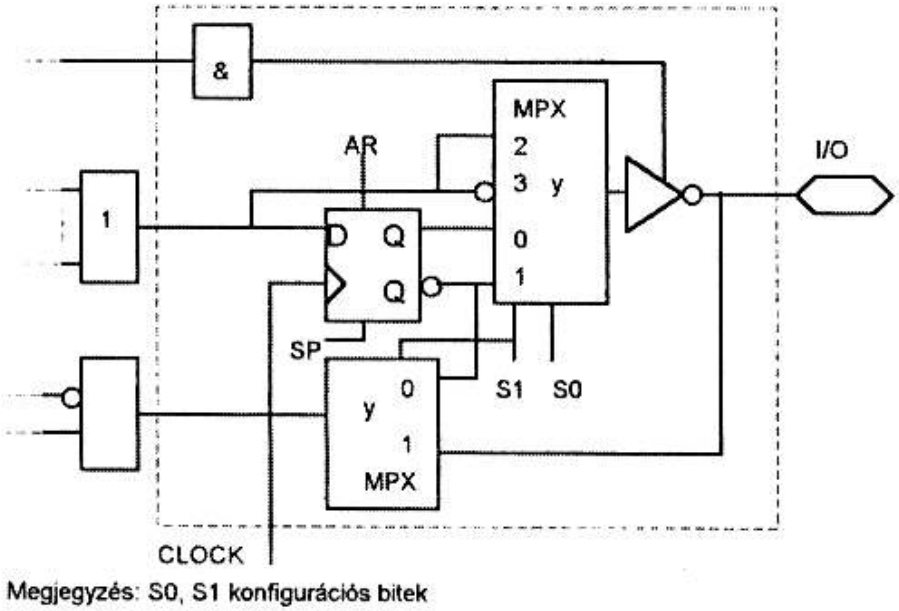
A GAL 16V8 globális felépítését mutatja a 2.11. ábra.



2.11. ábra. A GAL 16V8 blokkvázlata

A makrocellára a katalógusok OLMC (Output Logic MacroCell) rövidítéssel hivatkoznak, mivel elsősorban kimenetként szolgál.

Továbbfejlesztett változatra példa a GAL22V10, ennek makrocellája látható a 2.12. ábrán.



2.12. ábra. A GAL 22V10 makrocellája.

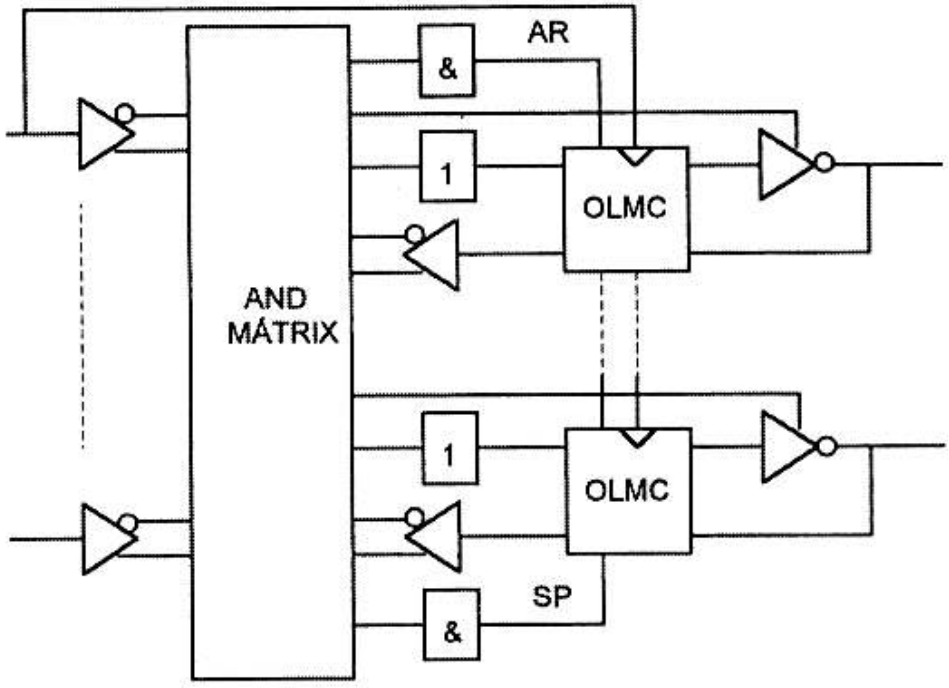
Az ábrán S0, S1 konfigurációs bitek, SP szinkron preset, AR aszinkron reset.

Konfigurációs lehetőségek:

1. Regiszteres kimenet aktív alacsony szinttel.
2. Regiszteres kimenet aktív magas szinttel.
3. Kombinációs kimenet aktív alacsony szinttel.
4. Kombinációs kimenet aktív magas szinttel.

A kimenet engedélyezése mindegyik esetben ptermmel történik, de ez az AND kapu a kimeneti függvény képzésében nem vehet részt.

A 10 D típusú regiszter közösített aszinkron resettel és szinkron presettel rendelkezik. Mindkét bemenet ptermmel vezérelhető. 11 általános bemenete van, ezen kívül az órajel is használható bemenetként. A GAL22V10 globális felépítését mutatja a 2.13. ábra.



Jelölések: AR aszinkron reset
SP szinkron preset

2.13. ábra. A GAL 22V10 blokkvázlata

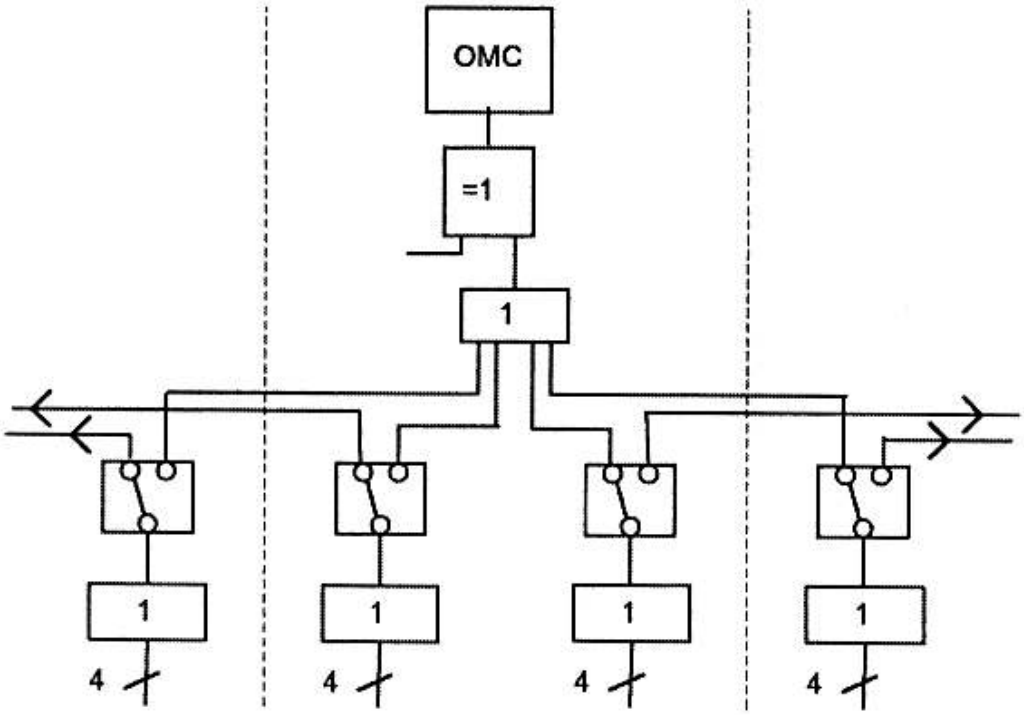
A 16V8 és 22V10 21-21 PAL típust képes emulálni. Széles sebességtartományban léteznek, 25ns-tól 5ns-os késleltetésűig. Az ispGAL 22V10 in system programozható változat, vagyis beforrasztás után is felprogramozható megfelelő hardver kialakítás esetén.

A speciálisan vezérlési célokra tervezett PLD-k (PLS) kivételével, a gyártók a *PAL struktúrát preferálják* a PLA-val szemben, melynek oka valószínűleg az OR mátrix területigénye. Ennek ára, hogy a termek megosztása fix. Ezen a PLA struktúra megkerülésével kétféle módon próbálnak segíteni.

Az egyik módszer, hogy az egyes makrocellákhoz kapcsolódó 2 szintű hálózatban az *AND kapuk számát különbözőre választják*. Így a szükségletekhez

legközelebb állót lehet kijelölni. Ennek a megközelítésnek hátránya, hogy megkötést jelent az adott célra használható lábakra, és így is maradnak kihasználatlan AND kapuk.

A másik módszer, a *pterm allokáció.* Ennek megoldására példa a 2.14. ábrán látható elrendezés.



2.14. ábra. Pterm allokáció

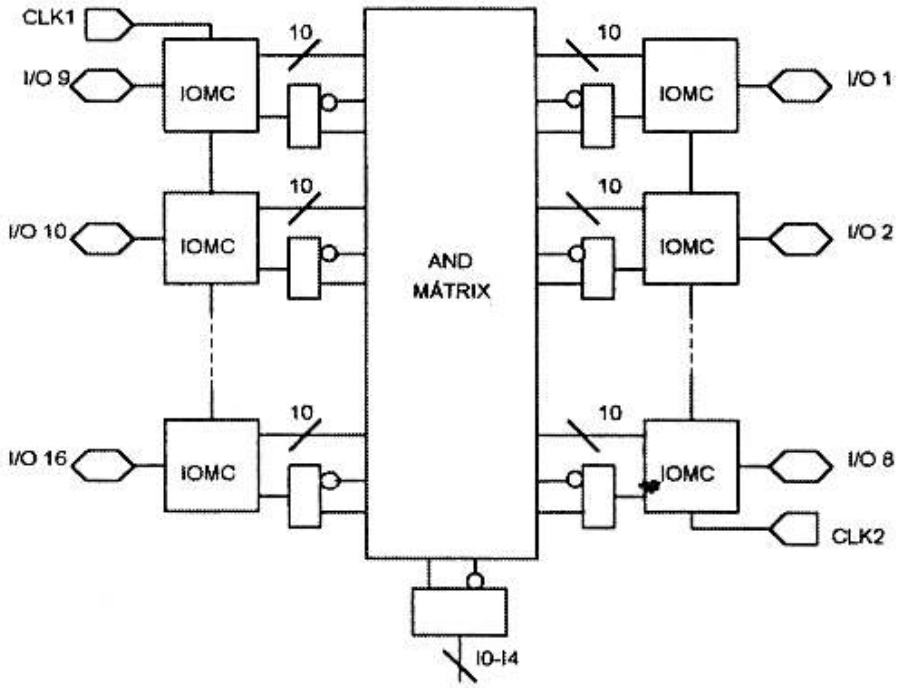
Itt minden makrocellához 2x4 pterm tartozik, azonban az egyik ill. a másik hozzárendelhető a szomszédos makrocellához. Szélsőséges esetben, ha egy makrocellához rendelt lábat bemenetként használnak, a ptermjei a szomszédjaihoz rendelhetők.

A CPLD struktúra megjelenése

A PLD-k között megjelentek olyanok, melyek az átmenetet képviselik az egyszerű PLD-k és a CPLD-k (Complex PLD) között. Ilyen az 5C060 (Intel), EP610 (Altera) EPLD (Erasable Programmable Logic Device). Ezek gyakorlatilag a CPLD-k minden jellegzetességét magukon viselik, csak bonyolultságukban maradnak alul. Az EP610 blokkvázlatát mutatja a 2.15. ábra.

Az IC lábak legtöbbjéhez I/O makrocellát rendeltek. (16 I/O makrocella, 4 dedikált input.) Így nagyon rugalmasan a feladathoz igazítható a struktúra.

Makrocellánként 1 tárolóelem, mely D, T, SR és JK flip-flopként konfigurálható és ptermmel vezérelt aszinkron resettel rendelkezik. SR és JK beállítása esetén a két vezérlőfüggvény között tetszőlegesen megosztható a rendelkezésre álló 8 pterm. A vezérlőfüggvények invertálhatóak is.



2.15. ábra. Az EP610 blokkvázlata

Két független órajel bemenetet alakítottak ki, melyek 8-8 makrocellához vannak rendelve, így akár 2 független szinkron sorrendi hálózat is megvalósítható.

Ennél a típusnál már megjelenik az *aszinkron órajel* is. Ez azt jelenti, hogy minden egyes flip-flopnál lehetőség van arra, hogy az órajelet egy-egy ptermtől kapja. Az aszinkron órajelnél rövidebb a setup time, de hosszabb az órajel-kimenet kesleltetés. Így ezt is figyelembe véve kell dönteni, hogy melyiket alkalmazzuk.

Kisfogyasztású üzemmód (zero power)

Az újabb PLD egyre több típusánál választhat a felhasználó programozáskor, hogy gyors áramkört akar, vagy kisfogyasztásút (TURBO bit). Az utóbbi esetben, ha az eszköz bármely bemenetén 100ns ideig nincs változás, akkor átkapcsol kisfogyasztású (*stand-by, zero stand-by*) üzemmódba. Ez azonban 25ns-al megnöveli a késleltetési időt, a normál üzemmódhoz képest.

2.4. CPLD eszközök

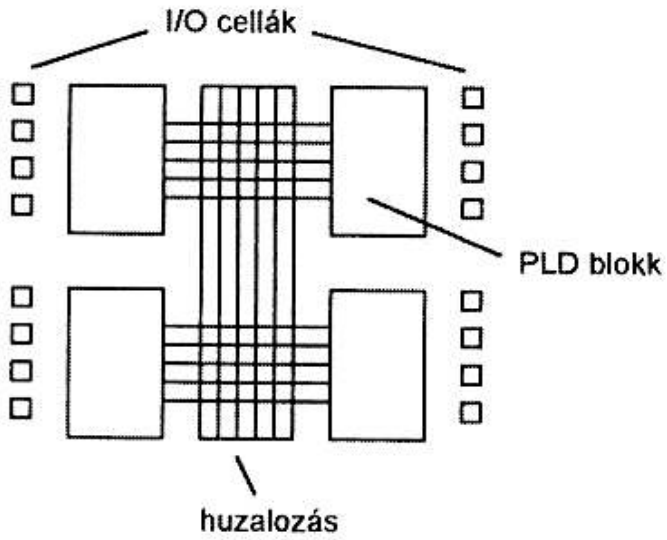
Az egyszerű PLD eszközök főként kisebb vezérlési feladatokra, SSI és MSI elemek kiváltására alkalmasak, mivel az egy IC-ben elhelyezhető logika mérete nagyon korlátozott. Bonyolultabb feladatokat csak több PLD segítségével lehetne megoldani. Ekkor a feladatot egy-egy IC-vel megvalósítható méretű részekre kell particionálni, ami

nem könnyű feladat. Az egységek közötti I/O csatlakozások megnövelik a késleltetési időt, így csökken az áramkör sebessége. A sok IC által elfoglalt NYÁK terület ára sem utolsó szempont. Mindezen okok miatt a bonyolultabb feladatok megvalósítására alkalmasabbak a CPLD (Complex PLD) eszközök, melyek az egyszerű PLD-k előnyeinek megtartásával (sebesség, egyszerű struktúra, egyszerű tervező rendszer, viszonylag olcsó ár), nagyobb bonyolultságú feladatok megvalósítására is alkalmasak.

A CPLD-k általános jellemzői:

- több PLD-t tartalmaznak egy tokban,
- blokkos felépítésűek, a blokkok nagyjából önálló PLD moduloknak tekinthetők,
- a makrocellák 1-3 regisztert tartalmaznak, az I/O cellák többnyire regisztert is tartalmaznak, a bejövő jel mintavételezésére,
- nagy órajel frekvencia érhető el,
- az időzítések az egyszerű PLD-khez hasonlóan könnyen számíthatók (nem így az FPGA-knál),
- a CPLD tervező rendszerek egyszerűek, tipikusan ugyanazzal a rendszerrel tervezhető a CPLD, mint a PLD, erre példa az ABEL rendszer, de vannak dedikált rendszerek is (Lattice pDS, SYNARIO),
- a legtöbb tipust lehetőség van maszkprogramozott áramkörre konvertálni (nagy sorozatnál olcsóbb).

Egy tipikus CPLD struktúrát mutat a 2.16. ábra.



2.16. ábra. CPLD struktúra

2.4.1. Az ATV 5000 (ATMEL)

Ez egy 68 lábú, közepes bonyolultságú CPLD eszköz, melyben EPROM cellák tárolják a konfigurációs információkat.

Jellemzői:

- a chip 4 db azonos kialakítású PLD-t tartalmaz egy tokban,
- egy-egy negyed 13 I/O cellát és 6 eltemetett cellát tartalmaz,
- minden I/O cellához 3 AND-OR hálózat, 2 flip-flop (az egyik eltemetett) és egy I/O buffer tartozik, a kimenetre jutó jel meginvertálható,
- minden I/O cellához egy bemeneti latch is tartozik (de természetesen ez kikerülhető), melyeknek negyedenként közös órajele van (egy dedikált bemenetről),
- az egyes I/O cellákhoz tartozó AND-OR hálózatok maximum 4 ill. 5 termet tartalmazhatnak,
- a flip-flopok konfigurálhatóan D vagy T típusúak lehetnek, aszinkron reset és preset bemenettel, melyet egy-egy pterm állíthat elő,
- az egyes negyedek önálló dedikált órajellel rendelkeznek, a flip-flopok órajele vagy ez (gyors szinkron hálózathoz, max. 33-50 MHz), vagy egy pterm által előállított egyedi órajel (max. 25-40 MHz),
- a kimeneti három állapotú buffert szintén ptermmel lehet engedélyezni,
- a chipben kétszintű huzalozási hierarchiát alakítottak ki, egy univerzális buszt, melynek jelei mindegyik negyedhez eljutnak és négy regionális buszt, melynek jelei csak a negyeden belül hozzáférhetők,
- az univerzális buszt az összes I/O cella kimenete meghajthatja,
- a regionális buszokat a negyeden belüli I/O cellák regiszter kimenetei képesek meghajtani és 8 csak bemenetként dedikált láb természetesen mindegyik ponáltan és negáltan,
- az AND-OR hálózatok ptermjeinek egy része (3 pterm) a regionális buszról, másik része (a konfigurációtól függően megmaradt 1 vagy 2 pterm) pedig az univerzális buszról kapja a jeleket,
- az eltemetett cellák egy regisztert és egy AND-OR hálózatot tartalmaznak, melynek 4 ptermje a regionális, egy pedig az univerzális buszra csatlakozik,
- viszonylag nagy a késleltetése 25-35ns.

Sebesség szempontjából az az optimális, ha egy-egy negyedben a szekunder változók önfüggőek (vagy önálló szinkron sorrendi hálózatot valósítanak meg, vagy ortogonális HT partíciók blokkjait). Ugyanis egy-egy negyed szekunder változói (regiszter kimenetei) csak a kimeneti logikán (EXOR és kimeneti buffer) keresztül juthatnak el a többi negyedhez, ami járulékos késleltetést jelent.

2.4.2. A Lattice ispLSI 1000 család

A Lattice pLSI és ispLSI 1000 (1016, 1024, 1032, 1048) családja az előbbinél rugalmasabb felépítésű.

Jellemzői:

- a logika alapegysége a Generic Logic Block (GLB),
- minden GLB 4db 18 bemenetű programozható AND-OR-XOR logikát tartalmaz, (20 pterm, az OR kapuk bemenet száma 4, 4, 5, 7) és 4 kombinációsként ill. regisztersként konfigurálható kimenetet, a regiszterek D, J-K és T típusúak lehetnek,
- az OR kapuk kimenetei és a GLB kimenetei tetszőlegesen összerendelhetők a PTSA (Product Term Sharing Array) segítségével, a PTSA a függvények bővítését is lehetővé teszi, vagyis az OR kapuk kimenetei VAGY kapcsolatba hozhatók,
- a PTSA-val gyorsabb logika is kialakítható, de ekkor csak 4 ptermje lehet minden egyes függvénynek és az OR kapuk GLB-hez rendelése előre definiált, továbbá a flip-flopok csak D típusúak lehetnek,
- a bemenetek közül 16 egy közös huzalozási mezőből (Global Routing Pool, GRP) jön, 2 pedig dedikált bemenetekről,
- az összes GLB kimenetei rácsatlakoznak a GRP-re, így bármely GLB kimenete bármely GLB bemenetét meghajthatja,
- a lábak többségéhez I/O cellák vannak rendelve, melyek egyenként programozhatók egyszerű, regiszteres, latchelt bemenetként, kimenetként, vagy kétirányú I/O-ként és ezek is rácsatlakozhatnak a GRP-re; az I/O cellák kimeneti polaritása programozható,
- az I/O cellák és a GLB-k ún. megablokkokba vannak szervezve, 16 I/O cella, 2 dedikált bemenet és 8 GLB alkot egy megablokkot, melyen belül az I/O cellák és a GLB-k egy kimeneti huzalozási mezőn (Output Routing Pool) keresztül rugalmasan egymáshoz rendelhetők, a GLB egy kimenete 4 rögzített I/O cella valamelyikéhez rendelhető,
- az I/O cellák kimenet engedélyezése (OE) a megablokkon belül közös, az engedélyező jel forrása a 8 GLB egyike lehet, de ez a jel csak a három állapotú kimenetként konfigurált I/O-k buffereire hatásos; ha egy I/O láb nem használt, aktív felhúzóellenállás biztosítja fix logikai szintjét,
- az áramkörben órajel elosztó áramkör (Clock Distribution Network) is található, az órajelek forrása 4 dedikált bemenet (Y0-Y3) és egy dedikált GLB kimenetei lehetnek; az órajel elosztó CLK0-CLK2 kimenetei a GLB-k regisztereinek órajele, míg az IOCLK0-IOCLK1 az I/O cellák regisztereinek órajele lehet (cellánként választhatóan); ezen kívül minden GLB-nek saját ptermmel előállított órajele lehet,
- a GLB-k és I/O cellák regiszterei törölhetők, az előbbi a dedikált RESET jellel és ptermmel előállított reset jellel (ezek VAGY kapcsolatban vannak), az utóbbi csak a dedikált RESET jellel,
- a GRP egy-egy jelének késleltetése az azt terhelő GLB-k számától is függ (1-16 GLB terhelés 1.5-4.5 ns késleltetést okoz).

A család tagjainak jellemzőit mutatja a 2.1. táblázat.

Típus	1016	1024	1032	1048
sűrűség (PLD kapu)	2000	4000	6000	8000
sebesség MHz	110	90	90	80
késleltetés ns	10	12	12	15
GLB-k	16	24	32	48
Regiszterek	96	144	192	288
Bemenetek + I/O-k	36	54	72	106
Láb	44-pin	68	84/100-pin	120-pin

2.1. táblázat.

Az 1000-es család tovább fejlesztett változatai a 2000 és 3000-es családok, melyek közül az utóbbi IEEE 1149.1 Boundary Scan interfésszel is rendelkezik. Ez a beépítés utáni tesztelést segítő standard szinkron soros interface.

Mindhárom családnak létezik az ispLSI (in system programmable LSI) változata is. Ezek olyan soros interfésszel rendelkeznek, amelyen keresztül a NYÁK-ba ültetés után is felprogramozhatók. Felprogramozás után az interfészhez rendelt lábak egy része bemenetként használható.

A rendszeren belüli programozhatóságnak több előnye is van:

- nincs szükség drága programozó berendezésre (egy PC printer portjára csatlakoztatható egységgel is elvégezhető a felprogramozás),
- az áramkör beforrasztás után is fel-, ill. átprogramozható,
- működés közben átkonfigurálható rendszer hozható létre.

Az interfész 5 jelet tartalmaz:

$\overline{\text{ispEN}}$ programozás (edit mód) engedélyezés

$\overline{\text{ispEN}}$=1 esetén a többi interfész láb bemenetként használható, $\overline{\text{ispEN}}$=0 esetén az I/O lábak harmadik állapotba kerülnek és engedélyezett az interfész működése. Az eszközt programozás alatt resetelni kell.

SDI soros adatbemenet és a belső állapotgép programozása
MODE állapotgép programozása
SDO soros adatkimenet
SCLK soros órajel

Az interfész 3 állapottal jellemezhető, melyek között az állapotváltást a MODE és SDI jelek vezérlik és órajel felfutó él hatására történik.

Idle State (Normal Operation): Ez az alapállapot. Ebben az állapotban órajel hatására az eszköz azonosító kódja beíródik az interfész shiftregiszterébe és MODE=0 alatt adott 7 órajelre kiolvasható a 8 bit az SDO vonalon keresztül.

Shift State (Load Commands): Ebbe az állapotba MODE, SDI=11 hatására kerül az Idle állapotból. Ebben az állapotban egy 5 bites parancsot lehet beshifteltetni az interfészbe, MODE, SDI=0X alatt. 15 különféle parancs létezik, melyek az eszköz különféle részeinek törlését, a beprogramozandó vagy kiolvasandó adat címének (mátrix sora) és az adat (a kiválasztott sor tartalma) megadását GLB és I/O regiszterek feltöltését (diagnosztika), az SDIN és SDOUT közvetlen összekötést (sorosan kapcsolt eszközök esetén az így konfigurált kimarad) írhatják elő. MODE, SDI=10 esetén visszakerül Idle állapotba.

Execute State (Execute Command): Ebbe az állapotba az előzőből MODE, SDI=11 esetén kerül. MODE, SDI=0X alatt adott megfelelő számú órajel hatására végrehajtja a legutóbb megadott parancsot. MODE, SDI=10 hatására az Idle, MODE, SDI=11 hatására pedig Shift State állapotba kerül.

Az interfész lehetővé teszi, hogy több eszközt is ugyanazon az 5 vonalas interfészen keresztül felprogramozzunk. Ilyenkor az SDO-SDI-k sorba kapcsolódnak, a többi jelet pedig párhuzamosan kapják az eszközök. Az utolsó SDO-ja visszacsatolódik a programozó eszközbe.

2.5. Tervezési szempontok PLD-knél

Az eddig ismertetett eszközöket főként SSI és MSI elemek kiváltására, egyszerű vezérlések megvalósítására, egyedi specifikációjú funkcionális elem létrehozására szokás alkalmazni. A megoldható feladatok méretét PLD-k esetén erősen korlátozza a regiszterek, ill. bemenetek viszonylag kis száma. CPLD-kkel már több MSI bonyolultságú feladat is megoldható.

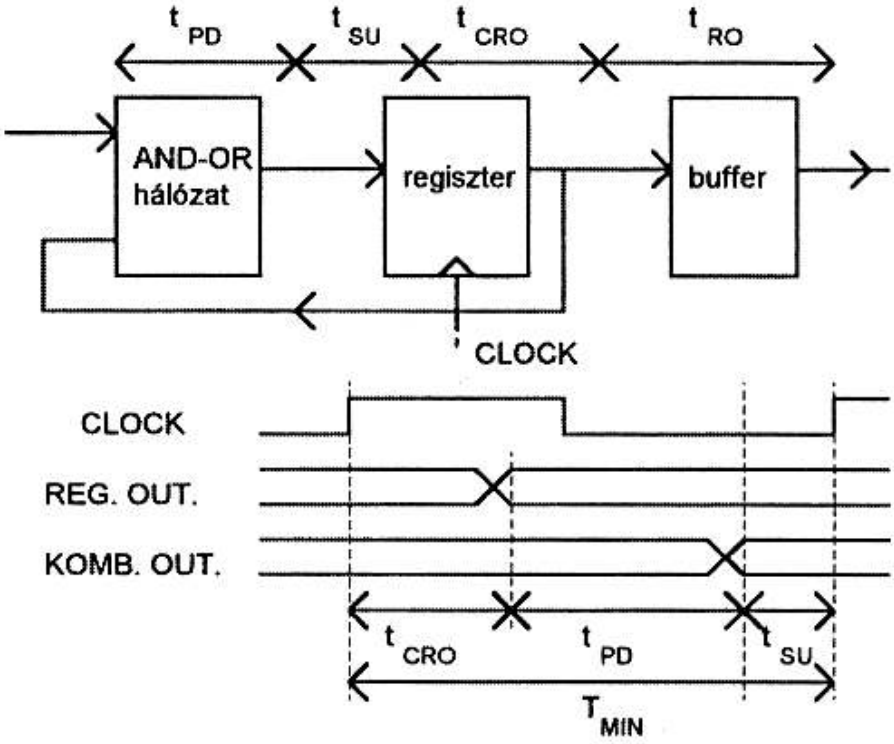
A feladat megfogalmazását követően első lépés a szóba jöhető eszközök kiválasztása. Itt elsődleges korlát a rendelkezésre álló fejlesztő rendszer és programozó készülék, hiszen csak az ezek által támogatott eszközöket használhatjuk. Ezután a rendelkezésre álló eszközökből az alábbi szempontok figyelembe vételével kell választanunk:

- bemeneti változók száma: szükséges bemenetek száma,
- állapotszám és kimeneti jelek száma alapján: a szükséges flip-flopok ill. makrocellák száma,
- szinkron vagy aszinkron feladat: szinkron, ill. aszinkron órajel lehetőségek,
- kezdeti állapot beállítás: flip-flopok preset ill. clear lehetősége,
- feladathoz legjobban megfelelő flip-flop típus (számlálónál pl. T ill. JK): flip-flop konfigurációs lehetőségek,
- sebességigény: megfelelő maximális órajel frekvencia, setup time, késleltetési idő.

2.5.1. Időzítési modell

A tervezésre különösen figyelni kell a nagy sebességű alkalmazásoknál, mivel ekkor a PLD-t a sebességhatár közelében kell működtetni. A PLD-k esetében viszonylag egyszerű a szinkron sorrendi hálózatként alkalmazott eszköz határfrekvenciáját kiszámítani.

Ehhez a 2.17. ábra szerinti időzítési modellt használjuk. (t_{PD}: kombinációs hálózat késleltetése, t_{SU}: regiszter setup time-ja, t_{CRO}: reg. kimenet késleltetése az órajelhez képest, t_{RO}: buffer késleltetése)



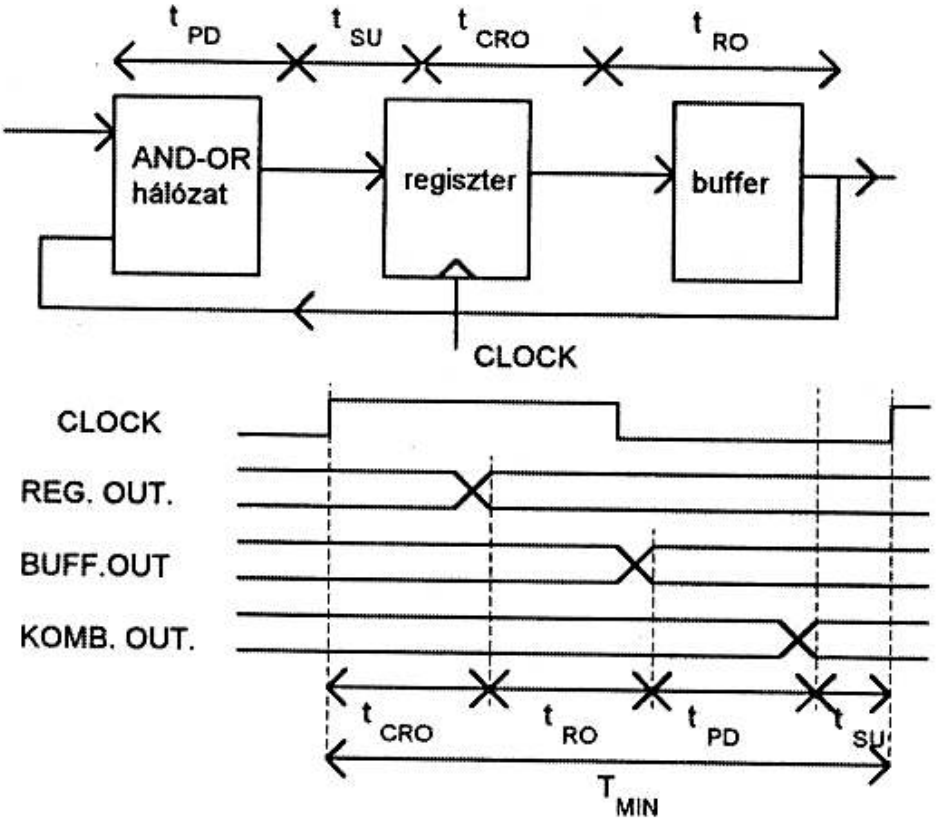
2.17. ábra. PLD időzítési modellje belső visszacsatolásnál

A maximális órajel frekvencia belső visszacsatolás esetén:

$$f_{CNT} = \frac{1}{t_{CRO} + t_{PD} + t_{SU}} \qquad (2.1)$$

Külső visszacsatolást alkalmazva ez lecsökken, a regiszter és kimenet közötti áramköri rész késleltetése miatt (2.18. ábra).
A maximális órajel frekvencia külső visszacsatolás esetén (kimeneti lábról kötjük vissza a jelet egy bemeneti lábra):

$$f_{CNT} = \frac{1}{t_{CRO} + t_{RO} + t_{PD} + t_{SU}} \qquad (2.2.)$$

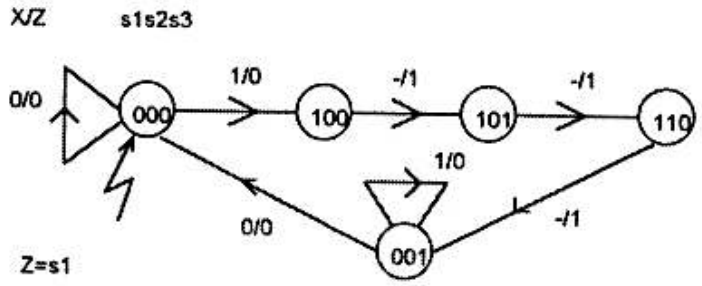


2.18. ábra. PLD időzítési modellje külső visszacsatolásnál

2.5.2. Állapotkódolás PLD-k esetén

A regiszteres PLD-k esetében a kimenet többnyire közvetlenül egy regiszterről jut a kimeneti meghajtóra, ill. a Mealy modell megvalósításához sokszor kevés a kombinációs kimenetek száma. Ilyenkor kedvezőbb, ha a hálózat kódolását a kimenet alapján végezzük, vagyis valamely kimenet(ek) egyben szekunder változó(k) is. Ekkor az állapotkódolás sokszor csak extra szekunder változók felvételével oldható meg, az állapotgráfból adódó megkötöttségek miatt.

A 2.19. ábra egy digitális monostabil multivibrátor állapotgráfját mutatja, melyet kimenet alapján kódoltunk.



2.19. ábra. Kimenet szerint kódolt digitális monostabil multivibrátor

Az automata bekapcsoláskor a 000 kódú állapotba kerül, ekkor 0 a kimenete. Az áramkört a bemenetére adott 1 szint indítja. Ezután 3 órajelig 1-et ad a kimenetén. Ezután arra vár, hogy megszüntessék az indító jelet, ekkor kerül újra alapállapotba.

2.5.3. Termek számának csökkentési lehetőségei

Sokszor előfordul, hogy a megoldandó feladat a rendelkezésre álló PLD-vel megvalósítható legnagyobb méret környékén mozog. Gyakori eset, hogy az igényelt termek száma egy adott megvalósításnál néhánnyal több, mint ami rendelkezésre áll. Ilyenkor az eszköztől függően több lehetőségünk lehet.

Egyik, már taglalt lehetőség, hogy a kritikus függvény negáltját valósítjuk meg, ha ez kevesebb termet igényel, s utána invertálunk, ha ezt a PLD lehetővé teszi.

Másik lehetőség, hogy a kritikus vezérlőfüggvényt két, esetleg több függvényből állítjuk elő (többszintű hálózatként), ha van még szabad kombinációs hálózatrész az eszközben. Ekkor azonban számolni kell a maximális órajelfrekvencia ill. késleltetési idő növekedésével.

Harmadik lehetőség (sorrendi hálózat esetén), hogy ha az eszközben konfigurálható a flip-flop típusa, akkor másikat választva újratervezzük az áramkört, reménykedve, hogy csökken a termek száma.

Termek számának csökkentési lehetőségei számlálók esetén

A termek számának csökkentésére jó példa a számlálók esete. Ha egy bináris számlálót D flip-flopokkal és JK flip-flopokkal is megtervezünk, ahogy növeljük a számláló modulusát annál nagyobb különbség mutatkozik a JK flip-flopos áramkör javára a termek számában (a T flip-floppal is hasonló a helyzet). Ez azért van, mert a D flip-flop esetén egy számláló bit 1 állapotának fennállási ideje alatt 1-et kell biztosítani a bemenetén, a megfelelő termekkel. A nagyobb helyiértékek felé ezt egyre több egymást követő állapot alatt kell megtenni. JK ill. T flip-flop esetén csak a 0-1 és 1-0 állapotátmenetek esetén szükséges term. Tehát számlálók esetén lehetőleg JK vagy T flip-flopot válasszunk.

Hasonló egyszerűsödéshez vezet, ha az egyes eszközökben a rendelkezésre álló EXOR kaput is tartalmazó kombinációs hálózatot kihasználjuk. (Ezt a fejlesztő rendszerek egy része automatikusan megteszi.)

Nem bináris esetben a termek száma többnyire nő a bináris számlálókhoz képest. Itt általában jobban járunk, ha bináris számlálót tervezünk, majd a modulusát szinkron törlést alkalmazva csökkentjük (ha az eszközben van ilyen lehetőség). Ez egyben megoldást jelent az illegális állapotok kezelésére is, hiszen valahány órajel után biztosan legális állapotba kerül a számláló.

Engedélyezhető számlálók esetén szintén előnyt jelent a JK és T flip-flop a D-hez képest. A tartás üzemmódban JK=T=0, ami az összes vezérlőfüggvényben minden AND kapunál 1 bemenetet foglal el. D esetén egy teljes kapu szükséges a D=Q biztosításához, és az összes többi AND kapun 1 bemenet, mellyel a többi term 0 értéke biztosítható.

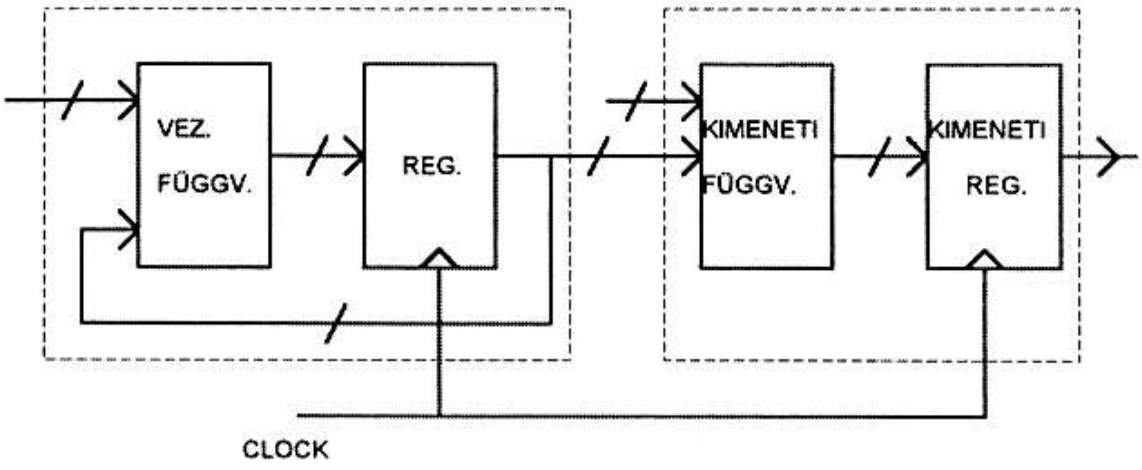
Gyors, nagy méretű számlálókat több eszköz kaszkádosításával lehet létrehozni, a számlálókkal foglalkozó fejezetben már taglalt módon, a legkisebb helyiérték carryjének előrecsatolásával.

Az imént elmondottak természetesen egyben szempontot is adnak arra, hogy milyen eszközt érdemes választani, ha számlálót tervezünk.

2.5.4. Tervezési szempontok PLD-s vezérlők esetén

Ciklikus vezérlési szekvenciák előállítására nagyon alkalmasak a Gray-kódú számlálók. Ezeknél az egymást követő állapotok kódja 1 Hamming-távolságú, így az állapotait kódoló kombinációs hálózat *hazárdmentesíthető*. Legegyszerűbb ilyen, a shiftregisztereknél már említett Johnson számláló. Hátránya, hogy nem használja ki a flip-flopok száma alapján lehetséges állapotokat. Előnye, hogy a vezérlőfüggvények nagyon egyszerűek és könnyű dekódolni az állapotait. Természetesen a lehetséges állapotokat teljesen kihasználó Gray kódú számláló is tervezhető.

Hazárdmentes kimenetet produkáló vezérlőt két egyszerű PLD sorbakapcsolásával a 2.20. ábrán látható módon is kialakíthatunk. Ez tulajdonképpen egy olyan More modell szerint működő hálózat, amelyet egy Mealy modell szerint működőből a kimenetre kapcsolt regiszterrel alakítottunk ki. Az ilyen típusú megvalósításnál azonban figyelembe kell venni, hogy a kimenet egy órajelet késik a szokásos Mealy-modellhez képest. A metastabilitás esélyének csökkentésére és egyéb okokból (lsd. később) a bemenetre is regisztert illik tenni, aszinkron bemenő jelek esetén.



2.20. ábra. Hazárdmentes kimenetű SSH

Az egyszerű PLD-s vezérlőkkel elvileg *megvalósítható állapotszám* látszólag elég nagy, ha a 8-12 regiszterből indulunk ki. A gyakorlatban ez a szám maximum kb. *30-40*. Ennek az az oka, hogy egyfelől a regiszterek egy részét általában kimenetként használjuk (kimenet szerinti állapotkódolás, hazárdmentesítés), másfelől a rendelkezésre álló termek száma az állapotszámot is jelentősen korlátozza.

A visszacsatolt kombinációs hálózatból felépített aszinkron hálózatot lehetőleg kerülni kell, mert nehezen kézbetartható a működése és tesztelése is nagyon nehéz. Ahol aszinkron működésre van szükség, ott inkább *gyors órajelű szinkron hálózatot*

alkalmazzunk, esetleg önálló (aszinkron) órajellel és set-reset bemenettel rendelkező flip-flopos eszközöket.

A gyors órajelű szinkron hálózat tkp. az aszinkron hálózat állapotgráfjának szinkron sorrendi hálózattal történő megvalósítása. Ha egy így kialakított szinkron sorrendi hálózatot elég gyors órajellel működtetünk (a minimális órajel frekvenciát a feladat által igényelt válaszidő alapján lehet meghatározni), akkor az kis késleltetés kivételével hasonlóan fog működni, mint a megfelelő aszinkron hálózat, viszont elkerültük az aszinkron hálózat tervezésének problémáit (kritikus versenyhelyzet, lényeges hazárd).

2.6. PLD tervezési környezet

A PLD tervező rendszerek a tervezést és szimulációt ill. tesztelést segítő programrendszerből és a programozó készülékből állnak. A tervezés folyamatát a 2.21. ábrán látható folyamatábra mutatja.

A terv leírása valamely magas szintű nyelven történik. Ilyenek az ABEL, VHDL, AHDL, PALASM stb.

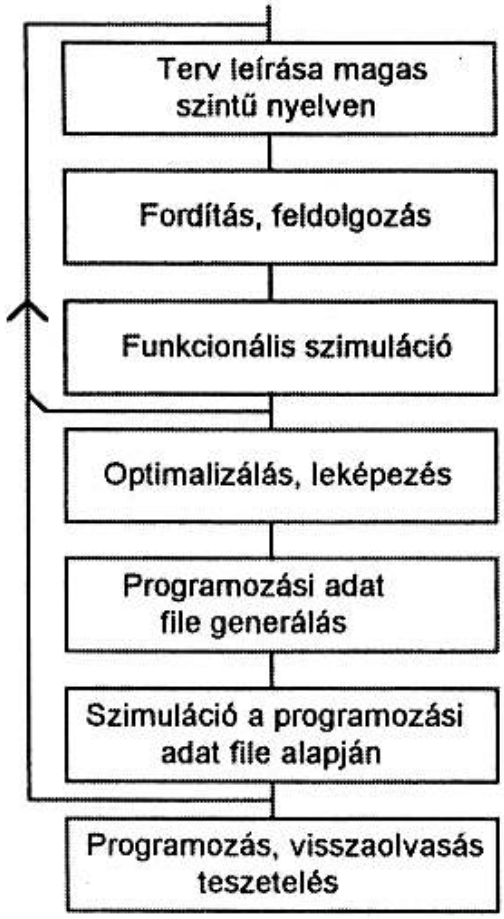
A fordítás előtt a szöveges leírást szintaktikailag ellenőrzi a fordító, kifejti a logikai egyenleteket, igazságtáblákat, állapotgép leírásokat, és némi egyszerűsítést is elvégez.

A funkcionális szimuláció során a rendszer ellenőrzi, hogy a lefordított leírás alapján a tervben szereplő tesztvektorokra az ott megadott válaszok generálódnak-e.

Az optimalizálás során a program minimalizálja a logikai függvényeket, az eszköz architektúrájának figyelembevételével.

A programozási adatokat valamely standard file formátumban állítják elő a rendszerek. Az egyik legelterjedtebb file formátum a JEDEC. A programozási adatokat tartalmazó file és az eszköz modelljének ismeretében a rendszer szimulálja, hogy ha az adott bitmintát beégetnék az eszközbe, a megadott tesztvektorokra a megadott válaszok generálódnának-e.

A programozás előtt a programozó készülék a chipben levő elektronikus azonosító (Electronic Signature, Factory ID) alapján ellenőrzi, hogy valóban a megadott típusú és gyártójú eszköz van-e a programozó foglalatába illesztve. Ezután teszteli, hogy üres-e az eszköz. Ha nem üres, és EEPROM-os az IC, akkor kérésre kitörli, majd beprogramozza a progamozási adatokat tartalmazó file-ban megadott bitmintát. Ezek után egyes égetők az IC-n is ellenőrzik a tesztvektorokat.



2.21. ábra. A PLD tervezés folyamata

A PLD tervezési környezetek csoportosítása

a. Gyártó specifikus környezetek (pl. Lattice pDS, Intel PLD shell , SYNARIO):

- csak egy gyártó áramköreit ismerik,
- az eszközök speciális tulajdonságait nagyon jól képesek kihasználni,
- az új eszközöket támogató szoftver-hardver gyorsan megjelenik.

b. Univerzális fejlesztői környezetek (pl. DATA I/O ABEL):

- sok gyártó sok eszközét ismerik, nagy eszközkönyvtárral rendelkeznek,
- egészen speciális tulajdonságokat nem támogatnak.

c. CAD tervező rendszer PLD tervező része (pl. XABEL):

- adott környezethez implementált univerzális PLD tervező eszköz,
- csak a CAD rendszerrel együtt hozzáférhető (drága).

Az ABEL HDL nyelv használatát, ill. annak valamely változatát nagyon sok tervező rendszer támogatja. Ez az oka, hogy a függelékben konkrétan bemutatjuk és mintapéldával is illusztráljuk az ABEL HDL nyelvet.

2.7. FPGA eszközök

Az FPGA-k architektúrája az MPGA-kéhoz hasonlít. A chip felületen többnyire egyenletesen vannak elhelyezve a konfigurálható logikai blokkok, és azok összekötését lehetővé tevő, hierarchikus huzalozási erőforrások.

Általános jellemzőik:

- A nagy alkatrészsűrűség és programozhatóság miatt prototípus tervezésére és kis sorozatú nagy bonyolultságú speciális logikák megvalósítására a legalkalmasabb eszközök.
- A PLD-khez képest egyszerűbbek és kisebbek a logikai cellák, így azokhoz hasonló bonyolultságú logika csak többszintű hálózattal építhető fel.
- A minőségi paramétereik (jelterjedési idők, maximális órajel frekvencia, órajel csúszás) megvalósítás függőek, így nagymértékben függnek a tervező rendszer minőségétől is.
- A hagyományos gate array-khez képest alacsonyabb a kapusűrűségük, lassabbak, és nagy sorozat esetén magasabb az áruk.

Programozás szempontjából két technológia terjedt el.

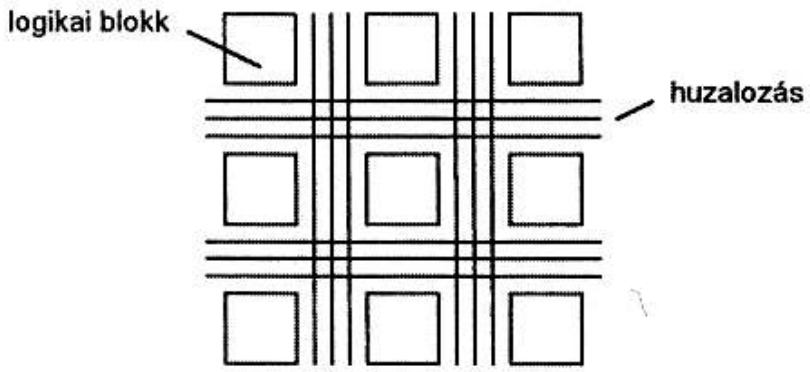
- Az antifuse-os (egyszer programozható) így maradandó a konfiguráció és jóval kisebb chip területet igényel, mint az SRAM.
- SRAM-os, a konfiguráció bármikor megváltoztatható, de kikapcsoláskor elvész. A konfiguráció megőrzése külön tárolót igényel.

2.7.1. Az FPGA-k elrendezései

Az FPGA-kat geometriai elrendezésük alapján négy csoportba osztják. Ezeket és jellemzőiket ismertetjük röviden, az alábbiakban.

Szimmetrikus tömb

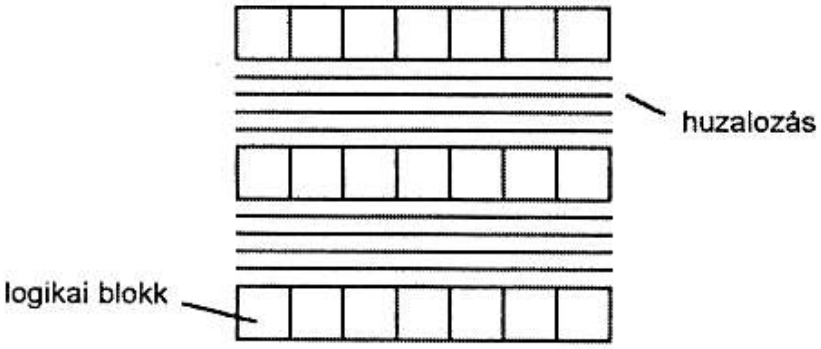
A logikai blokkok mátrix elrendezésűek (2.22. ábra). A logikai blokkokat horizontálisan és vertikálisan elhelyezett huzalozással lehet összekötni. (Pl. XILINX FPGA-k.)



2.22. ábra. Szimmetrikus tömb

Aszimmetrikus vagy sor bázisú

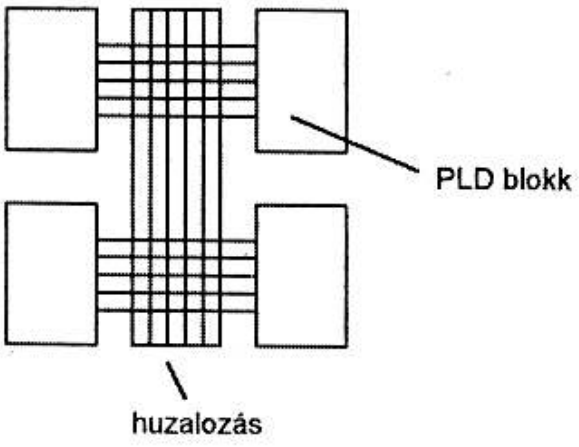
A logikai blokkok sorokba vannak rendezve, közöttük horizontális és vertikális huzalozás található (2.23. ábra).



2.23. ábra. Aszimmetrikus, vagy sor bázisú

Hierarchikus PLD (egyes szakkönyvek a CPLD-ket is az FPGA-khoz sorolják)

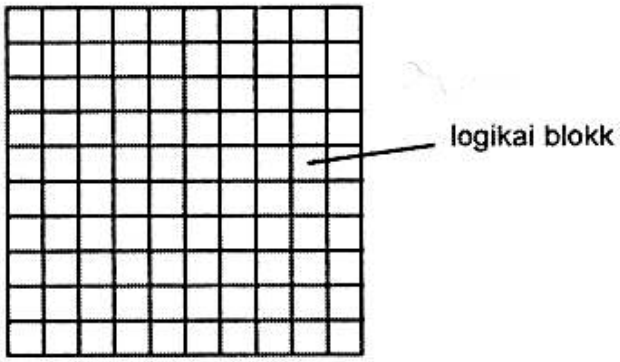
A PLD jellegű logikai blokkokat globális huzalozási mező, a PLD blokkon belüli logikát pedig lokális huzalozás kapcsolja össze (2.24. ábra).



2.24. ábra. Hierarchikus PLD (CPLD)

Egyszintű vagy Sea of gate típusú

Az elemi logikai erőforrások (tranzisztorok, kapuk) szinte egyenletesen helyezkednek el a chip felületén, közöttük minimális huzalozás található (2.25. ábra).



2.25. ábra. Sea of gate

Logikai erőforrások

Az FPGA-k logikai erőforrásai általában azonos felépítésűek, a felületen viszonylag egyenletesen elosztva. Az egyes családok eltérő komplexitású blokkokból építkeznek. A legkisebb bonyolultság a tranzisztor szint, a legnagyobb pedig a PLD-k konfigurálható logikai blokkjának feleltethető meg.

Alap építőelem

Az FPGA-k alap építőelemei többnyire a következők:
- tranzisztor (tranzisztor tömbök),
- NAND kapu,
- multiplexer logika,
- memória táblázat (Look Up Table).

Huzalozási erőforrások

Az FPGA-k huzalozása általában valamilyen hierarchikus elrendezést követ:
- közvetlen huzalozás a szomszédos kapcsolatok létrehozására (a logikai blokk jobb, bal, alsó, felső szomszédjával való összekötések megvalósítására),
- huzalozás a lokális ill. regionális kapcsolatok létrehozására (a modulba csoportosított logikai blokkok közötti kapcsolatok létrehozására),
- globális huzalozási erőforrások általános célra (főként kis megengedhető késleltetésű jelekhez), vagy speciális célra órajelhez, nagyobb áramot elviselni képes vonalak a nagy meghajtóképességet igénylő kimenetekhez.

I/O cellák

Az FPGA-k bemeneteire I/O cellákon keresztül kerül a bemenő jel, ill. ezen keresztül jut ki a kimenő jel. Az I/O cellák különféle tulajdonságai programozhatók:

- választhatóan direkt bemenetek, tárolt (regiszter vagy latch) bemenetek,
- sok esetben jelszint választási lehetőség is van (TTL kompatibilis, CMOS kompatibilis),
- programozhatóan direkt kimenetek, regiszteres kimenetek, három állapotú, open drain-es kimenet,
- kimeneti jelszint választási lehetőség,
- programozható slew rate korlátozási lehetőség.

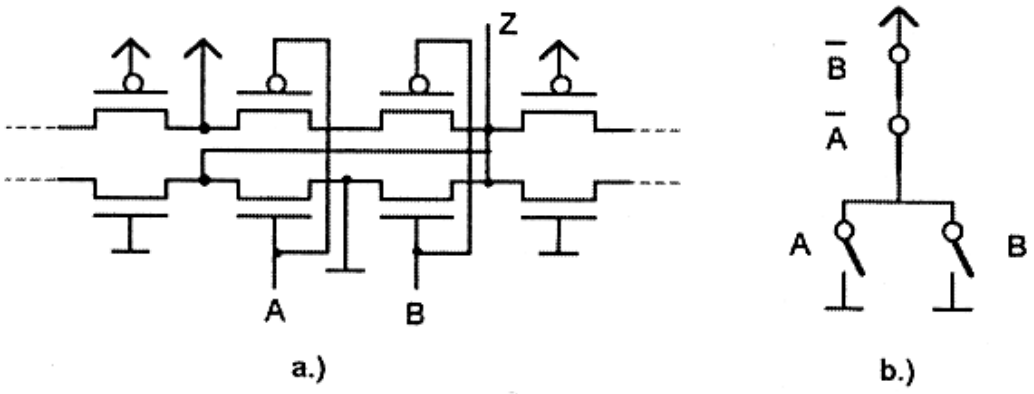
A következőkben néhány konkrét FPGA-t ismertetünk nagyon röviden. Részletes leírásuk a megfelelő katalógusokban találhatók.

2.7.2. Crosspoint 2000

A Crosspoint 2000 család a tranzisztorszintű logikai erőforrásokra épülő FPGA-ra példa:

- az architektúrája sor bázisú,
- a sorokat tranzisztor párok (Transistor Pair Tile, TPT) és tároló ill. multiplexer funkciót ellátni képes RLT (RAM Logic Tile) blokkok alkotják,
- a sorokat horizontális huzalozási szegmensek választják el,
- a sorok közötti összeköttetéseket vertikális huzalozási szegmensek biztosítják,
- minden TPT sor tkp. két tranzisztor sorból áll, egy NMOS és egy PMOS tranzisztorokból álló sorból,
- a huzalozás segítségével a tranzisztorokból CMOS logikai kapuk alakíthatók ki,
- egy sorban a tranzisztorokból kialakított logikák kikapcsolt tranzisztorokkal választhatók el egymástól.

A 2.26a ábra egy a tranzisztor tömbből kialakított NOR kaput mutat. A 2.26b ábra a kialakítás elvi működését magyarázza, a tranzisztorokat kapcsolókkal helyettesítve.



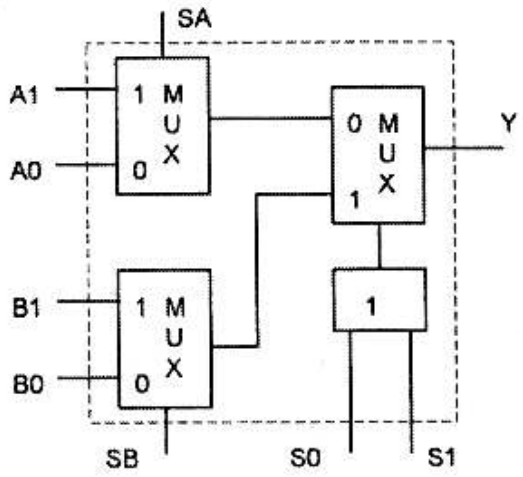
2.26. ábra. NOR kapu kialakítása a Crosspoint 2000-ben

- a TPT jellemzője, hogy a megvalósítandó függvény bonyolultságával arányos a költség,

- a család tagjai antifuse technológiával programozható,
- a család tagjainak bonyolultsága 1300-16000 kapuval ekvivalens,
- a különféle típusok 400-3900 regisztert tartalmaznak,
- típustól függően 90-250 I/O láb (programozható TTL/CMOS szint, slew rate, 4-8-12mA meghajtó képesség).

2.7.3. Actel 1,2,3

Multiplexer alapú logikára példa az Actel 1,2,3, ennek jellemzőit soroljuk fel a következőkben:



2.27. ábra. Multiplexer alapú logikai modul

- sor bázisú FPGA, (az egyes sorok közötti vízszintes huzalozást huzalozási csatornáknak nevezik),
- közepes bonyolultságú, multiplexer alapú logikai modulokat tartalmaz (2.27. ábra).

$$Y = (S0 + S1) * (SB * B1 + \overline{SB} * B0) + (\overline{S0} * \overline{S1}) * (SA * A1 + \overline{SA} * A0)$$

- 4 huzalozási erőforrás típussal rendelkezik: input szegmens, output szegmens, órajel szegmens, és huzalozási szegmens,
- a bemeneti szegmensek a logikai modulok (LM) bemeneteit köthetik a felette és alatta levő huzalozási szegmensekre,
- a kimeneti szegmens az LM-ok kimenetét kötheti össze a modul alatti és feletti huzalozási csatornákkal,
- egy-egy huzalozási szegmens változó hosszú, antifuse-al összekötött részekből áll,
- az órajel szegmens kis késleltetésű vonal, mely sok logikai blokkot képes összekötni, kis órajel csúszást biztosítva,
- az ACT 1-ben nincs dedikált flip-flop, az 2 modulból alakítható ki,
- a fejlesztő rendszerekben az ilyen 2-3 modulból kialakítható egységek megvalósítására ún. hard macrokat definiálnak,
- az ACT 2, 3-ban szekvenciális modul is van, az ACT 1-éhez hasonló felépítésű kombinációs modul mellett, mellyel különféle tárolókat lehet kialakítani,
- az ACT 1, 2-ben egyszerű I/O blokkok vannak 4 ill. 10mA meghajtó képességgel,

- az ACT 3 I/O modulja 2 regisztert tartalmaz (egy input és egy output regisztert), az output regiszter visszacsatolható a belső logikákhoz (mint a PAL-oknál).

Az Actel családok jellemzőit mutatja a 2.2. táblázat.

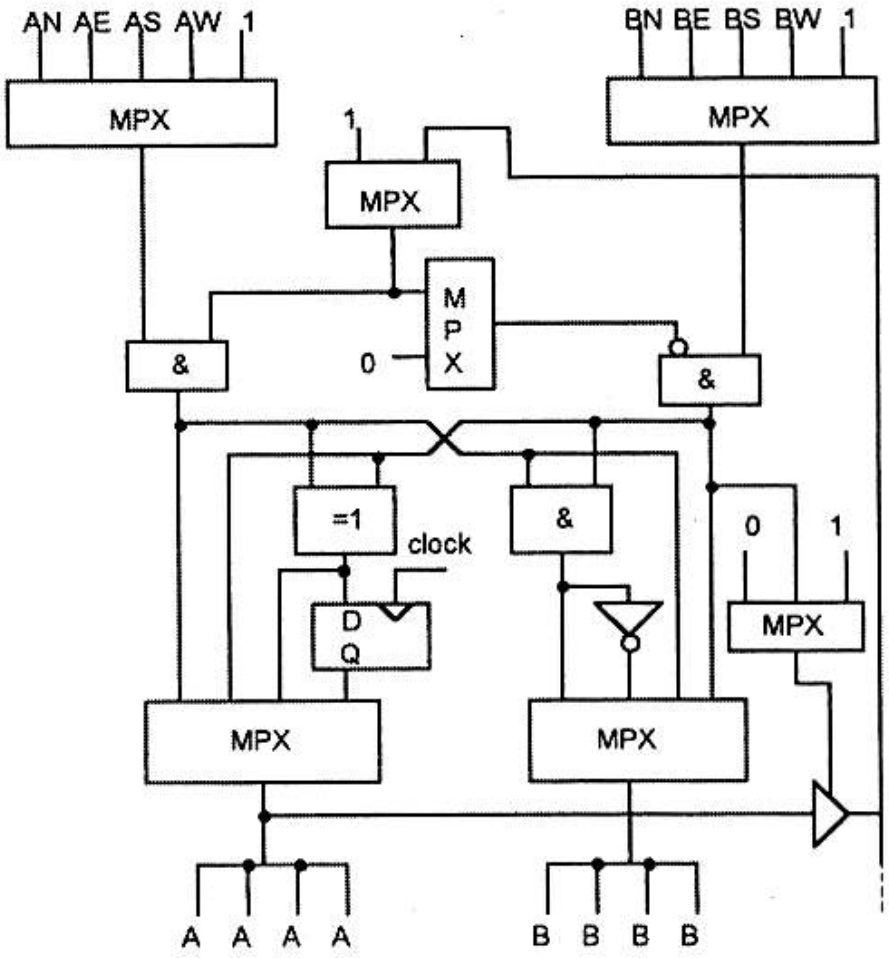
	kapuszám	regiszterek	I/O-k
ACT1	1000-2000	140-270	57-68
ACT2	2000-8000	560-1000	100-160
ACT3	2000-10000	250-1000	80-220

2.2. táblázat

2.7.4. AT 6000

Rugalmasan konfigurálható kapu alapú logikára példa az AT 6000 FPGA (Concurrent Logic):

- szimmetrikus elrendezésű, egyenletesen elosztott logikai cellák,
- kapukat és regisztert tartalmazó logikai cellák (2.28. ábra),
- különösen jó regiszter intenzív és aritmetikai alkalmazásokra,
- 44 konfigurációs lehetőség,
- 20 kombinációs konfiguráció (invertertől az összetett kapukig),
- 11 regiszteres konfiguráció (egyszerű flip-flop, multiplexer bemenetű regiszter, EXOR bemenetű regiszter stb.),
- 5 konstans és 5 három állapotú konfiguráció,
- 2 direkt összekötési lehetőség a szomszédos logikai blokkok között,
- 2-2 lokális és expresszbusz (kis késleltetésű vonalak) soronként és oszloponként,
- a lokális buszhoz a sorban és az oszlopban levő összes cella csatlakozik,
- egy-egy *logikai cella huzalozási erőforrásként* (merőleges lokális buszok összekapcsolása) is alkalmas,
- az expressz busz nem kapcsolódik közvetlenül a logikai blokkokhoz, hanem 8 cellánként egy kapcsoló áramkör lehetővé teszi a lokális busz expressz buszhoz kapcsolását,
- SRAM programozású, a konfiguráció 6 féle módon adható meg, melyek közül 3 mód bemenettel lehet választani:
  a. külső EPROM-al párhuzamosan, növekvő/csökkenő címek felé külső/belső órajelet használva,
  b. szinkron soros átvitel külső/belső órajellel,
  c. párhuzamos mikroprocesszoros porttal .
- egyetlen EPROM-ból több egység is felprogramozható, ilyenkor az egyik egység master, a többi slave,
- a chip *részleges konfigurálás*a is lehetséges.



2.28. ábra. Az AT6000 logikai blokkja

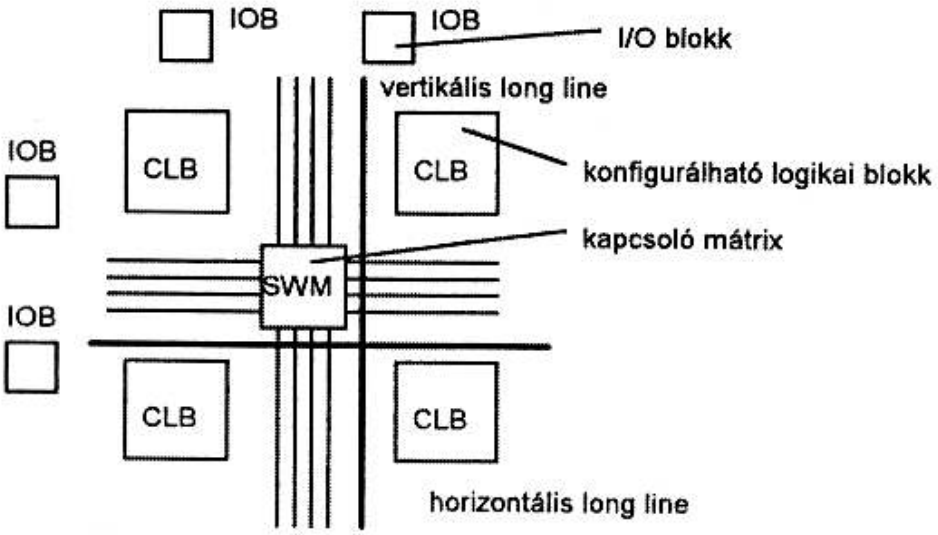
2.7.5. XILINX FPGA családok

Look up table alapú logikára példa a XILINX FPGA család.

Jellemzői:

- kapcsolómátrixokkal összeköthető horizontális és vertikális huzalozási szegmensekkel alakítható ki kapcsolat a távoli logikai blokkok között (2.29. ábra),
- a szomszédos logikai blokkok direkt is összekapcsolhatók,
- a gyors jelek számára horizontális és vertikális ún. long line-ok biztosítják a kis jelkésleltetést,
- a logikai blokkok SRAM (Look Up Table, LUT) alapúak, D flip-flopokkal kiegészítve (aszinkron set és reset lehetőséggel),
- az I/O blokkok konfigurálhatók kombinációs vagy regiszteres bemenetként, kombinációs kimenetként, és az XC3000 és XC4000 családnál regiszteres kimenetként is, programozható polaritással,

- az XC2000, XC3000 és XC4000 családok sok hasonlóságot mutatnak, de az újabb (nagyobb sorozatszámú) családokban nagyobb a logikai blokkok flexibilitása, a huzalozási lehetőségek és a flip-flopok száma nő,
- szimmetrikus elrendezésű, egyenletesen elhelyezett logikai blokkok.

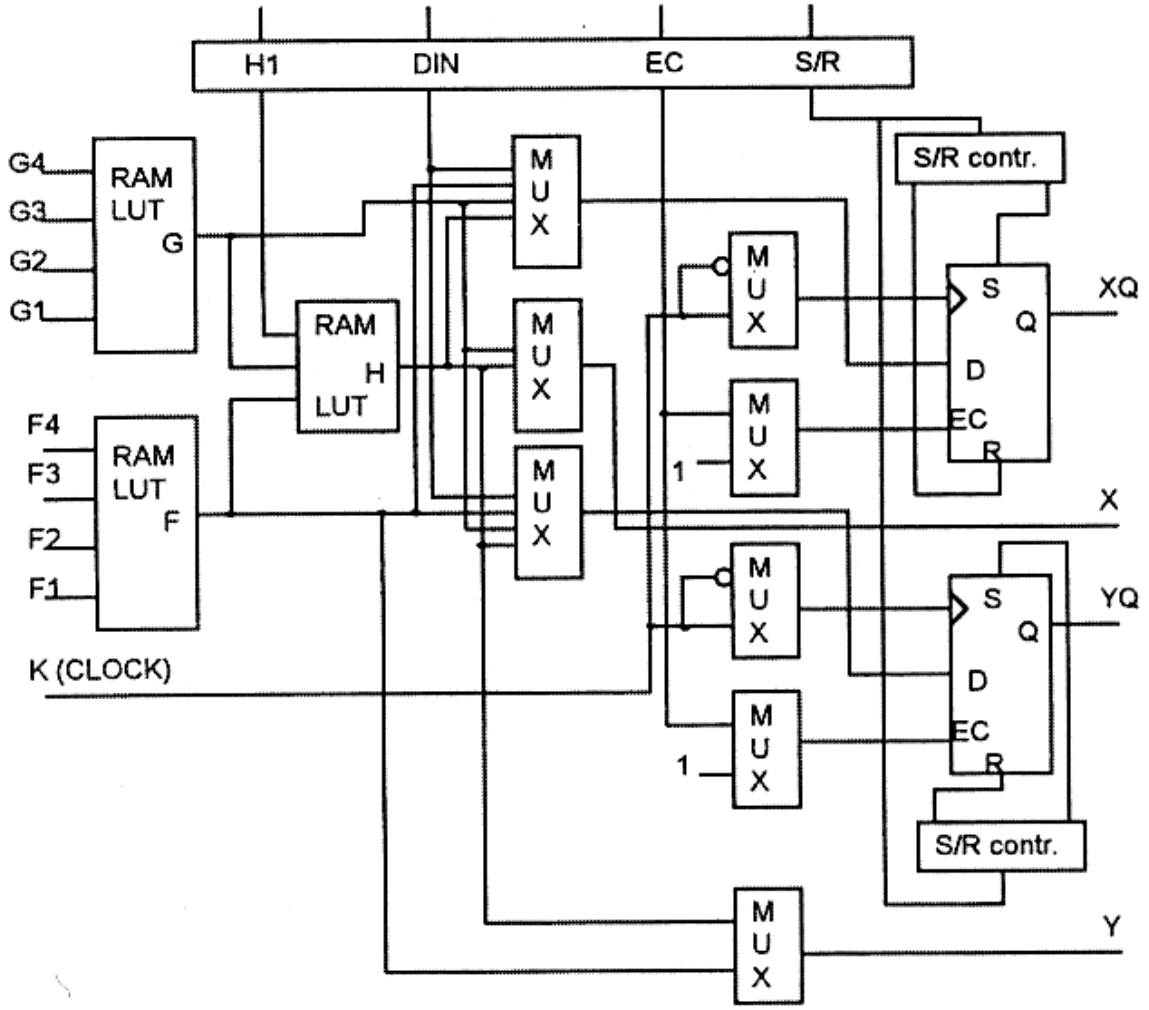


2.29. ábra. A XILINX FPGA-k tipikus struktúrája

Az XC4000 család CLB-jének (2.30. ábra.) specialitásai (ezek egy része az ábrán nincs feltüntetve):

- Egy CLB-vel megvalósítható:
  a. 2 tetszőleges, független 4 változós függvény,
  b. 1 tetszőleges 5 változós függvény,
- 2 D flip-flopot tartalmaz, melyek tárolhatják:
  a. a függvények kimenetét,
  b. egy külső jelet,
- a flip-flopok órajele közös, de az élérzékenység függetlenül programozható,
- gyors *beépített carry logikával* rendelkezik, gyors aritmetikák, számlálók hatékony megvalósítására,
- egy CLB-vel 2 bitnyi aritmetika készíthető,
- a CLB gyors RAM-ként is konfigurálható:
  a. ciklusideje 10 nsec!,
  b. 16x2 bit,
  c. 32x1 bit,
  d. 16x2 bit RAM üzemmódban a 2 RAM kimenet, tetszőleges függvénye is előállítható.

A RAM nagy sebessége miatt még 1 nsec-os hazárd sem lehet a vezérlőjeleiben, ezért a vezérlés megtervezésénél rendkívüli figyelemmel kell eljárni. Az újabb verziókban ezen szinkron beírású RAM alkalmazásával javítottak.



2.30. ábra. Az XC4000 logikai blokkja

- a horizontális long line-ok meghajtására minden CLB alatt és felett egy 3-state buffer áll rendelkezésre, melyeket bármely jel engedélyezhet,
- egy vertikális long line-t felhasználva engedélyezésre, busz meghajtót is létre lehet hozni.

A XILINX FPGA-k felkonfigurálási lehetőségei

A konfiguráció 6 féle módon adható meg, melyek közül 3 mód bemenettel lehet választani:

- *Master soros mód*, melyben a konfigurációt soros PROM-ból olvassa be a XILINX. Több FPGA is láncba kapcsolható, s a konfiguráló PROM-ok is, az órajelet a master adja.
- *Slave soros mód*, ekkor a konfiguráció egy külső egységről (pl. mikroprocesszor port) jön bit sorosan. Az órajelet is a külső egység adja, a felprogramozandó eszközök láncba köthetők (daisy chain).
- *Master párhuzamos mód*, ekkor a konfigurációt közönséges PROM-ból, EPROM-ból olvassa be az eszköz, a szokásos módon megcímezve azt, byte sorosan.

- *Szinkron periféria mód*, itt az adatokat párhuzamosan várja az eszköz, a külső órajel lefutó éléhez szinkronizálva. Több eszköz is láncba kapcsolható, de azok már sorosan kapják az adatot a legelső eszköz DOUT kimenetéről (az órajelet közösíteni kell).
- *Aszinkron periféria mód*, itt is párhuzamosan várja az adatot az eszköz, a CS0, CS1 (mikroprocesszoros rendszer címdekóderéből), és WS (mikroprocesszoros rendszer WR jele) bemenetek ÉS kapcsolatának hátsó élénél. A *RDY / $\overline{BUSY}$* jelzi, hogy új adat jöhet. Több eszköz láncba kapcsolható, de azok már sorosan kapják az adatot a legelső eszköz DOUT kimenetéről, az órajelet is a legelső eszköz adja.

2.7.6. FPGA fejlesztési környezet

Az FPGA-k tervezését CAD rendszerekkel végzik. Ezek a bonyolultabb feladatokból adódóan nagyobb bonyolultságú és nehezebben kezelhető szoftverek, a PLD tervező rendszerekhez képest.

Az FPGA tervezési ciklusa

a. A terv bevitele

A tervet kapcsolási rajzzal és/vagy hardver leíró nyelv segítségével lehet bevinni.

A kapcsolási rajz funkcionális makro blokkokból építkezhet, melyek lehetnek:

a. könyvtári funkcionális blokkok,
b. saját tervezésű funkcionális blokkok.

Kapcsolási rajz készítésekor többszintű hierarchia lehetséges (egy durvább kapcsolási rajz egyes részeit újabb kapcsolási rajzok definiálnak). A bevitt tervet a tervező rendszer ellenőrzi abból a szempontból, hogy teljesíti-e a különféle előírásokat (pl. szintaktikai szabályok).

b. Szimuláció és funkcionális verifikáció

A tervező rendszer létrehozza a bevitt terv belső logikai reprezentációját. Ezután a tesztvektorok alapján ellenőrzi, hogy a bevitt terv megfelel-e funkcionálisan a specifikációnak.

c. A terv leképzése a konkrét FPGA architektúrájára (technology mapping)

Ebben a lépésben a tervezős rendszer a logikai blokkok által biztosított elemek felhasználásával megtervezi a logikai függvényeket és tárolóelemeket.

d. Elhelyezés és huzalozás

A tervező rendszer a logikai blokkokat megfelelteti az IC-ben elhelyezkedő fizikai blokkoknak. Ezután megtervezi a fizikai összeköttetéseket (huzalozás).
Ez a legkritikusabb rész a terv minősége szempontjából. Érdemes inkább drágább, de jó minőségű tervező rendszert vásárolni.

e. A behuzalozott terv késleltetéseinek számítása

Itt a valódi áramköri késleltetéseket próbálja meghatározni a rendszer.

f. Szimuláció és időzítési verifikáció a számított késleltetéseket figyelembe véve

Időkritikus terveknél különösen nagyon fontos a valós időadatokon alapuló szimuláció (pl. setup time betartása), de itt derülhetnek ki az esetleges hazárdok is.

g. FPGA konfigurációs adatok létrehozása

A konfigurációs adatokat (esetleg szabványos file formátumban) létrehozzák és tárolják.

h. FPGA konfigurálása vagy programozása

A konfigurációs adatok alapján a programozó készülék felprogramozza az FPGA-t vagy az EPROM-ot (Xilinx).

i. A beprogramozott eszköz teszteléseA beprogramozott eszközt a valós működési környezet szimulálásával, ill. a valós környezetben tesztelik. A teszteléshez hozzátartoznak a parametrikus vizsgálatok is. Pl. ellenőrzik, hogy a hőmérsékleti, tápfeszültség stb. határokon hogyan viselkedik az eszköz.

2.7.7. Tervezési szempontok FPGA eszközök alkalmazása esetén

Az FPGA-k tulajdonságai más szempontok figyelembe vételét igénylik, mint a PLD vagy SSI-MSI elemekkel történő tervezés. Ráadásul nagyon fontos az egyes típusok egyedi tulajdonságainak figyelembe vétele, mert enélkül nem lehet kihasználni az eszközök lehetőségeit, így mind az erőforrások kihasználtsága, mind a sebesség szempontjából kevésbé hatékony tervek születnek. Ezért a tervezés előtt részletesen tanulmányozni kell a rendelkezésre álló eszközök belső felépítését.

Az FPGA-k közös jellegzetességei

- Az FPGA többnyire regiszterben gazdag architektúra (szemben a PLD-kkel, ami inkább logikában gazdag), így nem kell nagyon spórolni a regiszterekkel.
- A logikai blokkjai kisebb bemenet számra optimalizáltak, mint a CPLD blokkjai.
- A huzalozás és az elhelyezés bizonytalansága miatt nehezebben számíthatók a késleltetések, ezért nagyon fontos a valós időadatokkal történő szimuláció.
- A tervező rendszerek többségének része a funkcionális blokk editor (pld: XILINX XBLOCK). Ez lehetővé teszi, hogy a feladathoz legjobban simuló saját funkcionális elemeket hozzunk létre. A funkcionális elemek tulajdonságait paraméterezéssel állíthatjuk be, pld. számláló modulusa, szinkron vagy aszinkron törlés stb., így nagyobb a tervezői szabadság.

- Könyvtári MSI elemek itt is léteznek (TTL, CMOS sorozat), de ezek alkalmazása nem eredményez optimális megvalósítást, bár a fel nem használt logikai részt többnyire nem építik be a tervezőrendszerek. Pl. ha egy engedélyezhető számlálót állandóan engedélyezünk, az engedélyezéshez szükséges extra logikát kiegyszerűsítik.

Vezérlőegységek tervezése FPGA-val

Vezérlőegységek tervezése esetén hagyományos tervezésnél a minimális állapotszámra törekszünk. Ez a regiszterben szegény, logikában gazdag architektúrák esetén célszerű (PLD-k). A minimális állapotszámú állapotgépnél a flip-flopok vezérlőfüggvénye sok bemenetet igényel (maximálisan: szekunder változók száma + bemenetek száma). A számláló típusú vezérlő szintén nem túl hatékony, mert maga a tölthető számláló és a betöltendő érték előállítása bonyolult logikát eredményez.

FPGA esetén a vezérlőfüggvény minimális bemenetszáma fontosabb szempont, mint a regiszterekkel való spórolás, mert a logikai cellák kevés bemenetet tartalmaznak. Így az olyan állapotkódolás, mely ezt figyelembe veszi, optimálisabb megvalósítást eredményez.

One Hot Encodeing (másképpen: Bit Per State)

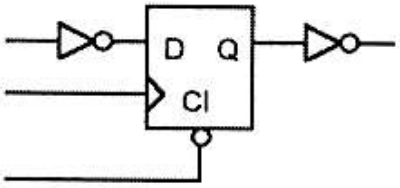
Ennél a módszernél annyi flip-flopra van szükség, ahány állapotú a vezérlő. Minden állapot kódjában csak egyetlen 1-es van. A tervezés a kódolást követően ugyanaz, mint a hagyományos szinkron sorrendi hálózatoknál.

A kódolás előnye, hogy egy-egy flip-flop vezérlőfüggvénye kevesebb bemenetű lesz, mint minimális állapotszámú (minimális állapotkód hosszúságú) kódolás esetén. A kevesebb bemenetszám igény következtében kevesebb logikai szintre van szükség (a kevés bemenettel rendelkező logikai cellákkal több bemenetet csak több szinttel lehet megvalósítani). A kimeneti függvény is egyszerűbb, hiszen nem kell dekódolni az állapotokat.

Hátránya, hogy minden állapotátmenet 2 Hamming-távolságú állapotkód változást eredményez, ami hazárdot okozhat a kimeneti függvényben (funkcionális hazárd).

A kezdeti állapot beállítása One Hot Encodeing esetén

Az OHE állapotkódolásnál problémát jelent a kezdeti állapot beállítása, hiszen az FPGA-k a tápfeszültség megjelenésekor többnyire törlik az összes flip-flopot. Ez az OHE kódolásnál illegális állapotot jelent. A problémát úgy lehet megkerülni, hogy azon *D flip-flop bemenetére és kimenetére, invertert* kötünk, amelynek 1 kezdőértéket akarunk adni (2.31. ábra).



2.31. ábra. Kezdeti állapot 1-be állítása törölhető flip-flop esetén

Elosztott, hierarchikus vezérlő

Nagyobb állapotszám esetén érdemes részekre osztani a feladatot, oly módon, hogy egy-egy részfeladatot egy kisebb állapotszámú vezérlő old meg, s az egyes vezérlőket egy 'master' vezérlő indítja. Ennek előnye, hogy a kisebb vezérlőket könnyebb tervezni és tesztelni.

Az FPGA-knál inkább az elosztott vezérlést célszerű alkalmazni, az egyetlen nagy vezérlő helyett. Egyrészt a már említett okból, másrészt így a vezérlő közel kerülhet a jeleit felhasználó egységekhez, ami kisebb jelkésleltetést jelent, s ezért gyorsabb működést enged meg.

Az órajel csúszás

Gyors ill. nagy rendszereknél a jelváltozásokhoz képest hosszú órajel utak ill. órajel pufferelés esetén hibásan működhet a szinkron sorrendi hálózat, ha ugyanazt az órajelet különböző helyeken használjuk fel. A probléma oka az, hogy a késleltések miatt az órajelek nincsnek fázisban (órajel csúszás).

Példaként vegyünk két olyan szinkron sorrendi hálózatot S1 és S2, amelyek közül S1 felhasználja S2 kimeneteit feltétel bemenetként. Tegyük fel, hogy az órajel csúszás miatt S2 hamarabb kapja meg az órajelet, mint S1. Ekkort előfordulhat, hogy S2 kimenete már megváltozik, mikorra S1 megkapja az órajelet, s így a már megváltozott kimenetet érzékelve, S2 nem a specifikáció szerinti állapotba ugrik.

A jelenség elkerülése miatt fontos, hogy az FPGA-knál az órajelnek használt jeleket az erre a célra fenntartott gyors vezetékeken vezessük. Ha bufferelést alkalmazunk az órajelen, akkor egy órajel elosztó csomópontból minden irányban tegyünk buffert, hogy a késleltetéseket kiegyenlítsük.

A késleltetések szempontjából is kellően át nem gondolt tervezés a rendszer működésképtelenségéhez ill. nehezen felderíthető soft hibákhoz vezethet. Figyelembe kell venni, hogy a késleltetések időben is változnak, egyrészt az alkatrészek öregedése miatt hosszú távon, másrészt a környezeti hőmérséklet függvényében, ráadásul az alkatrészek késleltetésének szórása van (a különböző alkatrészeknél kismértékben eltérnek a késleltetések). Így a tervezési hibák esetleg a prototípuson nem jelentkeznek, csak sokkal később. A tervezés minél későbbi szintjén jelentkezik a hiba, annál költségesebb a kijavítása.

Álvéletlen generátor, mint számláló

A szokásos bináris, decimális stb. számlálók helyett az FPGA struktúrához jobban illeszkedik, a már ismeretett, EXOR kapukkal visszacsatolt shiftregiszterrel megvalósított számláló (álvéletlen generátor). Az elérhető állapotszám csak 1-el kevesebb, mint a bináris számlálóknál. A vezérlő függvények nagyon egyszerűek.

Hátránya, hogy az egymást követő állapotok kódja nem a megszokott növekvő bináris számok szerint következik, így az egyes kimenetek nem rendelkeznek állandó kitöltési tényezővel, ezért frekvencia osztóként nem alkalmazható ez a megoldás. Külső ROM címzésére is kényelmetlen, mert a beégetendő adatok címét is az álvéletlen generátorhoz kellene igazítani.

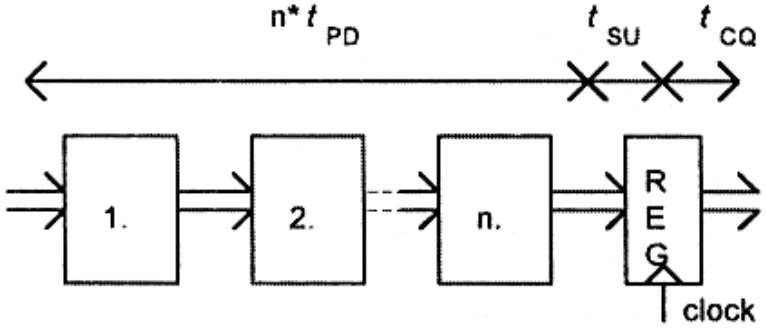
Az ilyen számláló modulusát törléses módszerrel nem lehet csökkenteni (a 000..0 állapotban bennt ragad), csak betöltéses módszerrel.

A pipeline alkalmazása

Ha az adatstruktúra komplex funkcióit részműveletekre bontjuk, jobban áttekinthetőek és tervezhetőek a logikák. Ezt a felbontást legtöbbször maga a feladat sugallja. A részműveletekre bontás után azonban célszerű regisztereket elhelyezni az egyes műveletvégző egységek közé, ugyanis ezzel meggyorsíthatjuk az áramkör működését. Ezt nevezik pipeline struktúrának.

Ha az egyes műveletvégző egységeket (kombinációs hálózatok) egymás után kapcsolva, az egész végére teszünk egyetlen regisztert (2.32. ábra), az órajel minimális periódus ideje n műveletvégző egység esetén:

$$T_{CLK} = nt_{PD} + t_{SU} + t_{CQ} \qquad (2.3)$$

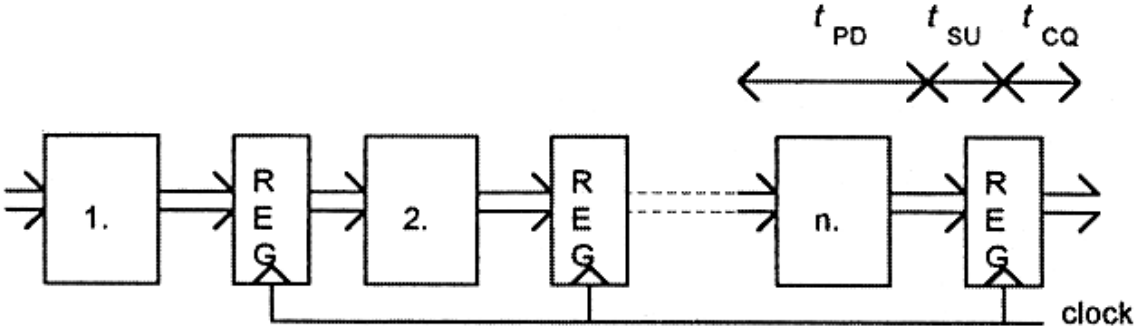


2.32. ábra. Soros feldolgozás többszintű kombinációs hálózattal

t_{PD}: a kombinációs hálózat késleltetése, t_{SU}: a regiszter setup time-ja, t_{CQ}: a regiszter kimenet órajelhez viszonyított késése.

Ha a műveletvégző egységek közé regisztereket teszünk (2.33. ábra), akkor ugyan az átfutási idő megnő (az első eredmény viszonylag hosszú idő múlva keletkezik), de a hatékonyság jelentősen javul, mert az órajel periódusideje lecsökkenthető:

$$T'_{CLK} = t_{PD} + t_{SU} + t_{CQ} \qquad (2.4)$$



2.33. ábra. pipeline-os soros feldolgozás időzítési modellje

Az előbbi képletekben az egyszerűség miatt az egyes műveletvégző egységek késletetési idejét azonosnak vettük.

2.8. Áramköri tulajdonságokból adódó problémák és kivédésük

A nagy bonyolultságú logikákkal való tervezés során célszerű szem előtt tartani bizonyos, az áramköri tulajdonságokból adódó problémákat. Ez az oka, hogy a következőkben röviden megemlítünk néhány, inkább a digitális elektronikához kapcsolódó fogalmat.

2.8.1. Metastabilitás

Ha egy flip-flop bemenetén az órajelhez túl közel van a bemeneti jel változása (a setup time-ot nem tartjuk be), a flip-flop ún. metastabil állapotba kerülhet. Ez azt jelenti, hogy egyrészt a flip-flop kimenete az órajelhez képest a definiált késletetésnél jóval később jelenik meg, másrészt a kimenet egy ideig köztes (nem logikai) szinten marad. Ilyenkor nem jósolható meg a metastabil állapot hossza, s a végén kiadott logikai szint is véletlenszerű. A jelenség magyarázatához a logikai áramkörök tranzisztor szintű vizsgálata szükséges, ezért azzal itt nem foglalkozunk.

PLD struktúráknál a flip-flop setup time-jához hozzáadódik a kétszintű AND OR tömb késletetésének és az órajel meghajtó áramkör késletetésének különbsége, amely a direkt órajelezésű PLD-k esetén még nagyobb. Ezért erre figyelemmel kell lenni, nehogy az említett problémához vezessen.

A jelenség főként az órajelhez aszinkron jelek esetén lép fel, ami sok esetben elkerülhetetlen. Csökkenteni lehet az esélyét, két flip-flop egymás után kapcsolásával. Az órajel frekvenciát úgy kell megválasztani, hogy az első flip-flop metastabil állapota nagy valószínűséggel megszűnjön egy órajel periódusidő alatt. Mivel a metastabil állapot hosszát nem ismerjük pontosan, a módszer csak csökkenti a kialakulás esélyét, de nem szünteti meg teljesen. Ezért minden esetben, amikor csak lehetséges, szinkronizálni kell a flip-flopok adat és órajel bemenetét.

2.8.2. Ground Bounce ("ugráló földelés")

Ha egy IC belsejében nagyszámú kimenet egyszerre változik, tüskéket okozhat az IC táp és föld vezetékén, az IC belsejében a táp ill. föld láb és a chip közötti vezetéken, annak induktivitása miatt (hirtelen nagy áram változás). A jelenség nem csak a gyors bipoláris és CMOS eszközökben lép fel, hanem a nagy lábszámú gate array-kben is. A földpotenciál ugrás sokkal kritikusabb, mint a tápfeszültségé, innen ered az elnevezés. Az IC tervezők megfelelően elosztott földelési rendszerrel védekeznek a probléma ellen, dupla huzallozással készítik a föld és táp lábakat továbbá több föld és táp lábat alakítanak ki.

A felhasználó a következőket teheti:

Nagyon jó táp és föld feszültség szinteket kell biztosítani (stabil, megfelelő áramú tápegység, jó földelési rendszer, szűrőkondenzátorok, többrétegű (4) NYÁK).

Az IC összes föld és táp lábát fel kell használni.

Ha a az IC és az alkalmazás megengedi, ki kell használni a slew-rate korlátozás lehetőségét (75%-al képes csökkenteni a tranziens amplitúdót). Ezt ma már sok FPGA lehetővé teszi.

CMOS bemeneti szinteket kell alkalmazni, ha csak lehetséges (sok FPGA-nál programozható).

A nagyfrekvenciás ill. nagyáramú kimeneteket és a zavarérzékeny bemeneteket (órajel bemenet) szét kell választani, és az IC tok különböző földelési lábai közelében kell kivezetni. Ez is az FPGA lábkiosztásának egy szempontja.

2.8.3. Latchup

A CMOS áramkörökben parazita bipoláris tranzisztorokból SCR (Silicon Controlled Rectifier) struktúra jön létre, amely bizonyos feltételek esetén bekapcsolódva folyamatos táp-föld zárlatot, s ezzel az IC tönkremenetelét okozhatja.

A jelenség létrejöhet, ha:

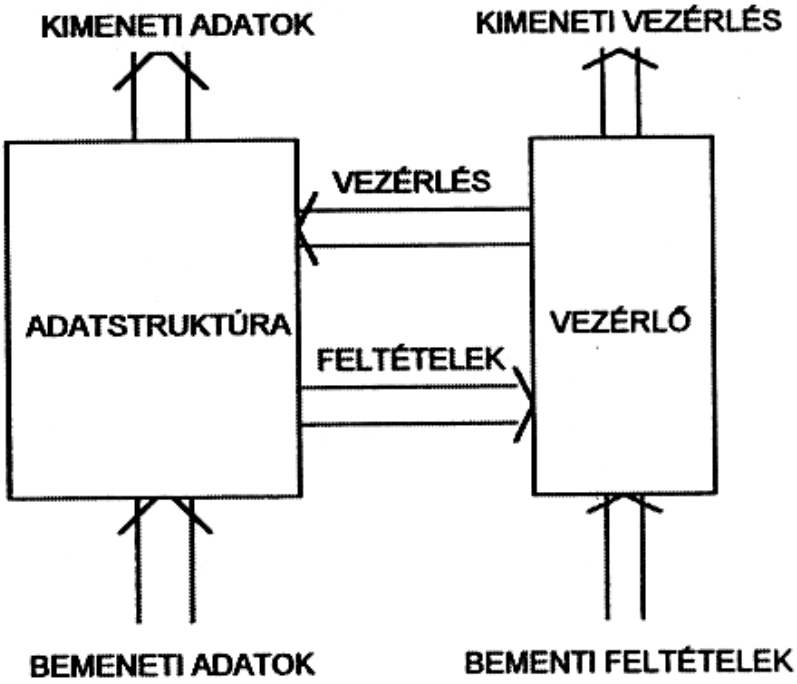
- az IC bemenetét meghajtjuk, mielőtt tápot kapna,
- túlfeszültség a táp és föld között (tápegység bekapcsoláskor túlleng, hosszú tápvezeték induktivitása miatt),
- túlhajtás (pl. túl nagy kapacitású vezeték meghajtása).

Kivédése: meg kell akadályozni, hogy a fenti feltételek létrejöhessenek.

3. TERVEZÉS ADATSTRUKTÚRA-VEZÉRLŐ SZEMLÉLETTEL

A megismert funkcionális elemek felhasználásával már viszonylag nagyobb áramkörök megtervezhetők, akár ad'hoch módszerrel is. Azonban egy terv annál kézbentarthatóbb, a tervezés annál több mozzanata gépesíthető, minél több szisztematikus módszert alkalmazunk a tervezés során. A címben szereplő eljárás is ezt célozza.

Alapötlete, hogy próbáljuk szétszedni két fő részre a feladatot, az egyik részt - ezt nevezzük adatstruktúrának - még mindig intuitív módon kell majd terveznünk, viszont a másiknál - ez a vezérlő - meg van a lehetőségünk a szisztematikus tervezésre, mert arra kidolgozott módszerek állnak rendelkezésünkre. Nagyon nagy méretű feladat esetén célszerű több egymástól jól elkülöníthető részre osztani azt, s egy-egy részfeladatra alkalmazni az adatstruktúra-vezérlő szemléletű tervezést.



3.1. ábra. Adatstruktúra-vezérlő felosztás

Ha egy feladatot a bemeneti és kimeneti adatok között elvégzendő, valamely algoritmussal megadott transzformációnak tekintünk, akkor az adatstruktúra a transzformáció egyes lépéseinek megvalósításához szükséges funkcionális elemek, és az azokat összekapcsoló adatutak összessége.

Az adatstruktúra rendelkezik egy felülettel a vezérlő felé. A vezérlő feladata, hogy az adatstruktúrából és a külvilágból jövő feltételjelek figyelembe vételével a vezérlőjelek segítségével megvalósítsa az adatstruktúrán az algoritmust (3.1. ábra).

Egy feladat hatékony megoldása természetesen nagyon függ az algoritmus megválasztásától. A választott algoritmus viszont nagyrészt meghatározza az adatstruktúrát. Adott adatstruktúra esetén már megadható a vezérlőjének folyamatábrája

(mely vezérlőjelet mikor kell kiadni a különböző feltételek függvényében). Ennek ismeretében pedig megtervezhető a vezérlő.

Vezérlők tervezése szisztematikus módszerrel

A vezérlő tulajdonképpen egy sorrendi hálózat, melynek bemenetei az adatstruktúrából és a környezetből jövő feltételjelek, kimenetei pedig a vezérlőjelek.

A vezérlő típusának kiválasztásához a fő szempontok, hogy az algoritmus milyen bonyolult (hány állapotot igényel), milyen a sebességigénye, mennyire rugalmas (egy esetleges módosítás mennyibe kerül). Ez utóbbi szempontból a ROM-ot tartalmazó vezérlők (mikroprogramozott és mikroprocesszoros vezérlő) rugalmasabbak, mint a többi ún. huzalozott logika.

Random logika

Ide tartozik a hagyományos aszinkron, szinkron sorrendi hálózat, és az ad'hoch vezérlő tervezés.

Hagyományos aszinkron sorrendi hálózat

Csak nagyon kevés állapotszám és nagyon nagy sebességigény esetén jöhet szóba. Tervezése nagyon nagy tapasztalatot igényel, működése nehezen kézbentartható, nehezen tesztelhető. Mindezek miatt lehetőleg kerüljük az alkalmazását.

Hagyományos szinkron sorrendi hálózat

Megvalósítására nagyon alkalmasak a PAL-ok, PLA-k. Ma már nagy sebességgel (100 MHz felett) működő példányok is léteznek. Tervezésüket tervező rendszerek segítik, melyek akár magas szintű nyelven leírt kódolt (esetleg kódolatlan) állapotgráfból képesek megtervezni az áramkört (pl. ABEL nyelv). Ezeket alkalmazva nagy állapotszámú SSH (szinkron sorrendi hálózat) is tervezhető, szemben a régebben alkalmazott kézi módszerrel (Karnaugh tábla), amikor pl. 16 állapot felett, és 2-nél több bemenet esetén már nagy nehézségekbe ütközött a tervezés.

Ad'hoch vezérlő tervezés

Egy gyakorlott tervező ad'hoch módszerrel is képes az adatstruktúra vezérlését megoldani. Az ilyen megoldásokban azonban sok a hibalehetőség, a tervezés folyamata lassú, s a terv mások számára kevéssé áttekinthető (a tervező helyettesítése problémás, ha változtatásra van szükség) és nehezen módosítható.

Szisztematikus vezérlők

A szisztematikus vezérlőkhöz a fázisregiszteres, mikroprogramozott és mikroprocesszoros vezérlő tartozik. A szisztematikus vezérlők hardver kialakítása előre meghatározott struktúrát követ, szemben a random logikával. Tervezésük egyszerűbb,

jobban kézben tartható. Sokkal bonyolultabb vezérlések valósíthatók meg, mint a random logika esetében.

Fázisregiszteres vezérlő

A fázisregiszteres vezérlőnél a hálózat állapotát az ún. fázisregiszter tárolja. A tervező feladata a folyamatábra alapján, a struktúra feladatfüggő részeinek (kombinációs hálózat) megtervezése. Itt lényeges a különbség a hagyományos sorrendi hálózathoz képest, hogy a struktúra egy része adott, és hogy nem állapotgráf, hanem folyamatábra alapján kell tervezni, ugyanis az utóbbinál csak a kétfelé ágazás megengedett. A fázisregiszteres vezérlő egyik típusa a számlálós vezérlő.

Mikroprogramozott vezérlő

A mikroprogramozott vezérlők esetében a vezérlő működését egy ROM-ban tárolt mikroprogram határozza meg. A feladat bizonyos mértékű módosítása esetén, csak a ROM tartalmát kell módosítani, így ez sokkal rugalmasabb, mint az előbbi huzalozott logikák és sebességben sem sokkal marad le tőlük.

Mikroprocesszoros vezérlő

A mikroprocesszoros vezérlő a legkomplexebb feladatok megoldására is alkalmas, ami egyrészt a mikroproceszor aritmetikai képességeinek, másrészt a kialakított struktúra rendkívüli rugalmasságának köszönhető. Sebessége viszont némileg elmarad a többi vezérlő sebességéhez képest.

3.1. Számláló típusú vezérlő

3.1.1. Lép vagy várakozik típusú vezérlő

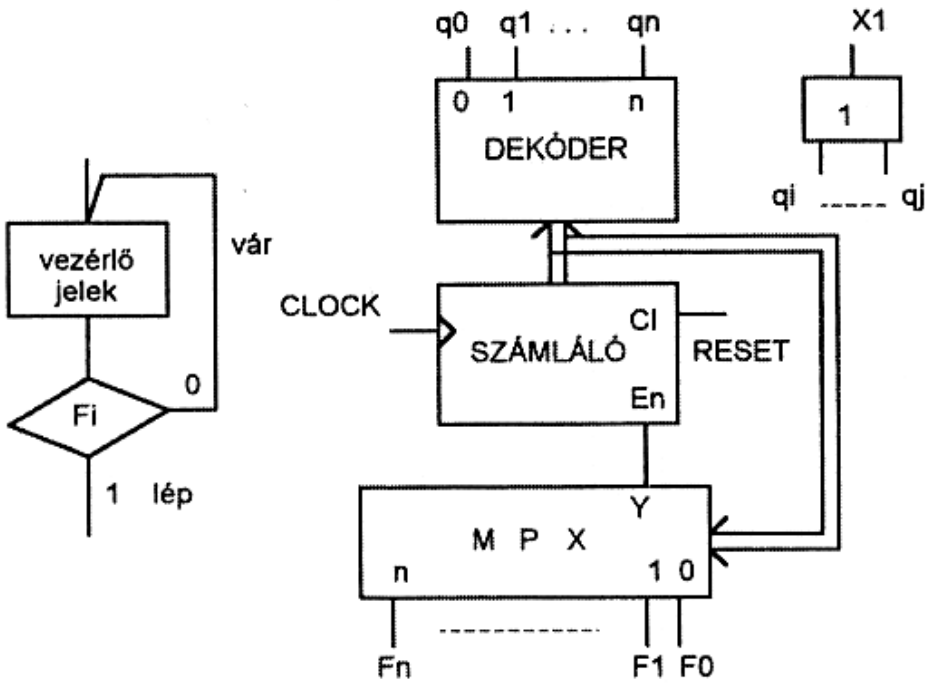
A számláló típusú vezérlő legegyszerűbb változata olyan folyamatábrát képes megvalósítani, amely csak lépést vagy várakozást tartalmaz (3.2. ábra).
A vezérlő a RESET jel hatására a kezdőállapotba kerül (a számláló nullázódik). A számláló címzi a feltétel multiplexert, így az *i*-edik állapotban az F*i* feltétel lesz kiválasztva. Ha a kiválasztott feltétel teljesül, a számláló engedélyeződik, s a következő órajelre lép, egyébként marad az aktuális állapotában. Az i-edik állapotban a dekóder *i*-edik kimenete aktív.

A vezérlőjelek előállítása

A vezérlőjeleket a dekóder kimenetek felhasználásával lehet előállítani. Ha egy vezérlőjel több állapotban is aktív, akkor a megfelelő állapotokat dekódoló kimenetek (q_i) VAGY kapcsolatával állíthajuk elő. Az így előállított vezérlőjelek hazárdosak, mivel a dekóder kimenetén funkcionális hazárd lép fel, a több Hamming-távolságú bemeneti változás

(számláló kimenete) miatt. A hazárdmentes kimenet előállítását a 3.3 fejezetben ismertetjük.

Ha a jelet sok állapoton keresztül nem kell változtatni, akkor olcsóbb lehet az előállítása flip-flop segítségével. A flip-flop állapotát kell a megfelelő időpontban 0-ba vagy 1-be állítani. Ennek megvalósítását a 3.3.1. fejezetben részletesen tárgyaljuk.



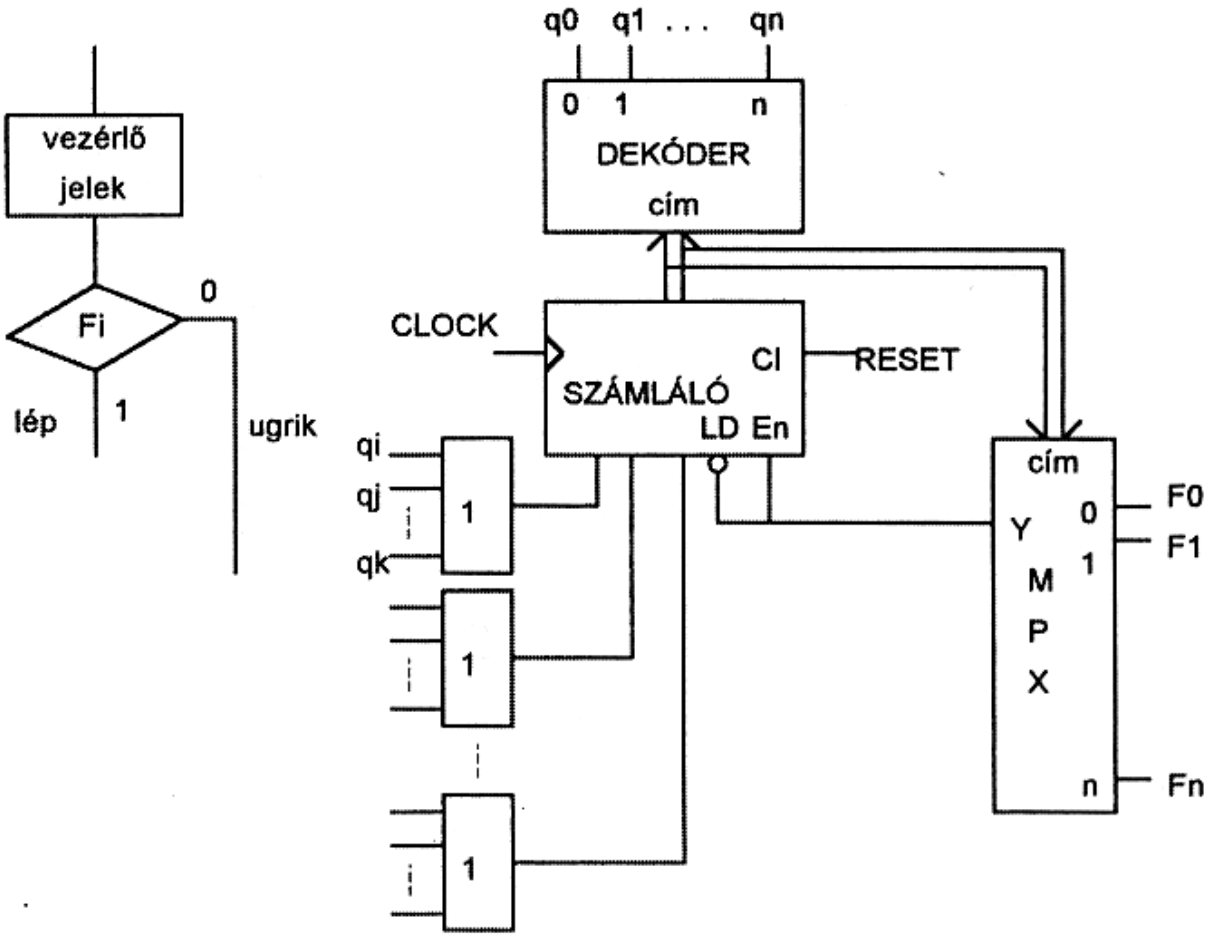
3.2. ábra. Lép vagy várakozik típusú számlálós vezérlő

3.1.2. Ugrik vagy lép típusú vezérlő

A 3.3. ábra szerinti módosítással a vezérlő egy feltétel nem teljesülése esetén ugrik, (betöltésbe vezérli a számlálót) egyébként pedig lép. Ezzel már minden folyamatábra megvalósítható. A számláló szinkron töltésű, egyébként hibás működéshez vezetne a multiplexer kimenetén jelentkező funkcionális hazárd ill. a nem teljesülő feltétel esetén kialakuló aszinkron visszacsatolás miatt.

A következő címet előállító logika megtervezése

Itt a számlálóba a betöltő bemenetekre kapcsolódó kombinációs hálózat (VAGY kapuk) által előállított érték töltődik be. Ha az *i*-edik állapotból a *j*-edik állapotba kell ugrani, akkor a dekóder q_i kimenetét rá kell kötni mindazon VAGY kapuk bemenetére, amely VAGY kapukhoz tartozó helyiértéken a *j*-t bináris számként felírva, 1-es szerepel. (Pl. 4 bites számláló esetén, ha a 2. állapotból a 9.-be kell ugrani (QdQcQbQa=1001), akkor a q_2-t rá kell kötni a számláló Dd és Da betöltő bemeneteire menő VAGY kapura.



3.3. ábra. Ugrik vagy lép típusú vezérlő

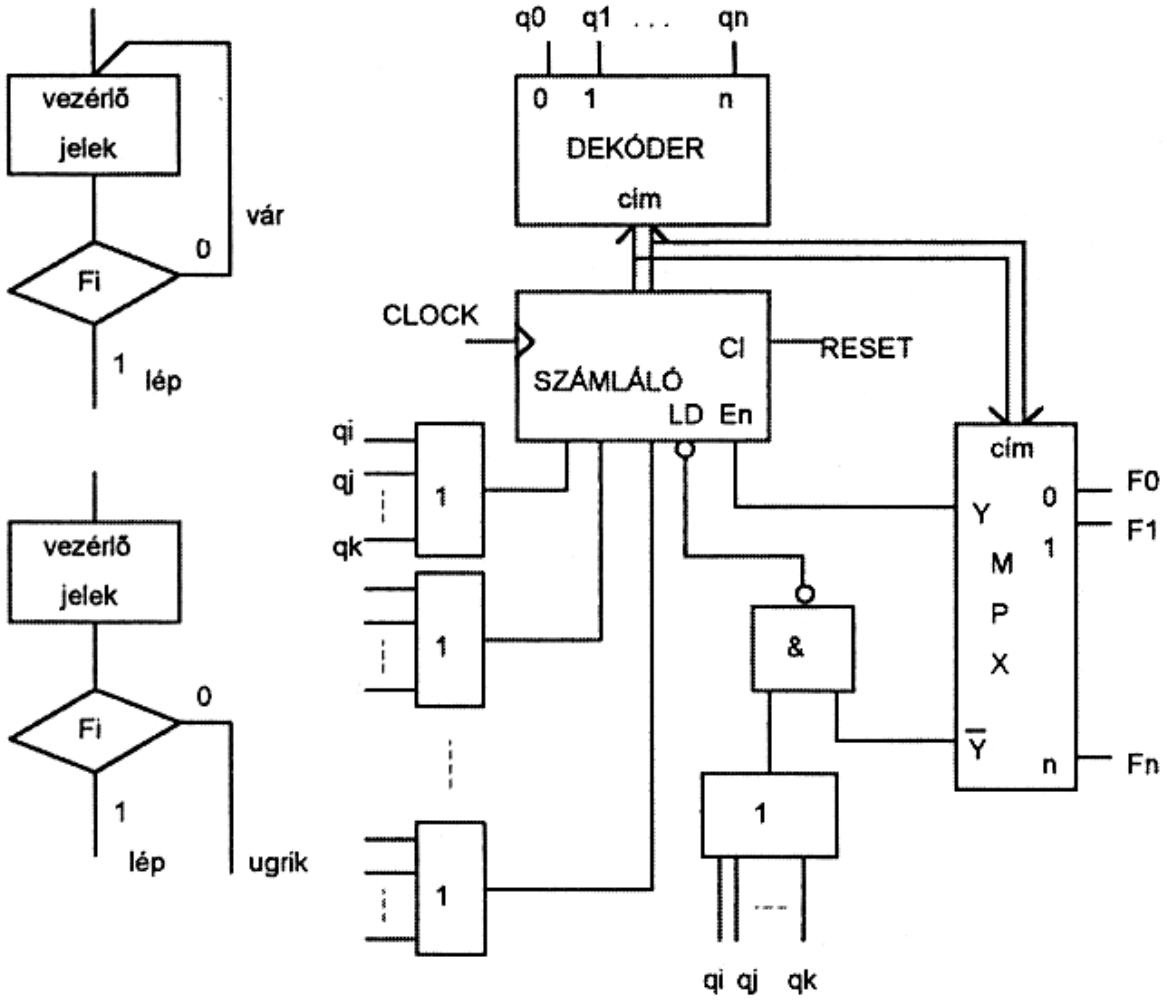
3.1.3. Ugrik, lép vagy várakozik típusú vezérlő

Ha egy folyamatábra sok várakozást tartalmaz, akkor mivel az előbbi vezérlőnél a várakozás is csak ugrással oldható meg, sok bemenetű VAGY kapukra lesz szükség. Egy további módosítással ezen segíteni lehet. A 3.4. ábrán a számláló betöltő bemenetén egy járulékos kombinációs hálózat található. Itt a feltétel nem teljesülése esetén a betöltés feltétele, hogy az aktuális állapothoz tartozó dekóder kimenetet rákapcsoljuk a járulékos kapu bemenetére. Ha ezt nem tesszük meg, akkor a feltétel nem teljesülése esetén várakozni fog a vezérlő. Ezzel egy lépést, várakozást és ugrást megvalósító vezérlőt kaptunk.

Ha a számláló típusú vezérlőket összehasonlítjuk egy hagyományos szinkron sorrendi hálózattal, akkor a 3.1. táblázatban látható megfeleltetéseket találjuk.

Hagyományos szinkron hálózat:	*Számláló típusú vezérlő:*
állapottároló	számláló regiszterei
kimeneti kombinációs hálózat	dekóder, és kimeneti hálózat
szekunder változók vezérlő függvényei	multiplexer, betöltő logika, számláló belső logikája

3.1. táblázat. Hagyományos SSH és számláló típusú vezérlő összehasonlítása



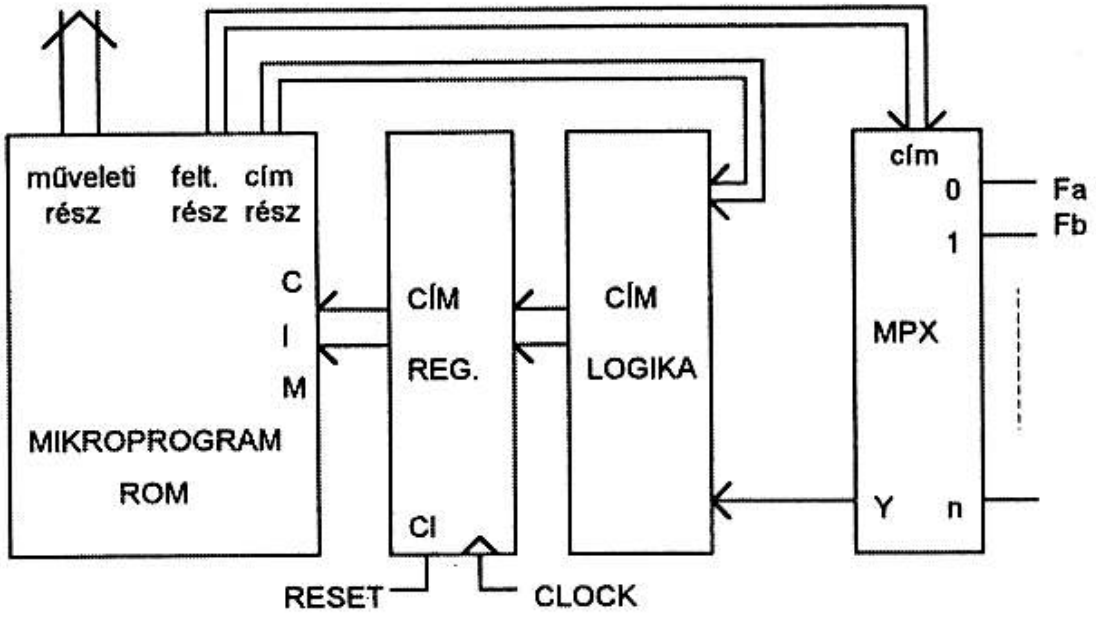
3.4. ábra. Ugrik, lép vagy várakozik típusú vezérlő

A számláló típusú vezérlőnek ma már csak történeti jelentősége van (belőle alakult ki a számlálós címregiszterű mikroprogramozott vezérlő), diszkrét áramkörökből való felépítésük gazdaságtalan. Esetleg chipek belsejében létrehozandó kis állapotszámú vezérlők esetén jöhetnek szóba. Azonban a jelenlegi CAD rendszerekkel ilyen esetekben célszerűbb hagyományos szinkron sorrendi hálózatot tervezni az állapotgráf megadásával, mivel ez optimálisabb megoldást eredményez.

3.2. Mikroprogramozott vezérlő

Ez a vezérlő is szinkron sorrendi hálózat. Az állapottároló a címregiszter. Az összes többi egység kombinációs hálózat. (ROM mint univerzális kombinációs hálózat.) A mikroprogramozott vezérlő legnagyobb előnye, hogy a megvalósítandó vezérlést, a mikroprogram ROM-ban tárolt mikroutasítások írják le. Ha egy megtervezett vezérlésen változtatni kell, akkor az esetek nagyobb részében csak a mikroprogram ROM tartalmát kell megváltoztatni. Az állapotszámnak a ROM-ok mérete szab egy határt, ami ma már igen nagy. Azonban a gyakorlatban néhány ezer állapot felett többnyire nem célszerű

mikroprogramozott vezérlőt használni, mert ilyen bonyolultságú feladatok sokkal kényelmesebben oldhatók meg mikroprocesszorral. Inkább azoknál a feladatoknál lehet szükséges az alkalmazásuk, amikor az állapotszám viszonylag nagy, de a sebességigény miatt a lassabb mikroprocesszoros vezérlők nem alkalmazhatók. A mikroprogramozott vezérlők általános blokksémáját mutatja a 3.5. ábra.



3.5. ábra. Mikroprogramozott vezérlő általános blokksémája

A mikroprogramozott vezérlő ún. **mikroutasítás**okat hajt végre. Egy mikroutasítás 3 részből áll:

<művéleti rész>< feltétel rész>< cím rész>

Az *i*-edik állapotban, a mikroutasítás feltétel része kiválasztja a figyelni kívánt feltételt. A címlogika a kiválasztott feltétel, és a mikroutasítás címrésze függvényében előállítja a következő mikroutasítás címét. A mikroutasítás műveleti része adja a kimenetet.

3.2.1. A vezérlőjelek kódolása

A vezérlőjeleket a mikroutasítás műveleti részéből 3-féle módon állíthatjuk elő.

Horizontális kódolás

Ha a folyamatábra olyan, hogy vannak állapotok, amikor több vezérlőjel egyszerre aktív, akkor az ún. horizontális kódolást lehet alkalmazni (3.6a ábra). Ez azt jelenti, hogy a mikroutasítás műveleti részének minden egyes bitje egy külön vezérlőjelhez van rendelve. Ez hosszú mikroutasítást, nagy szóhosszúságú ROM-ot igényel, viszont a maximálisan párhuzamos működés miatt gyors.

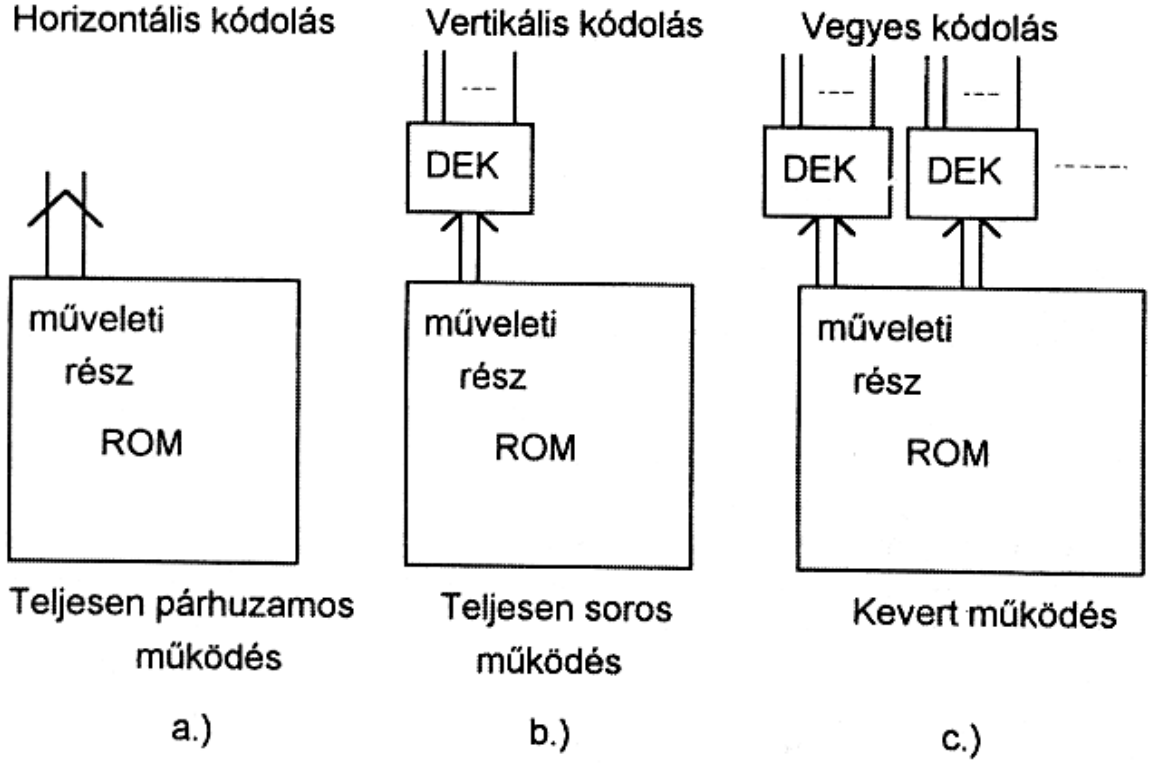
Vertikális kódolás

Ha a folyamatábra minden egyes állapotában csak maximum egy vezérlőjel aktív, akkor műveleti részben csak a vezérlőjel sorszámát kell tárolni binárisan, s ez az információ többnyire sokkal kevesebb bitet igényel, mint a horizontális megoldás. Ekkor a vezérlőjeleket a műveleti rész által címzett dekóder állítja elő. Ezt a módszert nevezzük vertikális kódolásnak (3.6b ábra). A ROM szószélessége szempontjából ez a legkedvezőbb, viszont a soros működés miatt lassú.

Kevert kódolás

Ha a vezérlőjeleket olyan diszjunkt csoportokra tudjuk osztani, hogy egy-egy csoporton belül mindig csak maximum egy jel aktív, akkor az egyes jelcsoportokon belül horizontális kódolást alkalmazhatunk. Ezt a módszert nevezik kevert kódolásnak (3.6c ábra).

A folyamatábra megtervezésénél bizonyos esetekben a feladat megoldása szempontjából közömbös, hogy bizonyos jeleket egymás után, vagy egyszerre adunk ki. Ekkor, ha a szekvenciális végrehajtás mellett döntünk, akkor nő az állapotszám, s ezért esetleg eggyel több címbittel rendelkező (kétszer nagyobb) ROM-ra lehet szükség. Ha a párhuzamos végrehajtást részesítjük előnyben, akkor viszont a szószélesség nő, s esetleg egy ROM-ot párhuzamosan kell kötnünk a már meglevővel.

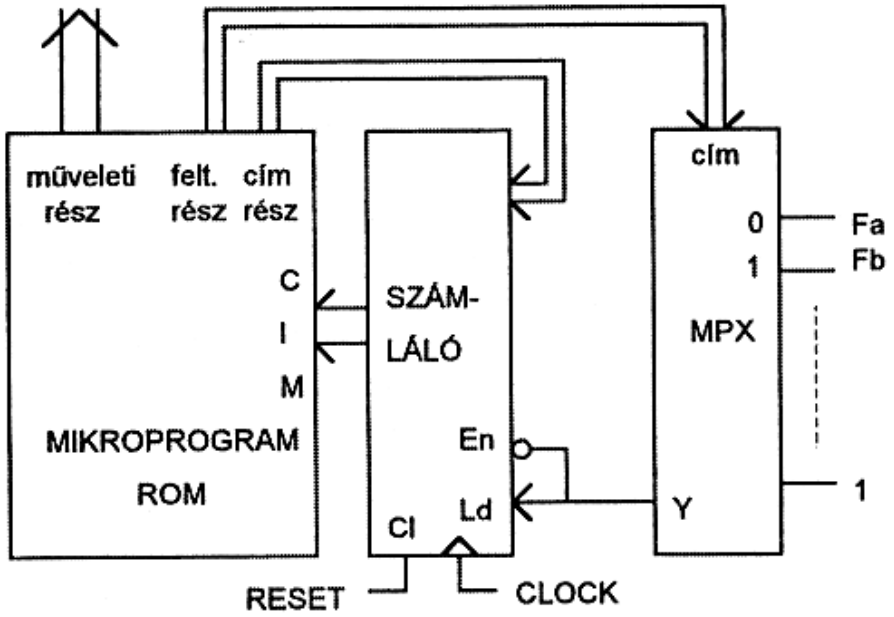


3.6. ábra. Mikroprogramozott vezérlők kimeneti kódolása

A következőkben a teljesség igénye nélkül 3 különféle címlogikájú mikroprogramozott vezérlőt mutatunk be. A bemutatandó mikroprogramozott vezérlő struktúráknál bonyolultabbak is léteznek, azonban a cél itt inkább az elvek ismertetése.

3.2.2. Számlálós címregiszterű mikroprogramozott vezérlő

Ez a típus nagyon hasonlít a számláló típusú (fázisregiszteres) vezérlőhöz. A blokkvázlatát a 3.7. ábra mutatja. Itt a címlogikát és címregisztert egy számláló valósítja meg. Az *i*-edik állapotban, ha a feltételrész által kiválasztott feltétel teljesül, akkor a számláló lép, vagyis a következő mikroutasítást címzi meg. Ellenkező esetben a mikroutasítás címrészében megadott mikroutasításra ugrik. A feltétel nélküli ugrást és továbblépést az 1 konstans feltétel kiválasztásával lehet elérni (lépés esetén a következő címre ugrunk). Előnye, hogy kellemes programozni, és viszonylag rövidek a mikroutasítások.

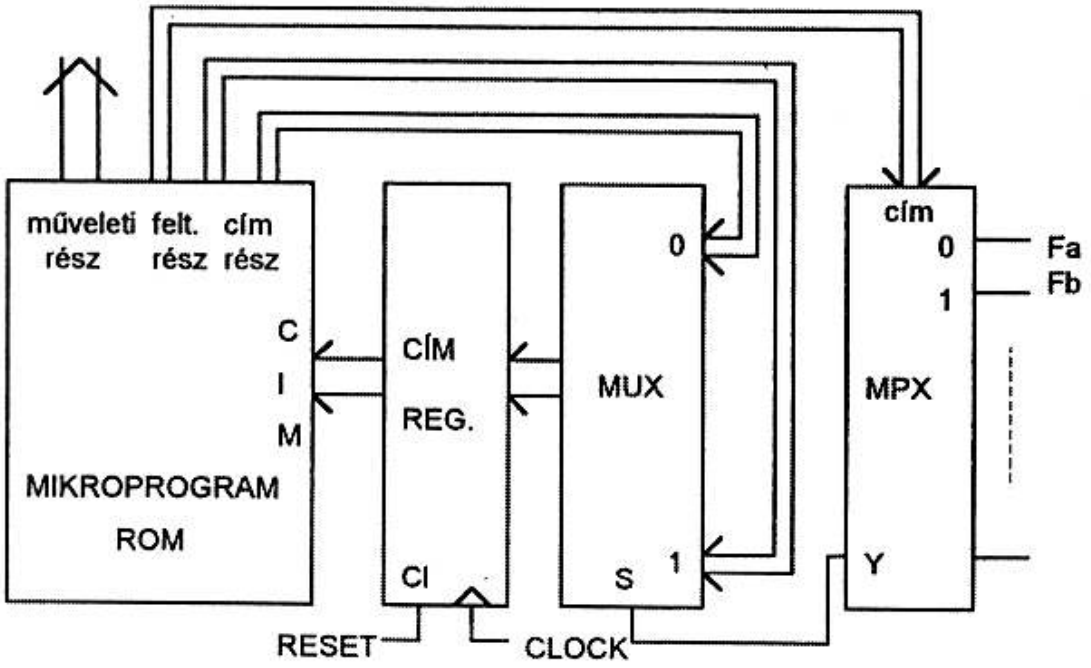


3.7. ábra. Számlálós címregiszterű mikroprogramozott vezérlő

3.2.3. Két címrészből választó mikroprogramozott vezérlő

Ennek címrésze két részre van osztva. Az adott állapotban a kiválasztott feltétel dönti el, hogy a kettő közül melyik íródik be a címregiszterbe (3.8. ábra). Feltétel nélküli elágazást a két címrész azonossága esetén kapunk.

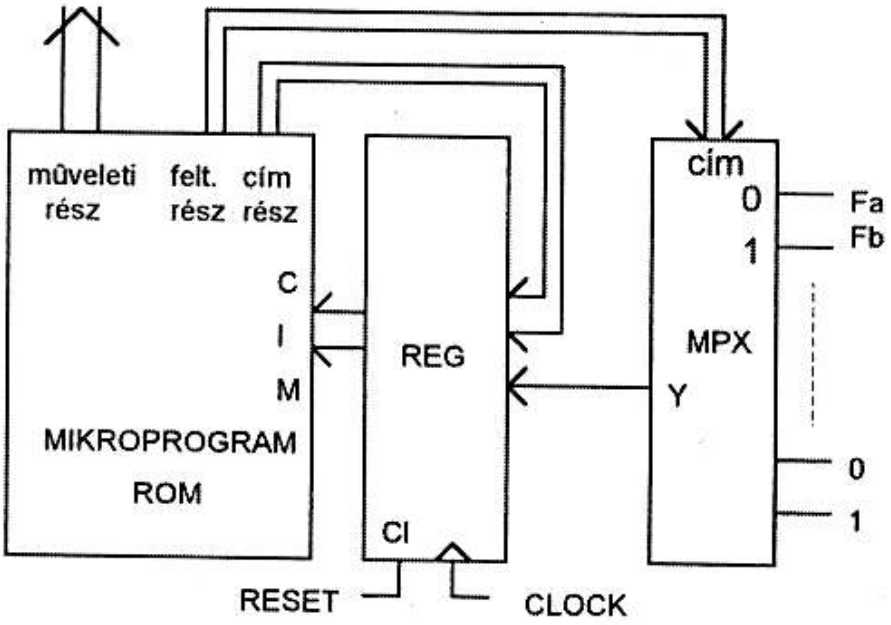
Hátránya, hogy mivel a címrész mindkét fele annyi bitből áll, mint a címregiszter, ezért széles mikroutasítást igényel. Ezen úgy lehet segíteni, ha a multiplexer bemenetére adott egyik címet a címregiszter aktuális tartalmának és az egyik címrésznek az előjeles összeadásával képezzük. Ekkor a feltételtől függően vagy abszolút ugrást hajt végre a vezérlő, vagy az aktuális mikroutasítás címéhez képest relatív ugrást. A relatív ugrás tartományának korlátozásával mikroutasítás biteket spórolhatunk meg. A relatív címet kettes komplemensben célszerű tárolni. Ennek a típusnak is előnye, hogy kényelmes hozzá mikroprogramot írni.



3.8. ábra. Két címrészből választó mikroprogramozott vezérlő

3.2.4. Feltételt címbe másoló mikroprogramozott vezérlő

A harmadik típusú cím előállítási módszernél a mikroutasítás címrésze egy bittel rövidebb, mint ami a ROM címzéséhez szükséges. A legkisebb helyiértékű bit helyére a kiválasztott feltételt másolják be. Így az adott állapotból a címrész által kijelölt két cím közül a feltételtől függően vagy a páros, vagy a páratlan címűre adódik a vezérlés (3.9. ábra). Itt is lehetőség van feltétel nélküli elágazásra, a konstans 0 ill. 1 kiválasztásával. Ennek a típusnak előnye az egyszerű felépítés, viszont a programozása elég kényelmetlen.



3.9. ábra. Feltételt címbe másoló mikroprogramozott vezérlő

A mikroprogramozott vezérlők sebesség növelésének a ROM szab korlátot. A nagyobb méretű ROM-okat a hasonló méretű RAM-okkal összehasonlítva, az utóbbiak jóval gyorsabbak. Ezért nagy állapotszám és sebességigény esetén a ROM helyett RAM-ot alkalmaznak, de ekkor a struktúra bonyolódik, hiszen meg kell oldani, a mikroprogramot tartalmazó ROM adatainak a RAM-ba történő átmásolását is. A mikroprogramozott vezérlő jelentősége régebben abban állt, hogy egy esetleges módosítás esetén többnyire csak a mikroprogram ROM tartalmát kell megváltoztatni, szemben a huzalozott logikával, ami részben áttervezést igényel. Ma már ezt az előnyt, s így a mikroprogramozott vezérlők jelentőségét csökkentik a nagy bonyolultságú újraprogramozható eszközök, az FPGA-k. A mikroprogramozott vezérlő struktúrát, a könnyű tervezhetőség miatt azonban egy FPGA belsejében is célszerű lehet létrehozni, külső ROM-ot alkalmazva (nagyméretű mikroprogram esetén), vagy azt FPGA belsejében kialakítva.

3.3. Órajelezési technikák, engedélyező és működtető jelek előállítása

Az adatstruktúra működtetése során alkalmazandó jeleket a felhasználás szempontjából két csoportra oszthatjuk.

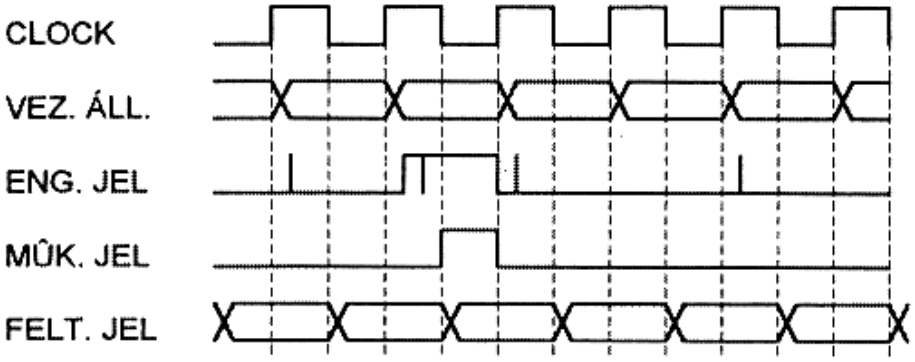
Az egyik csoportba azok a jelek tartoznak, amelyeknél nem zavaró a hazárd jelenléte, (pl. multiplexerek címzése, számláló engedélyezése stb.), ezeket **engedélyező típusú jel**eknek szokás nevezni.

A másik csoportba azokat a jeleket sorolják, amelyeknek feltétlenül hazárdmentesnek kell lenni (órajel bemenetek, statikus törlő ill. preset jellegű bemenetek). Az utóbbi csoport neve **működtető típusú jel**, avagy előállításukra célozva **kapuzott órajel**.

Az alábbiakban kétféle technikát mutatunk a fenti jelek előállítására, az egy és a kétfázisú órajelezést.

3.3.1. Egyfázisú órajelezési technika

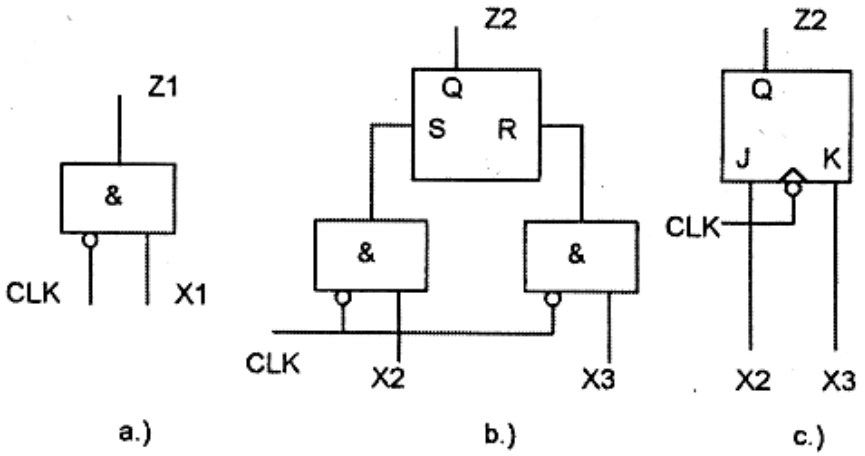
Ennél a technikánál a vezérlő és az adatstruktúra működtetése ugyanazon órajel fel- ill. lefutó élére történik. Példánkban (3.10. ábra) a vezérlő a felfutó, az adatstruktúra a lefutó élre működik.



3.10. ábra. Egyfázisú órajelezési technika jellemző idődiagramja

Az engedélyező jel tkp. a vezérlő egy vagy több állapotát kikódoló kombinációs hálózattal állítható elő. Számláló típusú vezérlő esetén, ha a jel több állapotban is aktív, akkor ezen állapotok kikódoltját VAGY kapcsolatba kell hozni. Mikroprogramozott vezérlőnél is tkp. ez történik, de itt a függvényt a ROM mint univerzális kombinációs hálózat állítja elő, nincs szükség külön VAGY kapura. Az így előállított jel hazárdos, hiszen a kombinációs hálózat bemenetén több Hamming-távolságú a változás, mert nem biztosított az egymást követő állapotok szomszédossága. Mikroprogramozott vezérlő esetén pedig a ROM kimenete annak belső felépítése miatt hazárdos.

A működtető jelet - a vezérlőt működtető órajel élet tekintve az órajel periódus kezdetének, - a periódus második felének és az adott állapotot kikódoló jelnek az ÉS kapcsolatával hozzuk létre. A fenti esetben a függvény: $ENG_i \bullet \overline{CLOCK}$ (3.11a ábra).



3.11. ábra. Hazárdmentes jelek előállítása

Hazárd, amint az idődiagramból látható, akkor jönne létre, ha az órajel negáltja beleérne az állapotot dekódoló jel megszűnő szakaszába. Ez azért nem fordul elő, mert az állapotváltás, s így a dekódoltjának változása is, az órajel hatására történik, viszont az állapottároló és a dekódoló hálózat együttes késleltetése több, mint az órajel késése, a negálást is figyelembe véve. A helyzet még kedvezőbbé tehető, ha az órajel és negáltja közötti késleltetést minimálisra csökkentjük. Ezt úgy érhetjük el, ha az órajeleket egy T flip-flop ponált és negált kimenetére kapcsolt meghajtókról vesszük, vagy a ponált órajel útjába egy nem invertáló meghajtót teszünk, hogy kiegyenlítsük az inverter késleltetését. A gyakorlatban, az egyszerű előállíthatóság miatt általában 1/2 kitöltési tényezőjű jelet használnak. (A periodikus jel kitöltési tényezője a jel magas szintjének aránya a periódusidőhöz.)

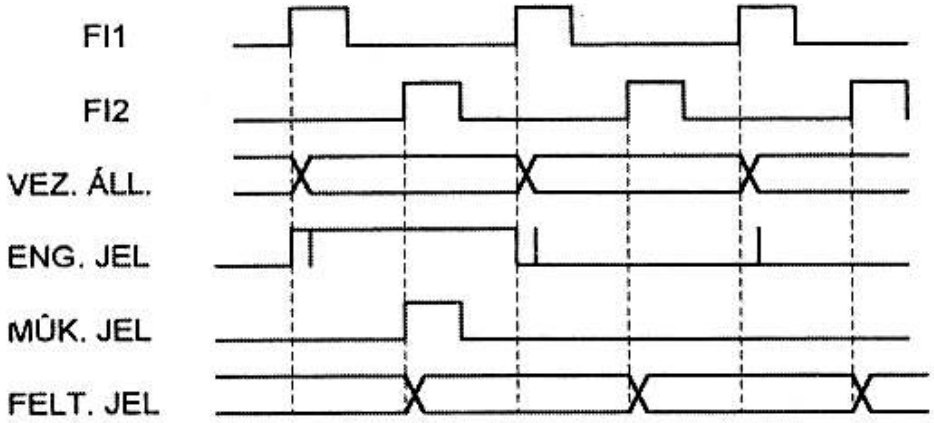
Ha olyan jelre van szükség, amely több egymást követő állapoton keresztül aktív, akkor azt SR vagy JK flip-floppal célszerű megoldani, főként, ha a hazárdmentesség is igény. (Annyi bemenetű VAGY kapuval is megoldható, ahány állapotban aktivizálandó a jel, de ez sok állapot esetén költséges, s ha még hazárdmentesíteni is kell, akkor egy D flip-floppal mintavételezni szükséges.) A 3.11b ábra az SR flip-flopos, a 3.11c ábra pedig JK flip-flopos megoldást mutat. Az X2 jel hatására 1-be, az X3 hatására pedig 0-ba billen a flip-flop. A 3.11b ábra szerinti megoldás hazárdmentesíti az X2 és X3 jeleket, s ezt a hazárdmentes jelet vezeti a flip-flop statikus törlő ill. preset bementére. A 3.11c ábra

szerinti megoldásnál a szinkron JK flip-flop negált órajel éllel mintavételezi az X2 ill. X3 jeleket, amikor azok már stabilak.

A feltételjeleket célszerű az adatstruktúrát működtető jellel mintavételezni, így a feltételjel változása és a vezérlő állapotváltozása között megfelelő előkészítési idő marad. Ezt az időt befolyásolja az órajel kitöltési tényezője. 1/2-nél nagyobb kitöltési tényező esetén nő az előkészítési idő, viszont csökken a hazárdmentes impulzus szélessége.

3.3.2. Kétfázisú órajelezési technika

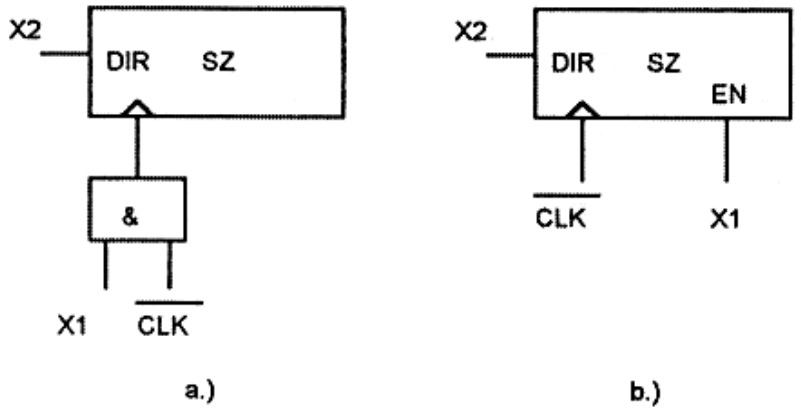
Ebben az esetben (3.12. ábra) két, egymáshoz képest 180 fokkal eltolt fázisú órajelet alkalmazunk (FI1, FI2), melyek kitöltési tényezője 1/4. (Ha a kitöltési tényező 1/2 lenne, akkor tulajdonképpen az egyfázisú órajelezéshez jutnánk vissza, hiszen FI1 negáltja lenne FI2.) A vezérlőt a FI1-el, az adatstruktúrát a FI2-vel működtetjük. Az igazi különbség a működtető jel kialakításában van, amelyet $ENG_i \bullet FI2$ szerint állítunk elő. Ez a jel a vezérlő állapotváltásának közepén van, így a hazárdmentesség még nagyobb biztonsággal teljesül, az ilyen rendszer nagyobb órajel csúszást visel el, mint az egyfázisú órajellel működtetett, ezért főként nagy ill. bonyolult rendszerek esetén célszerű alkalmazása.



3.12. ábra. Kétfázisú órajelezési technika jellemző idődiagramja

Az órajelezési technikák lényegének jobb megértése végett tekintsük azt az egyszerű esetet, amikor egy vezérlővel annak valamely állapotában egy szinkron fel-le számlálót akarunk 1-el felfele léptetni, s a következő állapotban elágazási feltételként akarjuk használni annak állapotát. Erre több megoldás is elképzelhető. Minden esetben először be kell állítani a számlálási irányt. Ezt az adott állapotban előállított engedélyező jellel kell megtennünk, mert be kell tartanunk a számlálási irány és órajel között előírt előkészítési időt, s ezért a lehető leghamarabb előálló jelet kell alkalmazni. Ezután léptetni kell a számlálót. Az adatstruktúrát felépíthetjük úgy, hogy egy működtető jelet kapcsolunk a számláló órajel bemenetére. Ha nem ismernénk az órajelezési technikákat, akkor is előbb utóbb rá kellene jönnünk, hogy az órajel ponáltját felhasználva, csak nehézkesen tudunk hazárdmentes órajelet produkálni (akkora késleltetést kellene beiktatni az órajel útjába, ami biztosítja, hogy az engedélyező jel már nem hazárdos, amikor a késleltetett órajel megérkezik, sőt az előkészítési időnek is le kell telni), vagyis rá vagyunk kényszerítve az

$ENG_1 \bullet \overline{CLOCK}$ megoldásra (3.13a ábra). Ennek a megoldásnak hátránya, hogy az így előállított jel késik az eredeti órajelhez képest, továbbá egy külön kapu szükséges a megvalósításához.

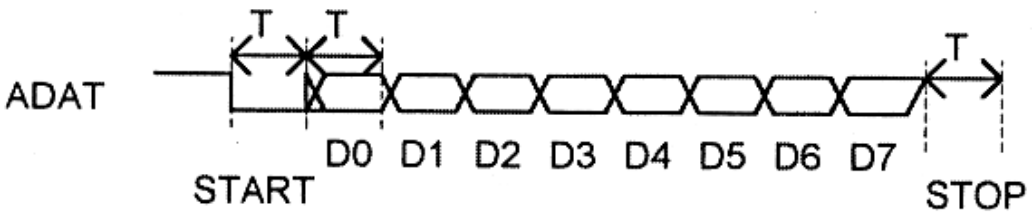


3.13. ábra. Fel-le számláló léptetése egyfázisú órajelezés esetén

Az adatstruktúrát felépíthetnénk úgy is, hogy a számláló órajel bemenetére a vezérlő órajelét (CLOCK) kapcsoljuk, s az engedélyező jelet használjuk a számláltatásra. Ebben az esetben azonban a léptetés csak a következő órajel hatására történik meg. Ráadásul a megoldás érzékeny az órajel csúszásra, hiszen a számláló engedélyezése és órajele veszélyesen közel esik egymáshoz. Így itt is arra kell rájönnünk, hogy a vezérlőhöz képest negált órajellel kell működtetnünk a számlálót, mert ekkor kb. fél órajelnyi időtartalék marad az engedélyezés és az órajel között, ráadásul ugyanennyi idővel hamarabb lép a számláló (3.13b ábra). Bonyolult adatstruktúra esetén 2-nél több fázisú órajelet is ki lehet alakítani.

3.4. Mintapélda funkcionális elemekkel való tervezésre

A mintapéldában egy aszinkron soros vevőt fogunk megtervezni. Az adó a 3.14. ábra idődiagramja szerint az adatvezetéken sorosan adja az adatokat.



3.14. ábra. Aszinkron soros átvitel idődiagramja

Az adatvezeték alapállapotban magas szinten van. Az adatátvitel a START bittel kezdődik (T ideig 0 szint, erre vár a vevő). Ezután jönnek az adatbitek D0-D7

sorrendben (T időnként), majd a STOP bit, amely legalább T órajelnyi hosszúságú, s újabb átvitelig ez marad az adatvezetéken.

A feladat olyan aszinkron soros vevő komplett funkcionális blokkvázlatának a megtervezése, amely az elmondott formátumban érkező adatokat venni képes, az összes bit beérkezése után átírja az adat byte-ot egy regiszterbe és egy RDY vonalon jelzi, hogy érvényes adat van a regiszterben. Az áramkör figyeli a STOP bitet is, és ha nem 1 értékű, akkor keret hibát jelez az FERR vonalon. Az RDY és FERR jelet egy külső CL jellel lehet törölni. Az áramkör az adatbitek regiszterbe való átírása után újabb adatot képes fogadni a soros vonalon.

A feladat részletes megfogalmazása, a szükséges funkcionális elemek meghatározása

a. Kezdetben (bekapcsolás után) az RDY jelet 0-ba kell állítani, ill. a vevőnek alaphelyzetbe kell állnia. Ezt bekapcsoláskor rövid ideig (néhány msec) aktivizálódó RESET jellel biztosíthatjuk. A RDY jel előállításához szükség van egy flip-flopra.

b. A vevőnek természetesen ismernie kell a bitidőt, hiszen enélkül nem képes értelmezni az adatokat. Az átvitel sebességét bit rate-nek nevezik, és értékét bit/sec-ban adják meg. (A gyakran használt Baud rate a jelváltási sebesség, ami ebben az esetben megegyezik a bit rate-tel.) A gyakorlatban az adási és a vételi sebességet az átvitel megkezdése előtt azonos Baud rate értékre állítják. Mivel az adó és a vevő külön órajel generátorral rendelkezik, ezért ezek frekvenciája egy kicsit mindig eltér egymástól. Egy-egy byte vétele során az eltérés nem vezethet bittévesztéshez. Ezért órajel generátorként a nagy frekvencia stabilitási igény miatt kvarc oszcillátort használnak.

c. A vevő várja a START bit megérkezését, vagyis az adatvonalat elég sűrűn (szokás szerint minimálisan 16-szoros frekvenciával) kell mintavételeznie. Tehát a kvarc oszcillátoros óragenerátor kimenetén a frekvencia 16/T. A vezérlésnek figyelnie kell, hogy mikor lesz először 0 az adatvonal.

d. Ha megjött a START bit, akkor fél bitidőt célszerű várni, hogy kb. a bitek érvényességének közepére találjon a mintavételezés. Ez szintén a bittévesztés elkerülését szolgálja. Ettől az időponttól kezdve T időnként 9-szer mintavételezni kell az adatvonalat és tárolni kell az adatbiteket. A bitidő T/16 periódusidejű órajelből történő előállítása végett szükség van egy 16-os számlálóra. A bitek számlálásához egy legalább 10-es számlálót használunk (alapállapot, és a bitek). Az adatbitek mintavételezéséhez és tárolásához egy 8 bites shiftregiszter szükséges.

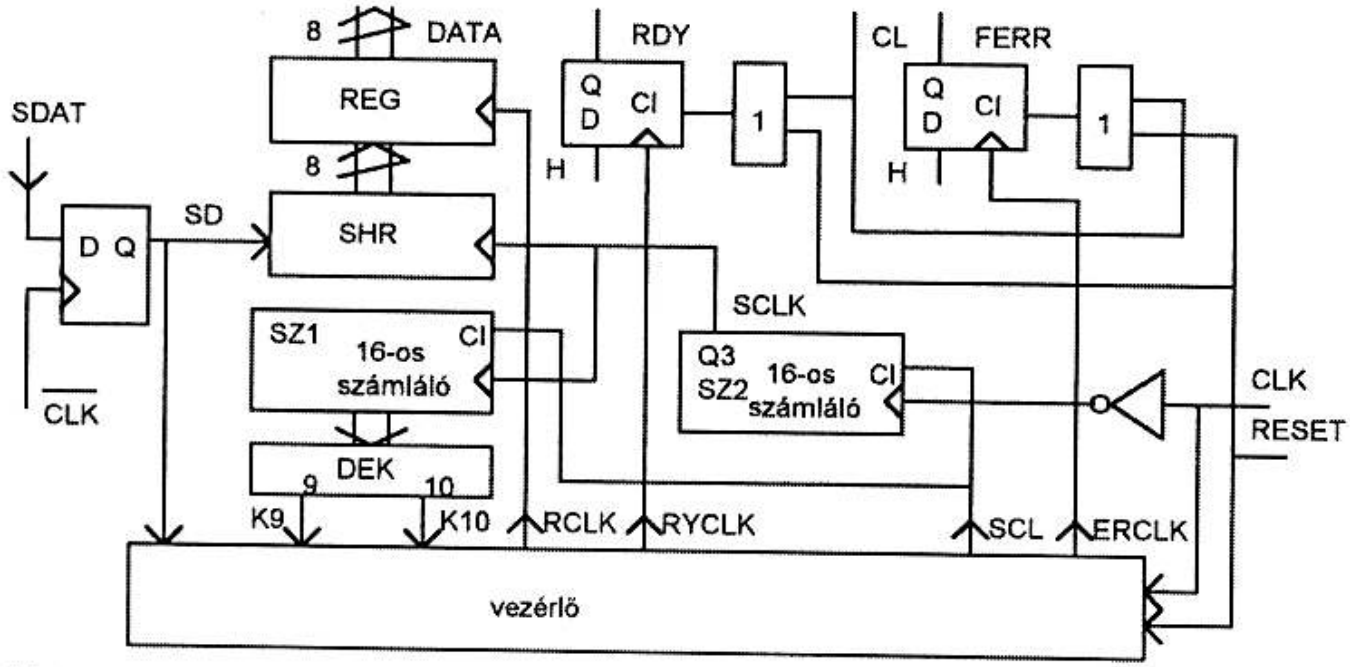
e. Az utolsó bit beérkezése után át kell írni a teljes bejött szót egy regiszterbe és a RDY vonalon jelezni. Ehhez 8 bites regisztert választunk.

f. Figyelni kell a STOP bitet, és jelezni, ha hibás az értéke. Az FERR jel előállításához flip-flopot használunk.

A feladatot adatstruktúra-vezérlő szemlélettel oldjuk meg. Egyfázisú órajelezést alkalmazunk, mivel a feladat viszonylag egyszerű.

A részletes adatstruktúra felrajzolása

Ez intuitív lépés, sok gyakorlást igényel. A felrajzolás (3.15. ábra) után célszerű részletesen végiggondolni, hogy miként fog működni az áramkör.



Megjegyzés: a számlálók szinkron törlésűek (CI)

3.15. ábra. Aszinkron soros vevő adatstruktúrája

A vezérlő alapállapotban az SDAT jelet figyeli, s mindaddig alapállapotban marad és az SCL jellel folyamatosan törli a számlálókat, amíg az 1-es értékű. Az SDAT jelet egy D flip-flopba mintavételezzük (külső aszinkron jel), a vezérlőt működtető CLK negáltjával (egyfázisú órajelezési technika). Amint SDAT 0-vá válik, akkor a vezérlő megszünteti a számlálók törlését, s ezzel kezdetét veszi az adatok mintavételezése a shiftregiszterbe. Mivel a vezérlő a T/16 periódusidejű CLK órajellel működik, ezért egy feltételjel aktívvá válása után maximum T/16 idő alatt reagál. Így a bitek mintavételezése kicsit késni fog az ideálishoz képest (flip-flopba mintavételeződik a START bit, vezérlő észleli és megszünteti a törlést, számláló elkezd számolni), azonban ez a késés 2T/16 és T/16 közé esik (3.18. ábra), ezért ha megfelelően pontos az adó és vevő oszcillátora, ez nem okoz gondot. (A késleltetés korrigálható, pl. ha az SZ2 számlálót törlés helyett 1-el feltöltjük.)

Az adatstruktúrában a bitszámláló (SZ1) és a soros adatvonalat mintavételező shiftregiszter (SHR) közös órajelet kap. Az ezeket működtető órajelet (SCLK) a T/16 periódusidejű CLK jel leosztásával állítjuk elő (SZ2 számláló). A START bit kezdete utáni fél bitidőnyi késleltetést az biztosítja, hogy az SCK órajelet az SZ2 számláló MSB-jéről (Q3) vesszük, ami a kezdeti törölt állapot után 8T/16 idő múlva adja az első felfutó órajel élet (0000-tól 1000-ig elszámolva telik el a fél bitidő).

A bitszámlálóra kapcsolt dekóder K9 kimenete jelzi a vezérlőnek, hogy bejött az utolsó adatbit, ezután írja át a vezérlő az adatot a shiftregiszterből a regiszterbe az RCLK jellel. A dekóder K10 kimenete jelzi a vezérlőnek, hogy bejött a STOP bit, ezután

ellenőrzi a vezérlő, hogy 1 értékű-e, s ha nem, akkor kiadja az ERCLK jelet és vár hogy 1 értékű legyen. (Ha a STOP bit hibás, akkor esély van arra, hogy hosszabb ideig 0 értékű az adatvonal, pl. 0-ba ragadás esetén. Ebben az esetben viszont hibásan START bitet érzékelhetne az áramkör, ha a hibás STOP bit megszűnését nem várná meg. Persze a STOP bit eltérő Baud rate beállítás következménye is lehet, ilyenkor nem védhető ki, hogy egy 0 értékű adatbitet START-nak érzékeljen az áramkör.)

A végén a vezérlő az RYCLK jellel bebillenti az RDY jelet, majd újra alaphelyzetbe áll, törli a számlálókat. A számlálóként szinkron törlésűeket választottunk, mert így nem kellett hazárdmentesen előállítani a törlő jelet.

A vezérlő nem figyeli az RDY megszűnését, vagyis egy adat vétele és regiszterbe írása után azonnal képes venni a következőt. Akkor van baj, ha a következő bejött adat átírja az előzőt, mielőtt azt kiolvasnák. Megoldható lenne, hogy ebben az esetben egy másik hiba jelző flip-flop mutassa a felülírás hibát. Bizonyos mértékig maga a felülírás hiba is kivédhető lenne, ha egyetlen buffer helyett, egy megfelelő méretű FIFO-t alkalmaznánk. (A FIFO olyan tároló, amelyből az először beírt adat olvasható ki először.) Ebben az esetben a soros vevő a FIFO-ba annyi adatot írhat, amennyit annak a mérete megenged, a másik oldalról pedig az adatok a beírás sorrendjében kiolvashatók. A valóságos soros vevőkben mindezek a funkciók (és még sok egyéb) egyetlen chipben megtalálhatók.

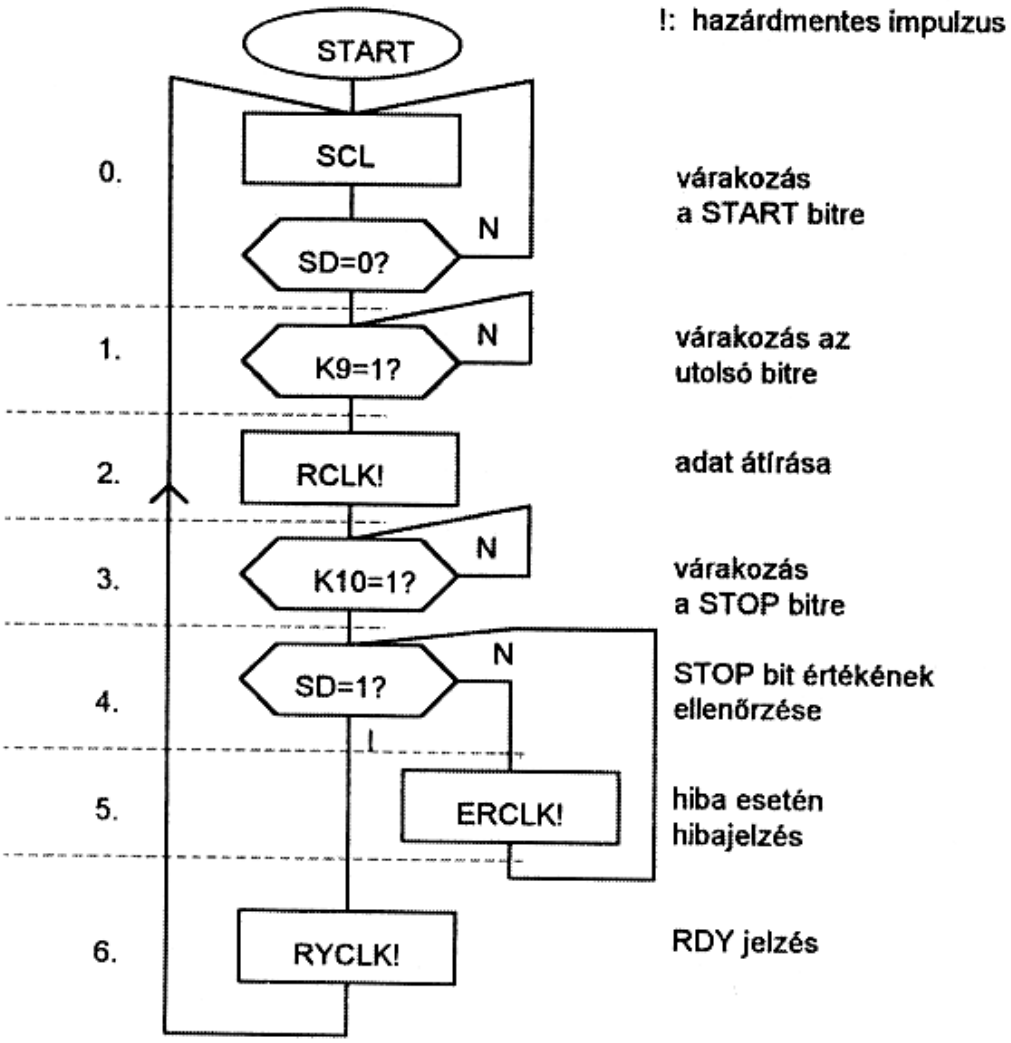
A vezérlő folyamatábrájának felrajzolása

Ha részletesen végiggondoltuk az adatstruktúra működését, akkor a folyamatábra felrajzolása tkp. a meggondolásaink, vagyis az adatstruktúrát működtető algoritmus formalizált, grafikusan megfogalmazott leírását jelenti (3.16. ábra).

A folyamatábra **állapot boxokból** és **döntési boxokból** áll. Az állapot boxba írjuk be az adott állapotban kiadandó vezérlőjeleket, a döntési boxba pedig az adott állapotban figyelt feltételt. A vezérlőjeleknél célszerű megkülönböztetni az engedélyező típusú jeleket a működtető típusú (kapuzott órajel) jelektől. Itt a működtető jeleket a jelnév után írt felkiáltójellel különböztetjük meg.

A folyamatábra egy-egy állapot boxjába csak azokat a jeleket tüntetjük fel, amelyek aktívak, a többit inaktívnak tekintjük. (Olyan eset is lehetséges, hogy a fel nem tüntetettek között van olyan, amely közömbös.) A folyamatábrán be kell jelölni a vezérlő állapotait is.

Felrajzolás után célszerű ellenőrizni, hogy minimális állapotszámmal oldottuk-e meg a feladatot. Itt főként annak ellenőrzésére gondolunk, hogy az egymás után kiadott vezérlőjelek nem adhatók-e ki egy állapoton belül. Ezt persze a kimenet kódolásával összhangban kell vizsgálni. (Vertikális kódolásnál pl. egyszerre csak egy vezérlőjel lehet aktív.)

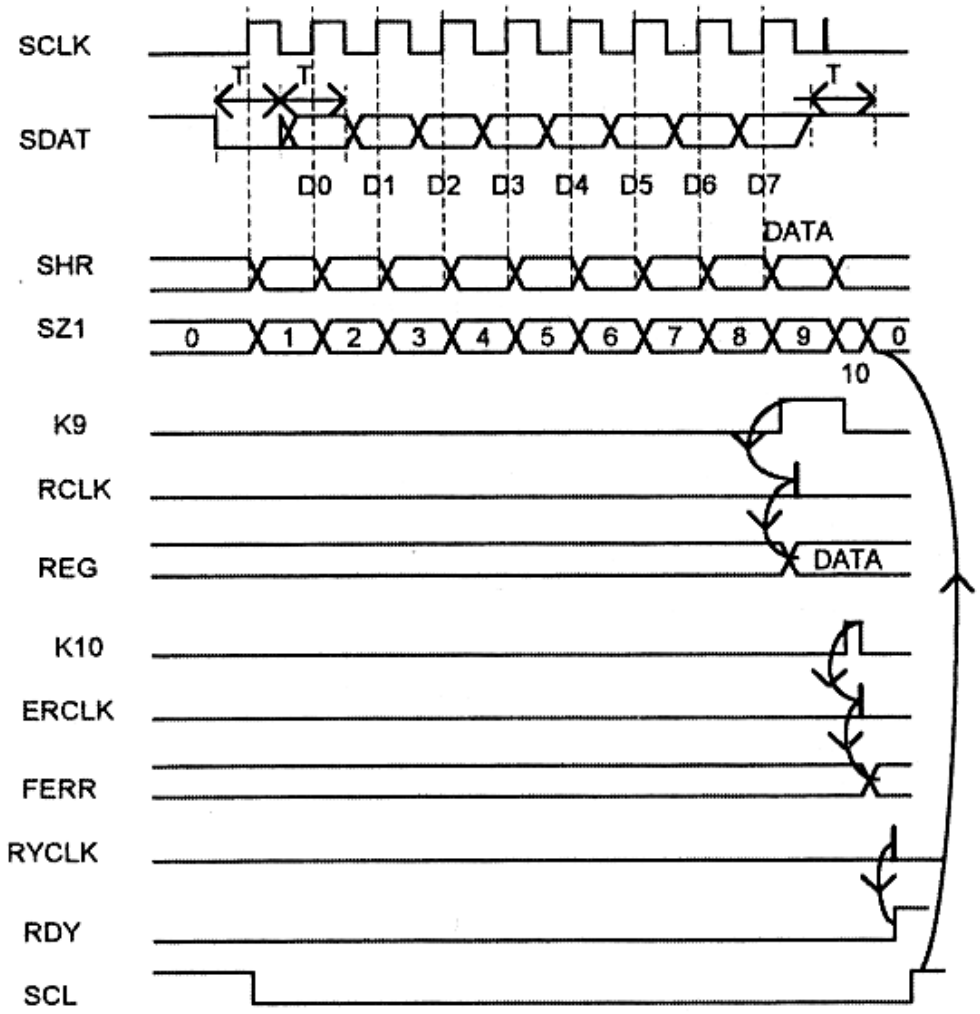


3.16. ábra. Aszinkron soros vevő vezérlőjének folyamatábrája

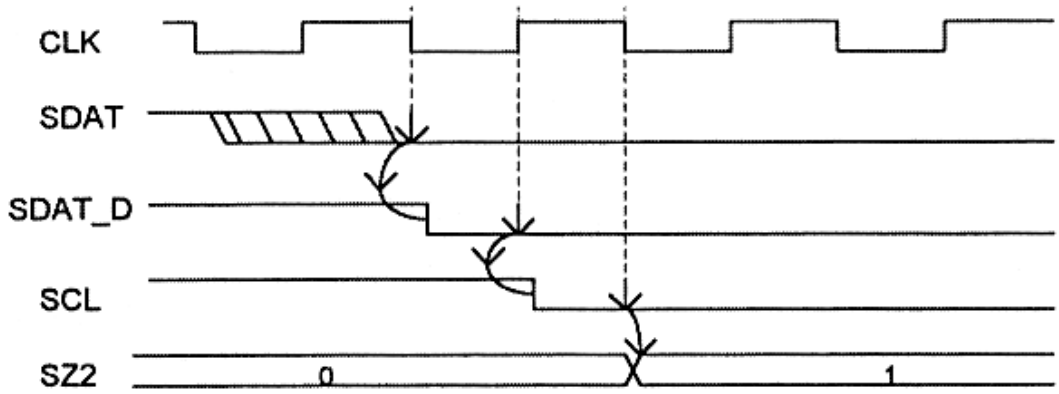
Szimuláció

A megtervezett vezérlő működését célszerű szimulálni és idődiagramon ellenőrizni (3.17. ábra). CAD rendszerek használata esetén az idődiagramot a szimulátor alkotja meg, a tesztvektorok és a teljes hálózat leírása (kapcsolási rajz, HDL nyelv) alapján. Azonban sok esetben már a feladat végiggondolása közben érdemes kézzel idődiagramot rajzolni, mert az segíti a feladat részproblémáinak felszínre hozását.

Mivel a vezérlő órajele (CLK) az SCLK jelnél 16-szor nagyobb frekvenciájú, ezért azt már nem volt lehetséges ábrázolni. Hasonló okból az idődiagram végén lezajlódó folyamatok nem egészen méretarányosak. Az idődiagramon a fontosabb helyeken íves nyilakkal jeleztük az ok-okozati összefüggéseket. Pl. K9 magas szintjét érzékelve adja ki a vezérlő az RCLK jelet, RCLK felfutó élének hatásra íródik be a regiszterbe az adat.



3.17. ábra. Aszinkron soros vevő működésének idődiagramja



3.18. ábra. Késleltetés SDAT érzékelésétől SZ2 elindulásáig

A vezérlőt kétféle módszerrel valósítjuk meg. Először lép vagy ugrik fajtájú számláló típusú vezérlővel, majd mikroprogramozottal.

Aszinkron soros vevő vezérlése számláló típusú vezérlővel

A vezérlő funkcionális blokkvázlatát mutatja a 3.19. ábra. A megtervezéséhez segítséget nyújt, ha elkészítjük a 3.2. táblázatot, mert a szükséges információt könnyen kiolvashatjuk belőle.

állapot	figyelt feltétel	köv. állapot, ha a feltétel 0	köv. állapot, ha a feltétel 1	ugrási kód c,b,a	kimenet
0.	SDAT	1. lép	0. ugrik	000	SCL
1.	K9	1. ugrik	2. lép	001	
2.	-	3.lép	3. lép	---	CLKR!
3.	K10	3. ugrik	4. lép	011	
4.	SDAT	5. lép	6. ugrik	110	
5.	-	-	4. ugrik	100	ERCLK!
6.	-	0. ugrik	-	000	RYCLK!

3.2. táblázat

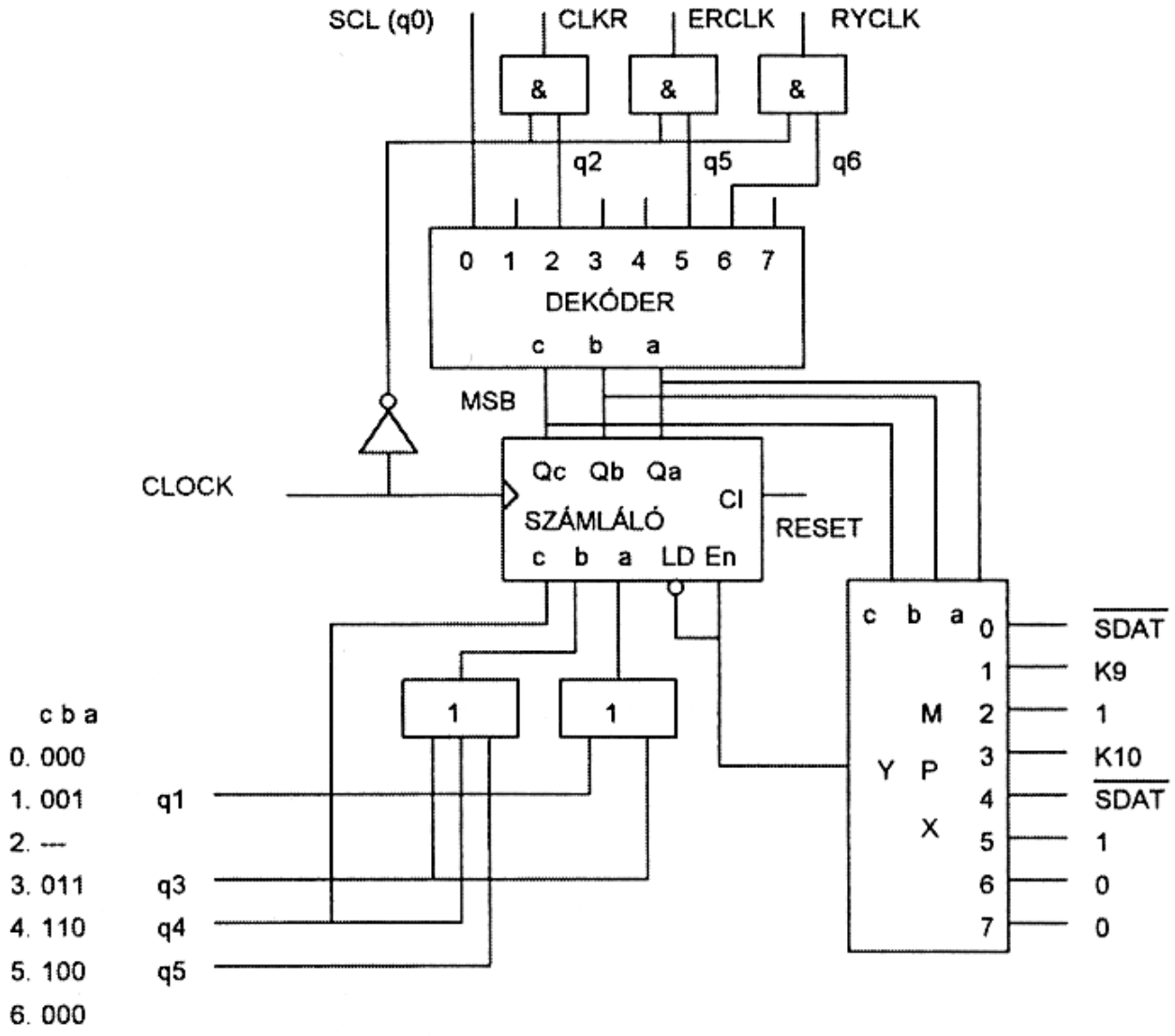
A folyamatábra 7 állapotot tartalmaz, ez meghatározza a vezérlő számlálójának, dekóderének és feltétel multiplexerének a méretét. Mivel a legközelebbi nagyobb standard méret MSI dekóder és multiplexer esetén 8, ezért 3/8-as dekódert 8/1-es multiplexert használunk. Számlálóknál a legközelebbi standard méret 10, ezért 10-es vagy 16-os számlálót alkalmazhatunk. (Itt nem probléma a kívántnál nagyobb modulus, mert a vezérlő a 6-os állapot fölé nem engedi számolni.) A számláló esetén nem tüntettük fel a rajzon a legnagyobb helyiértékű bitet. (Manapság, ha valaki esetleg ilyen vezérlő alkalmazása mellet dönt, akkor célszerűen programozható eszközből alakítja ki, s így természetesen nincs a standard MSI elemekhez kötve, ezért kevésbé redundáns megoldást készíthet.)

Elsőként a feltétel multiplexer *i*-edik bemenetére bekötjük az i-edik állapotban figyelt feltételjelet (0, 3, 4 állapotok). Az SDAT-ot negálva kell bekötni a 0-dik és 4-edik bemenetre, mert mindkét esetben SDAT=1 esetén kell ugrani, a vezérlő adott kialakításban pedig 0 bemeneti feltételre ugrik. A bemenetekre feltétel nélküli lépés esetén 1-et (2, 5 állapotok), feltétel nélküli ugrás esetén 0-át (6 állapot) kell kötni. A feltétel multiplexer 7-es bemenetére tetszőleges logikai szintet köthetünk, ha véletlenül előáll ez az állapot, mindenképpen a 0-ban folytatódik.

Ezután a kimeneti jeleket állítjuk elő. Az SCL jelet a 0-adik állapotban kell kiadni, s hazárdos lehet (mivel a számlálók szinkron törlésűek), ezért közvetlenül a dekóder 0-ás

kimenete állítja elő. A CLKR, RCLK és RYCLK impulzusok hazárdmentes előállítást igényelnek, ezért ezeket az egyfázisú órajelezésnél tanultak szerint kapuzzuk az órajel negáltjával. A példában nem szerepel olyan jel, amely több állapotban is aktív, ezért nincs szükség a VAGY kapus előállításra.

Az ugrási címet előállító hálózat felrajzolásához segítségképpen elkészítettük a 3.19. ábrán egy táblázatot is, mely mutatja, hogy mely állapotból milyen kódúba kell ugrani, ha a feltétel nem teljesül. Pl. A 4. állapotból a $QcQbQa$=110 állapotúba, ezért q4-et (ez a jel csak a vezérlő 4-es állapotában aktív) a számláló *c* és *b* bemenetére kapcsolt VAGY kapura kell kötni, mert a következő állapot kódjában itt szerepel 1-es. Az összes VAGY kapu bemenet bekötése után kiderül, hogy a c bemenetre kapcsolódó kapura csak egyetlen jel kapcsolódna, ezért ez a kapu elhagyható.



3.19. ábra. Soros vevő számláló típusú vezérlője

Aszinkron soros vevő vezérlése mikroprogramozott típusú vezérlővel

Számlálós címregiszterű mikroprogramozott vezérlőt választottunk a megvalósításhoz (3.20. ábra), mivel kisebb szószélességű ROM-ot igényel, mint a két címből választós, és kellemesebb programozni, mint a feltételt címbe másoló típust.

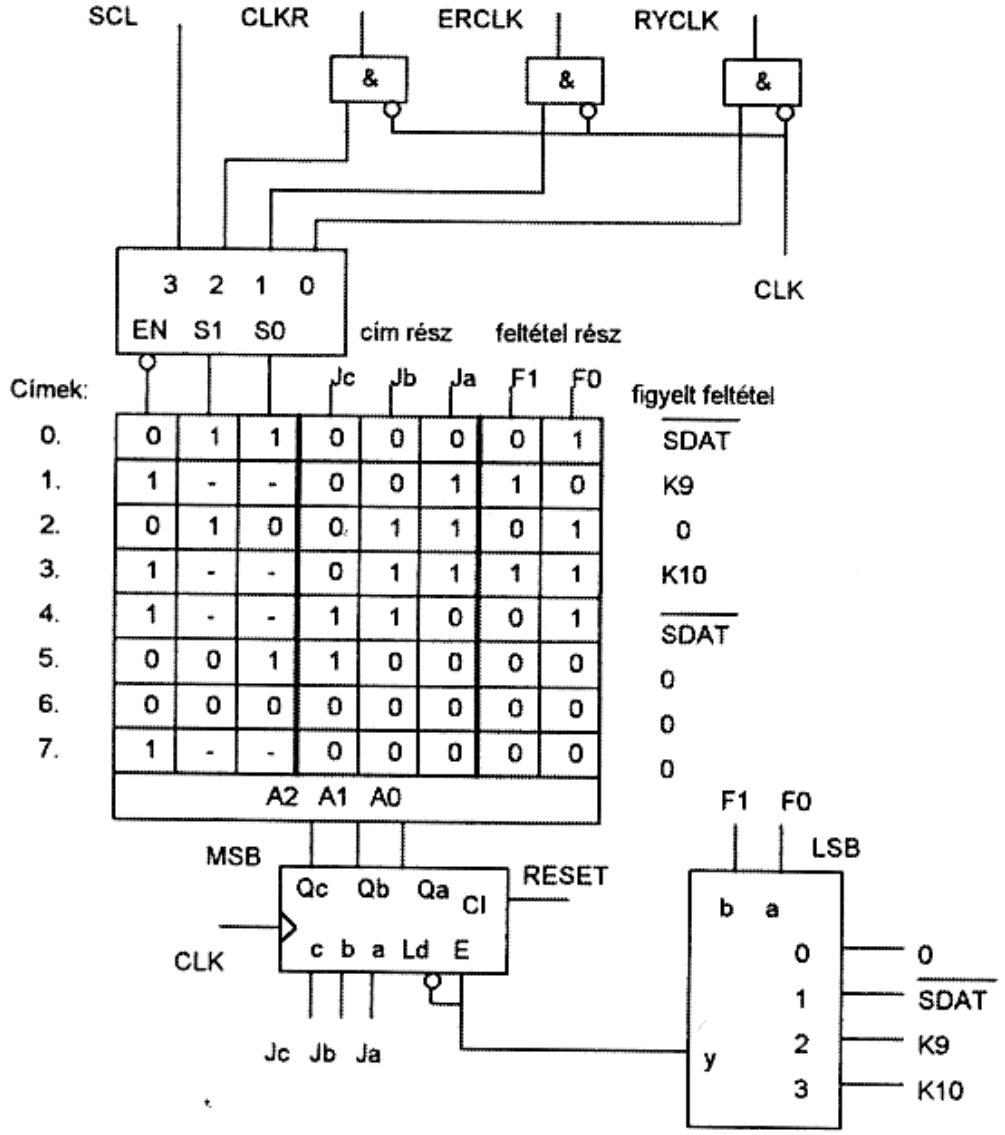
A számláló méretét itt is a folyamatábra állapotszáma határozza meg, mint az előbbi esetben.

A feltétel multiplexer méretét csak a figyelt feltételek (SDAT, K9, K10) száma befolyásolja, a feltétel nélküli eset miatt 1-el nagyobb méretű kell.

Minden feltételhez hozzárendelünk egy feltétel kódot (a feltétel multiplexer hozzá tartozó bemenetének címét):

feltétel nélküli ugrás: :00
$\overline{\text{SDAT}}$: 01
K9: 10
K10: 11

Mivel a folyamatábra olyan, hogy minden állapotban maximum egy jel aktív, ezért vertikális kódolású kimenetet alakítunk ki, vagyis dekóder állítja elő a kimeneteket. A dekóder mérete vagy 1-el nagyobb, mint a vezérlőjelek száma, mert van olyan állapot, ahol egyik vezérlőjel sem lehet aktív, s ekkor az egyébként nem használt kimenetet választjuk ki, vagy engedélyezhető dekódert választunk. Az utóbbi olcsóbb, mert így csak 2/4-es dekóderre van szükség, míg az előbbi esetben 3/8-as a legkisebb standard méret.



3.20. ábra. Mikroprogramozott vezérlős megvalósítás

Minden kimeneti jelhez hozzárendelünk egy kódot:

RYCLK	000
ERCLK	001
CLKR	010
SCL	011
tiltás	1 - -

A megfelelő kimeneteket (CLKR, ERCLK, RYCLK) a szokásos módon hazárdmentesítjük.

A mikroprogram utasítás felépítése: <vezérlőjel kódja><ugrási cím>< feltétel>

A mikroprogramot a folyamatábra alapján kódoljuk.

A példában a 0. állapotban a figyelt feltétel $\overline{SDAT}$, kódja: 01.
Az ugrási cím, ha a feltétel teljesül: 000.
A kiadandó jel SCL, kódja: 011.

Így a 0. címen elhelyezkedő mikroutasítás kódja: <011><000><01>

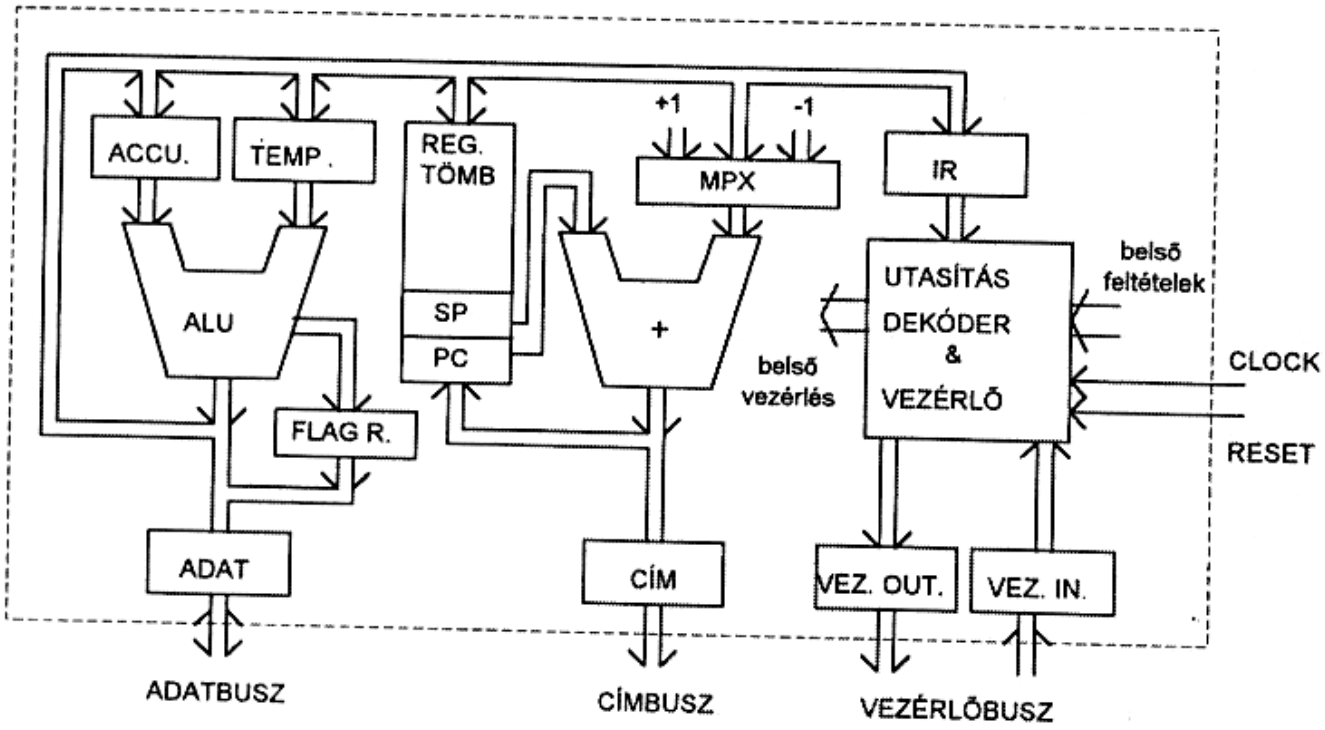
Egy általános célra kialakított mikroprogramozott vezérlő struktúrához a hatékony fejlesztés érdekében célszerű egy mikroprogram nyelvet kialakítani és fordító programot írni.

Az olvasónak javasoljuk, hogy gyakorlásképpen valósítsa meg más típusú vezérlőkkel is a 3.16. folyamatábrát.

4. A MIKROPROCESSZOR ÉS A MIKROPROCESSZOROS RENDSZER

Az előző fejezetben ismertetett vezérlőkkel a megoldható feladatok köre elég korlátozott. Felmerült az igény olyan univerzálisan használható vezérlőre, amely nagy tömegben és olcsón gyártható. Erre a célra fejlesztették ki a speciális adatstruktúrával kiegészített vezérlőket, a mikroprocesszorokat. A mikroprocesszor elődjének a korai számítógépek központi egysége (Central Processing Unit) tekinthető. Amikor a technológia képes lett, a számítógép központi egységének egyszerűsített architektúráját egyetlen chipen megvalósítani, akkor jött létre az első mikroprocesszor (μP, CPU). Manapság a nagyszámítógépek központi egységét is mikroprocesszor(ok) valósítják meg, ezeket a rendkívül bonyolult áramköröket szuper mikroprocesszoroknak nevezik. Itt csak az egyszerűbb, főként vezérlési feladatok megoldására alkalmazható CPU-kkal foglalkozunk.

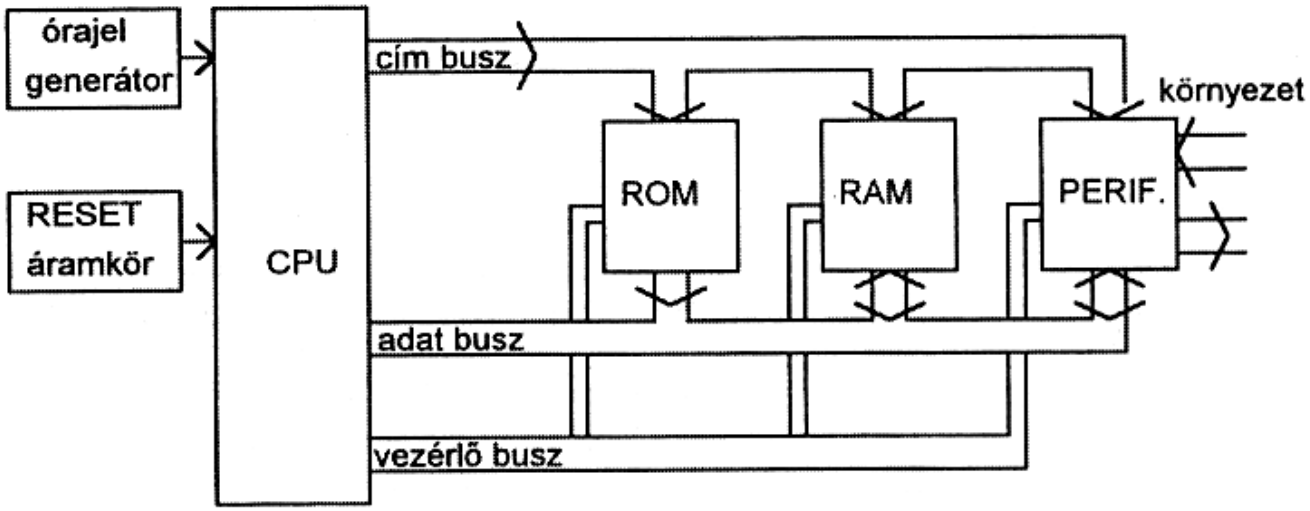
Egy egyszerű mikroprocesszor belső felépítését mutatja a 4.1. ábra.



4.1. ábra. Egyszerű CPU belső blokkvázlata

A mikroprocesszor önmagában nem használható, a működéséhez kiegészítő elemekre van szükség (ROM, RAM, perifériák, lásd: 4.2. ábra), azonban léteznek minden egységet egyetlen IC-ben integráló ún. mikrokontrolerek is (ezekről később szólunk).

Egy egyszerű μP-os rendszer felépítését mutatja a 4.2. ábra:



4.2. ábra. A mikroprocesszoros rendszer egyszerűsített blokkvázlata

A mikroprocesszor ill. μP-s rendszer főbb tulajdonságai, fontosabb fogalmak

- A mikroprocesszor a hozzá kapcsolódó egységekkel az ún. buszon keresztül kommunikál. A busz részei a címbusz (memória, vagy periféria kijelölésére), adatbusz (az adat továbbítására) és a vezérlőbusz (az adatforgalom típusának, irányának, időzítésének vezérlésére és egyéb funkciókra). Egy szó átvitelét megvalósító komplett adatátvitel a *buszciklus*. A lassabb perifériák (esetleg memóriák) és a CPU szinkronizálása végett a buszciklus hossza többnyire változtatható. Erre szolgál egyes processzoroknál a várakozás kérés, másoknál az adat elfogadás jel. Várakozás kérést csak akkor kell adni, ha szükséges, adat elfogadást viszont mindig.

- A mikroprocesszor működését a hozzá kapcsolódó program memóriában (többnyire ROM) tárolt *utasítások* vezérelik.

- A μP utasításokat képes beolvasni a memóriából (az utasítás regiszterbe), dekódolni (meghatározni, hogy mire utasították) és képes végrehajtani az utasítást. Az utasítás végrehajtása kisebb részekből tevődik össze, ezek a *gépi ciklus*ok (utasítás elővételi vagy fetch ciklus, memória olvasási ciklus, memória írási ciklus stb.). A *gépi ciklus*ok is kisebb részekből épülnek fel, ezek a gépi ütemek.

- A mikroprocesszor a környezettel ún. *perifériák*on keresztül tartja a kapcsolatot, képes a perifériából beolvasni, oda kiírni adatokat.

- A μP *aritmetikai, logikai és egyéb műveletek*et tud végrehajtani az adatokon, az ALU (aritmetikai logikai egység) segítségével.

- A μP az adatokat, ill. a műveletek részeredményeit el tudja tárolni a hozzákapcsolódó RAM-ba, ill. képes onnan visszaolvasni azokat.

- A RAM kijelölt részét *verem memória*ként (*stack*) tudja kezelni. Itt az utolsónak betett adathoz lehet először hozzáférni. A verem kezeléséhez az ún. *verem mutató*t (*stack pointer*) használja.

- A mikroprocesszor programjának futása megszakítható a μP interrupt bementére adott jellel. Bizonyos perifériák kezelése így kényelmesebb (pl. billentyűzet). Ilyenkor egy a perifériát kiszolgáló program rész (IT rutin) végrehajtása után folytatódik az eredeti program. (Az interrupt alatt ideiglenesen a stacken tárolódik a visszatérési cím.) Ezt nevezik *megszakítás*nak vagy *interrupt*nak. Kialakítható olyan interrupt rendszer, ahol az egyik IT rutint megszakíthatja egy másik (sürgősen kiszolgálandó periféria miatt), ezt nevezik *többszintű interrupt* rendszernek. Tilthatóság szempontjából két fajta megszakítás van, az egyik a CPU-n belül tiltható, a másik nem. Az utóbbit NMI-nek (Non Maskable Interrupt) nevezik.

- A memória és periféria közötti adatátvitel során, annak gyorsítása végett kikerülhető a mikroprocesszor, egy speciális egység, a DMA vezérlő (DMC) segítségével. Ezt *közvetlen memória hozzáférés*nek (Direct Memory Access) nevezik. Ilyenkor a CPU lekapcsolódik a buszról és a DMC vezérli az adatátvitelt. A busz vezérlési jogának átadás-átvételét nevezik *busz arbitráció*nak.

A mikroprocesszor ill. a μP-s rendszer fenti képességei alkalmassá teszik a μP-s rendszert bonyolult számítási ill. vezérlési feladatok megoldására. A továbbiakban mindezeket részletesen taglaljuk.

A egyes esetekben példaként konkrét mikroprocesszorokra is hivatkozunk, ilyenek a Z80, az I8086/286, a 8031/51, a PIC16C84. Konkrét és részletesebb információk a processzorok kézikönyvében találhatók.

4.1. Kommunikáció a mikroprocesszor és egyéb elemek között a buszon

Mint említettük, a CPU memóriával és perifériákkal az ún. buszon keresztül kommunikál. A **mikroprocesszoros busz** tulajdonképpen egy digitális információ továbbítására alkalmas kommunikációs csatorna, mely több, a funkciójukat tekintve egy halmazba sorolható jelcsoportból áll. Egy-egy ilyen jelcsoportot önmagában is szokás busznak nevezni.

A mikroprocesszoros buszra kapcsolódó egységeket két csoportra osztjuk.

- **Master**: képes a buszon lezajlódó adatátvitel vezérlésére
- **Slave**: ha a master kijelöli, képes résztvenni az adatátvitelben (mint az adat forrása, vagy mint az adatátvitel célja)

Master lehet: CPU és a DMA kontroller
Slave lehet: memóriák, perifériák

Egy teljes információ átvitel lebonyolítása a buszon két lépésből állhat:

- Az aktuális master kijelölése (ez fogja vezérelni a következő adatátvitelt). Ezt nevezik busz arbitrációnak.

- Az aktuális master kijelöli (megcímzi) a slave-et, majd megfelelő vezérlő jelek segítségével elvégzi az adatátvitel.

Az egyszerű mikroprocesszoros rendszerben egyetlen CPU és általában maximum egy-két DMA vezérlő van. A bekapcsolás után a CPU van kijelölve masterként (a CPU a default master). Ezért az első lépés általában elmarad, mivel többnyire a CPU használja a buszt.

4.1.1. A mikroprocesszoros busz részei és a részek funkciói

A mikroprocesszoros busz rendszerint három részből áll, adatbusz, címbusz és vezérlőbusz. Ez a buszfelület közvetlenül, vagy közvetve megjelenik a mikroprocesszor chipeken is. Az adat-, cím- és vezérlőbusz funkcióit és működését részletezzük az alábbiakban, főként a CPU felületén megjelenő jelekre koncentrálva.

Az következőkben egyes buszjeleket negálva jelölünk. Ezzel azt kívánjuk jelezni, hogy az adott jel a buszok többségénél alacsony aktív.

A jelnevek a különféle processzorok ill. buszok esetén eltérhetnek az itt megadottakól.

Adatbusz

Az egyszerű mikroprocesszoros rendszerben az aktuális master és a slave között egy közösen használt **adatbuszon** (Dn-D0) keresztül áramlik az információ. Az adatbusz egyik fontos jellemzője a **szószélesség**. Az első mikroprocesszoroknál ez 4 bit volt, e könyv írása idején az egyszerű vezérlési célú mikroprocesszoroknál 8 vagy 16 bit, a főként számítási célú mikroprocesszorok esetén pedig 32, 64 bit. Ez a jellemző jelentősen befolyásolja az adatátvitel hatékonyságát, hiszen szélesebb adatbuszon azonos idő alatt több adatot lehet mozgatni. Az integrálási technológia fejlődésével a processzorok adatbuszának szélessége is növekvő tendenciát mutat. Egyes processzoroknál (pl. I8086) a buszon *változó szószélességű adatátvitel* is lehetséges (byte, vagy szó átvitele), melyet külön vezérlőjel jelez.

Címbusz

A **címbusz** (Am-A0) az információ átvitelben résztvevő slave kijelölésére (megcímzésére) szolgál. A címbuszon levő bináris kódot (címet) figyelik és a saját címüket felismerik a slave-ek (a címdekóder segítségével). Minden slave-hez egy cím vagy egy címtartomány tartozik. Mivel a technológiai fejlődés egyre nagyobb méretű memóriákat tesz lehetővé, ehhez a mikroprocesszorok busza is alkalmazkodik, ezért a címbusz mérete és így a használható ROM és RAM terület (a címezhető tartomány) is egyre növekszik. Az egyszerű vezérlési célú mikroprocesszoroknál a címbusz szokásos mérete 16 bit, ami 64 kbyte megcímzését teszi lehetővé. Bonyolultabb processzoroknál Gbyte-os nagyságrendű a címezhető memória tartomány.

Vezérlőbusz

A vezérlőbusz jeleinek egy része az adatátvitel lebonyolítására szolgál. Ez egyrészt logikai jellegű funkció pl. adatáramlás iránya (olvasás/írás), típusa (pl. memória/periféria hozzáférés) ahol a jel szintje az információ hordozója, másrészt időzítés jellegű funkció, ahol a jelváltozás jellege (felfutó vagy lefutó él) és időpontja a fontos (pl. mikor stabil az adat).

Mivel az adatbusz kétirányú, ezért szükség van olyan jelre, amely megadja az **adatáramlás irányát**. Megállapodás szerint a master→slave irányt írásnak (write), a slave→master irányt pedig olvasásnak (read) nevezzük. Két módszer terjedt el az írás-olvasás vezérlésére.

Az első módszer:

$R / \overline{W}$ Az irány kijelölése történhet egyetlen jellel, melynek egyik szintje (H) olvasást, a másik (L) írást jelöl ki.

DS Adat érvényes jel (data strobe). Írás esetén ennek a jelnek a magas szintje vagy hátsó éle jelöli ki azt az időpontot, amikor a master által szolgáltatott adat stabil a buszon. Olvasás esetén a jel magas szintje jelzi, hogy a slave a buszra teheti az adatot, melyet a hátsó él környezetében mintavételez a master.

$DTACK$ Adat elfogadás jel (data acknowledge). Egyes processzoroknál a buszciklus csak akkor fejeződik be, ha a megcímzett egység (slave) kiadja ezt a jelet. Így a slave a saját igényéhez tudja igazítani a buszciklus hosszát.

A második módszer:

$\overline{RD}$, $\overline{WR}$ Két külön jellel is megadható az adatátvitel iránya. Az egyik olvasáskor ($\overline{RD}$), a másik íráskor ($\overline{WR}$) aktív. (Egyszerre sohasem lehetnek aktívak.) Ezek a jelek nem csak az irány információt hordozzák. A $\overline{WR}$ jel alacsony szintje vagy hátsó éle jelöli ki az időpontot, amikor a master által szolgáltatott adat stabil a buszon. A $\overline{RD}$ jel szintje jelzi, hogy a slave a buszra teheti az adatot, melyet a hátsó él környezetében mintavételez a master.

$\overline{WAIT}$ Várakozás kérés jel. Ha a slave a normál buszciklus alatt nem képes elkészülni az átvitellel, akkor ezen a vonalon keresztül kérheti a buszciklus meghosszabbítását.

A legtöbb mikroprocesszornál **megkülönböztetik a memóriákat a perifériáktól** (egyetlen $M / \overline{IO}$, vagy két külön jellel $\overline{MEMRQ}$, $\overline{IORQ}$). Ez azzal az előnyel jár, hogy a perifériák nem foglalnak el helyet a memória címtartományból.

$M / \overline{IO}$ Egyik szintje (H) memóriát, másik szintje (L) perifériát jelöl ki.

$\overline{MEMRQ}$, $\overline{IORQ}$

Az $\overline{MEMRQ}$ memóriát, az $\overline{IORQ}$ pedig perifériát jelöl ki. (Egyszerre sohasem aktivizálódhatnak.)

Eltérő funkciójuk miatt a mikroprocesszorok eltérő utasításokkal rendelkeznek a kétféle egység hatékony kezelésére.

Mivel a perifériára hivatkozó utasítások sok processzornál szegényesek, ezért néha programozási szempontból kellemesebb, ha a perifériára is hatásosak a memóriára hivatkozó utasítások. Másrészről esetleg egy periféria olyan sok címet lefoglal, hogy emiatt célszerű memóriaszerűen illeszteni. Ilyen okokból néha ún. **memóriába ágyazott periféria címzést** alakítanak ki. Ebben az esetben kijelölnek egy kis méretű (256 byte-1kbyte) memória tartományt a perifériák számára (többnyire a memóriatartomány legtetején). Ezt a tartományt a címből kikódolva előállítanak egy IO jelet, mely ugyanazt a szerepet játssza, mint az IORQ. A módszert később konkrét mintapéldán is bemutatjuk.

A következő jelek a megszakításokhoz kapcsolódnak.

IT (IRQ) Ez a CPU interrupt (megszakítás) kérő bemenete. Egy processzornak több IT kérő bemenete is lehet (pl.: I8085, MC8031/32). Bizonyos perifériák aktivizálják. Hatására megszakad a program normális utasítás végrehajtási sorrendje és az interrupt kérő azonosítása után a perifériát kiszolgáló program rész fut le. Utána (ha nincs újabb megszakítás) a megszakított program a megszakítást követő utasítással folytatódik. Az IT bemenetek programból globálisan letilthatók. Többnyire szintérzékeny, és *az interrupt kiszolgálásáig fenn kell tartani a kérést.*

INTA A CPU ezen a vonalon jelzi, hogy elfogadta az interrupt kérést. Vektoros ill. kód beolvasásos üzemmódban (lásd később) az adatbuszról egy vagy több byte-ot beolvas az INTA ciklusok alatt, s ez alapján határozza meg az IT rutin kezdőcímét. Van olyan CPU, amelynek nincs külön vezérlő jele erre a célra, hanem a meglévők egy speciális kombinációja jelzi (pl. Z80 esetén az M1 és IORQ egyszerre aktív).

NMI A CPU programból nem tiltható megszakítás kérő bemenete (Non Maskable Interrupt). Ez a bemenet él vagy impulzus érzékeny.

Az interrupt rendszer megvalósításától függően az interrupthoz még egyéb jelek is tartozhatnak.

A vezérlőbusz további néhány jele a **busz arbitrációt**, vagyis a busz vezérlési jogának átadását vezérli. Itt a CPU felületen megjelenő jeleket ismertetjük.

BUSRQ A buszkérés vagy más néven *HOLD* (tartás) vonal jelzi a CPU-nak, hogy a busz vezérlési jogát át kell adnia egy másik egységnek. Ekkor a CPU, amint az aktuális gépi ciklust (lásd később) befejezte, lekapcsolódik a buszról. Ez azt jelenti, hogy az összes buszmegható jeleit magas impedanciás állapotba (harmadik állapotba) teszi.

BUSACK (HLDA): Itt jelzi vissza a CPU a buszt elkérő egységnek, hogy lekapcsolódott a buszról, így azt egy másik master használhatja.

Bonyolultabb esetben az arbitrációs rendszertől függően az arbitrációhoz egyéb jelek is szükségesek, erről később lesz szó.

A vezérlőbusz része még néhány idáig fel nem sorolt jel is (CLOCK, RESET). Ezek szintén megjelennek a CPU felületen.

CLOCK A processzor órajele. Többnyire külső órajel generátor állítja elő, de vannak processzorok, amelyekhez csak kvarcot (az oszcillátor frekvenciáját meghatározó alkatrész) kell csatlakoztatni egy belső oszcillátorhoz. A CPU belső órajele sokszor a külső ill. belső oszcillátor frekvenciájának leosztásával, néha felszorzásával áll elő. (Az adott típusú processzornál alkalmazható maximális órajelfrekvenciát nem szabad túllépni!) A belső órajel adja a mikroprocesszor működésének alapütemét. Egyes CPU-knak külön órajel kimenete is van. Az órajelet a slave-ek is használhatják, ezért a vezérlőbusz része.

RESET A CPU-t és az összes többi egységet alaphelyzetbe hozó jel. Meg szokás adni a minimális hosszát, mégpedig többnyire a CPU órajel periódusát véve alapegységnek. A RESET jel hosszának be nem tartása esetén a processzor illegális állapotba kerülhet, ami hibás működéshez vezet. A reset áramkör a tápfeszültség megjelenésekor és esetleg egy külső nyomógomb hatására adja a RESET jelet.

Egyes processzorok egyéb speciális jelekkel is rendelkeznek (státus információk, többprocesszoros rendszer arbitrációjához szükséges jelek, koprocesszor illesztéséhez szükséges jelek), amelyek közül azonban csak egyesek részei a vezérlőbusznak (pl. a Z80 külön jelzi az utasítás elővételi ciklust az M1 jellel, amely - mint már említettük - az INTA jelzésében is részt vesz, ezért a vezérlőbusz része).

4.1.2. A kommunikáció időbeli lefolyása

A buszon lezajló, egy szó átvitelét megvalósító komplett adatátvitelt **buszciklus**nak nevezzük.

Egy buszciklus során az aktuális master a címbuszra adott megfelelő kombinációval megcímzi a slave-et. A slave felismeri (dekódolja) a saját címét.

Olvasás esetén a master a vezérlőbuszon keresztül engedélyezi, hogy a megcímzett slave rátegye az adatbuszra az adatot, melyet annak stabilizálódása után a master mintavételez.

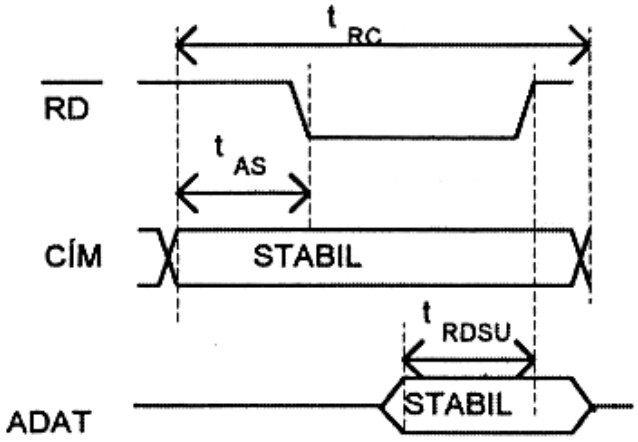
Írás estén a címzés után, - ha a master egy CPU -, a CPU ráteszi a kiírandó adatot az adatbuszra, majd a vezérlőbuszon keresztül egy beíró impulzust generál, melyet a slave felhasznál az adat belső regiszterébe mintavételezéséhez.

Mivel az adatbuszt minden egység használja, ezért arra többnyire három állapotú meghajtóval csatlakoznak az egyes egységek, de van olyan rendszer is ahol nyitott kollektorosan (open collectorosan).

A különféle CPU-k, az eltérő vezérlőbusz kialakítások miatt, eltérő módon végzik a kommunikációt. Az alábbiakban először eltekintünk attól, hogy a kommunikáció milyen típusú egységgel (memória vagy periféria) folyik, csak az olvasás és írás időbeli lefolyására koncentrálunk.

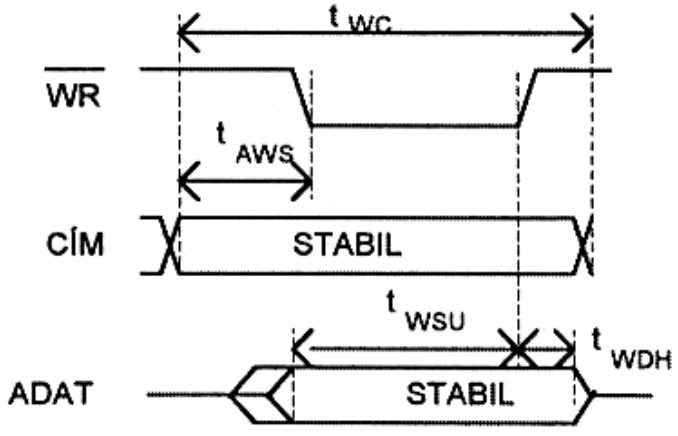
Olvasási és írási ciklus lefolyása a buszon két irányvezérlő jeles kialakítás esetén

Az olvasási ciklus lefolyását mutatja a 4.3. ábra. A busz master először a címet teszi ki a címbuszra. Ezután t_{AS} idő múlva aktivizálja a $\overline{RD}$ olvasás engedélyező jelet. Ha a címét felismeri, akkor a $\overline{RD}$ jel hatására teszi a buszra az adatot az olvasott egység, a rá jellemző késéssel. Az adatnak stabilizálódni kell, mielőtt az olvasó egységbe mintavételeződik (t_{RDSU}). Az olvasás engedélyező jel megszűnése után az olvasott egység némi késéssel lekapcsolódik az adatbuszról. Az olvasások minimálisan t_{RC} időnként követhetik egymást (az olvasás ciklusideje).



4.3. ábra. Olvasási ciklus két irányvezérlő jeles esetben

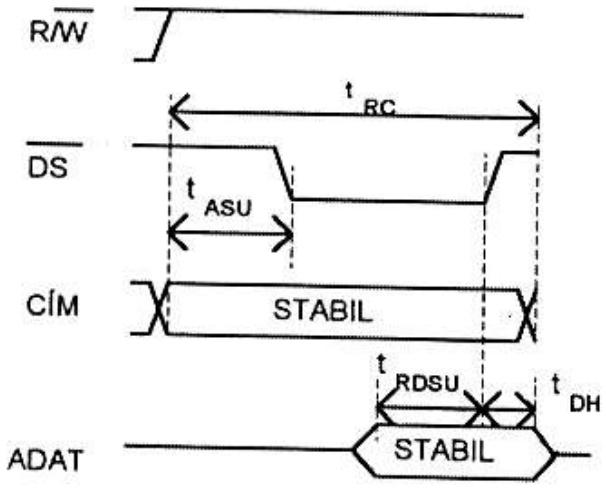
Az írási ciklus lefolyását mutatja a 4.4. ábra. A busz master először a címet és az adatot teszi ki a buszra. Ezután aktivizálja a $\overline{WR}$ beíró jelet. A beíró jel hátsó éle előtt t_{WSU} idővel az adat stabilizálódik és még utána is stabil t_{WDH}.ideig. Processzortól függően a $\overline{WR}$ jel aktív szinje vagy a hátsó éle jelöli ki az időpontot, amikor az írt egységbe az adatot mintavételezni lehet. Az írások minimálisan t_{WC} időnként követhetik egymást (az írás ciklusideje).



4.4. ábra. Írási ciklus két irányvezérlő jeles esetben

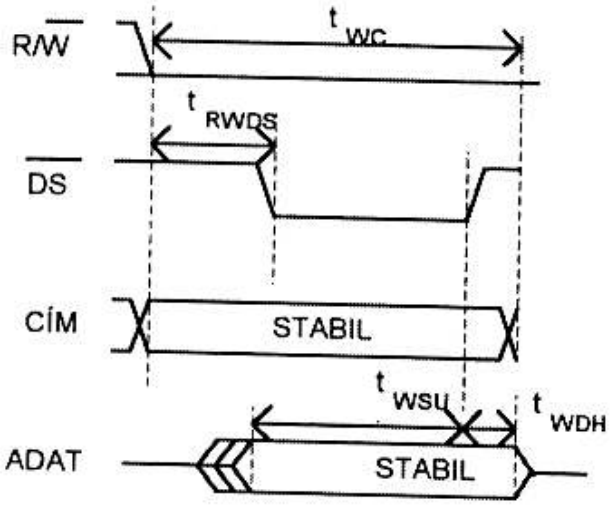
Olvasási és írási ciklus lefolyása a buszon egy irányvezérlő jeles kialakítás esetén

Egy irányvezérlő jeles esetben a az olvasás jelnek az $RD = R/\overline{W}\ DS$, az írás jelnek pedig $WR = \overline{R/\overline{W}}\ DS$ feleltethető meg. Az olvasási ciklus lefolyását mutatja két irányvezérlő jeles esetben a 4.5. ábra. A busz master először a címet teszi ki a címbuszra, s ezzel egyidőben az $R/\overline{W}$ jellel kijelöli az olvasási irányt. Ezután t_{AS} idő múlva aktivizálja a $\overline{DS}$ (data strobe) jelet. Ezek hatására teszi a buszra az adatot az olvasott egység, a rá jellemző késéssel. Az adatnak stabilizálódni kell, mielőtt az olvasó egységbe mintavételeződik az adatbusz tartalma (t_{RDSU}). Az olvasás engedélyező jel megszűnése után az olvasott egység némi késéssel lekapcsolódik az adatbuszról.



4.5. ábra. Olvasási ciklus egy irányvezérlő jeles esetben

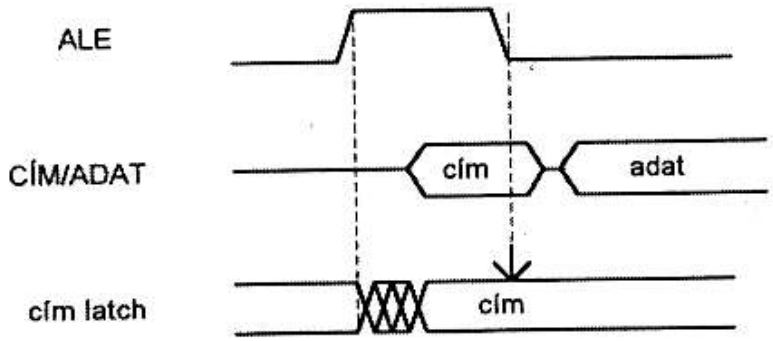
Az írási ciklus lefolyását mutatja egy irányvezérlő jeles esetben a 4.6. ábra. A busz master először a címet és az adatot teszi ki a cím- és adatbuszra, továbbá beállítja az írási irányt. Ezután aktivizálja a $\overline{DS}$ (data strobe) jelet. Ennek hátsó éle előtt t_{WSU} idővel az adat stabilizálódik és még utána is stabil t_{WDH} ideig. Processzortól függően a DS $\overline{R/\overline{W}}$ aktív szintje vagy hátsó éle jelöli ki az időpontot amikor az írt egység az adatot mintavételezheti.



4.6. ábra. Írási ciklus egy irányvezérlő jeles esetben

Időmultiplexált cím/adat busz

Az adat és címbusz a CPU felületén részben ugyanazokon a lábakon is megjelenhet (láb spórolás miatt), időben egymás után (időmultiplexálva, 4.7. ábra.), először a cím, majd az adat. Ekkor a cím stabil megjelenésének időpontját egy külön ALE (address latch enable) jel jelzi. Ezzel lehet a címet egy latchben eltárolni. A cím/adat az ALE magas szintje alatt már látszik a latch kimenetén (a latch átlátszó). Az ALE megszűnése hatására a cím eltárolódik. Ha D flip-flopot használnánk a cím tárolására, akkor csak az ALE hátsó éle után lenne stabil a cím, így annak dekódolására kevesebb idő jutna a slave-ben.



4.7. ábra. Időmultiplexált cím/adat busz

A memória és periféria olvasás engedélyező és beíró jelek kialakítása

A memóriáknál ill. a perifériáknál elő kell állítani következő információkat hordozó jeleket:

$\overline{MEMRD}$ memória olvasás engedélyezése

$\overline{MEMWR}$ memória írás vezérlése

$\overline{IORD}$ periféria olvasás engedélyezése

$\overline{IOWR}$ periféria írás vezérlése

Ezt mutatja egy és két vezérlőjeles memória-periféria megkülönböztetés esetén a 4.1. táblázat.

Egy vezérlőjeles memória-periféria megkülönböztetés	Két vezérlőjeles memória-periféria megkülönböztetés
$MEMRD = M / \overline{IO} \cdot RD$	$MEMRD = MEMRQ \cdot RD$
$MEMWR = M / \overline{IO} \cdot WR$	$MEMWR = MEMRQ \cdot WR$
$IORD = \overline{M / \overline{IO}} \cdot RD$	$IORD = IORQ \cdot RD$
$IOWR = \overline{M / \overline{IO}} \cdot WR$	$IOWR = IORQ \cdot WR$

4.1. táblázat. Memória és periféria vezérlőjeleinek előállítása

A táblázatban szereplő logikai kifejezések ponáltan adják meg az egyes jelek kialakítását. Pl. a MEMRD akkor aktív, ha MEMRQ aktív és RD aktív. A bal oldalon álló jelek, általában alacsony aktívak (az L szinthez van rendelve az igaz érték), amit negálás jellel jelölünk a táblázat előtt szereplő felsorolásban.

Egy irányvezérlő jeles írás-olvasás megkülönböztetés esetén - mint már említettük - az olvasás jelnek az $RD = R/\overline{W}\ DS$, az írás jelnek pedig $WR = \overline{R/\overline{W}}\ DS$ feleltethető meg, ezért ebben az esetben a fenti jelek előállítása ezek behelyettesítésével módosul.

4.1.3. Az aszinkron és a szinkron busz

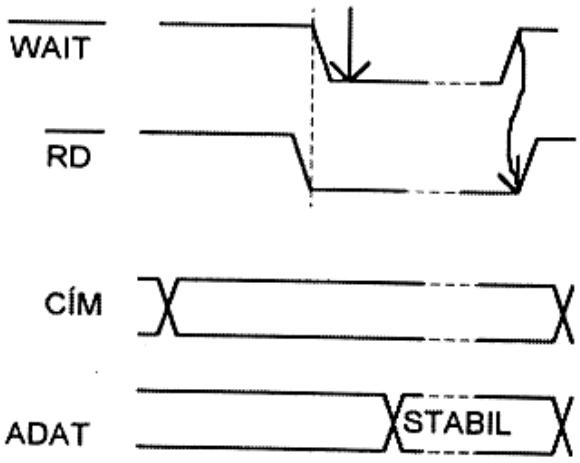
Szinkron busz és vezérlő jelei

Szinkron busz esetén a buszciklus a slave-től függetlenül is mindenképpen fix idő alatt lezajlik.

A lassú slave-ekhez való szinkronizálódás miatt többnyire lehetőség van a buszciklus meghosszabbítására. Azok az egységek hajtják meg a ciklus hosszabbítást előidéző $\overline{WAIT}$ vonalat, amelyek nem elég gyorsak az átvitelhez. A meghajtás nyitott kollektoros. Ha egy egységnek ciklushosszabbításra van szüksége, akkor a $\overline{WAIT}$ jelet a ciklus kezdetén, egy a busz mastertől függő időpontig (amikor a master mintavételezi a $\overline{WAIT}$ vonalat) feltétlenül meg kell hajtania, különben a busz master nem veszi figyelembe (4.8. ábra). Ha a busz master észrevette, hogy waitet kérnek tőle, akkor mindaddig nem fejezi be az aktuális buszciklust, ameddig a wait kérés meg nem szűnt.

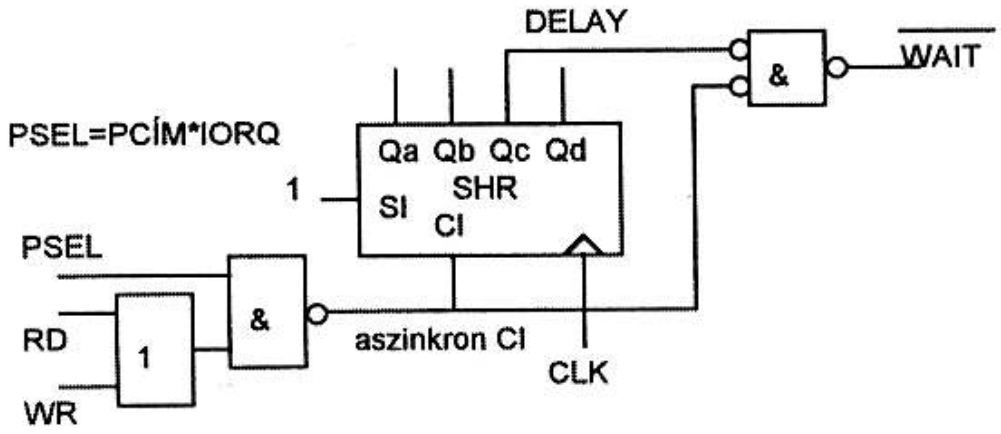
Egy hibás waitet igénylő periféria "lefagyaszthatja" a buszt, ami ellen a $\overline{WAIT}$ vonal és a CPU közé iktatott ún. watch-doggal lehet védekezni. Ez az áramkör megszünteti a wait kérést, ha az egy előre meghatározott időnél hosszabb ideig fennáll, és esetleg jelzi a hibát egy NMI kéréssel.

A módszer előnye, hogy mivel az egységek többségének a normál ciklushossz is elegendő, a wait kérő áramkörre csak ritkán van szükség.



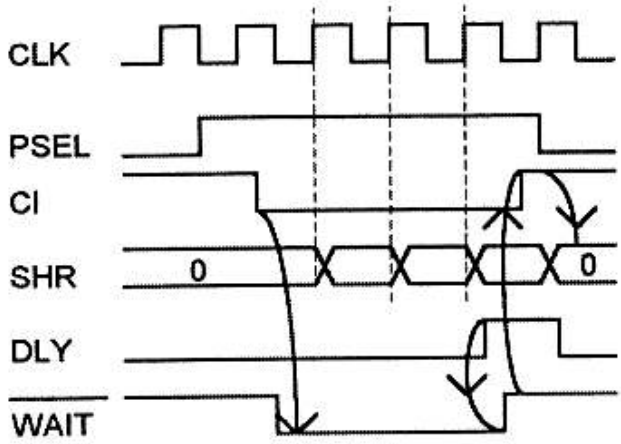
4.8. ábra. Szinkron busz idődiagramja

A WAIT jel előállítását mutatja az 4.9. ábra.



4.9. ábra. WAIT jel előállítása

Amíg a periféria nincs kiválasztva, addig a shiftregiszter törölve van. Amint a perifériához fordul a master, három kapu késleltetésnyi idő múlva aktivizálódik a WAIT mert még nincs DEALY jel. Ha a DELAY jel megjön, az letiltja a WAIT generálást, aminek hatására megszűnik az RD ill. WR és újra törlődik a shiftregiszter. Az elmondottak a 4.10. ábrán követhetők végig. Konkrét esetben figyelembe kell venni, hogy mennyi idő áll rendelkezésre a RD ill. a WR megjelenése után, a WAIT előállítására illetve, hogy a master számára hol szükséges a WAIT jel, hogy azt figyelembe vegye, s ennek megfelelő sebességű áramköröket kell alkalmazni.



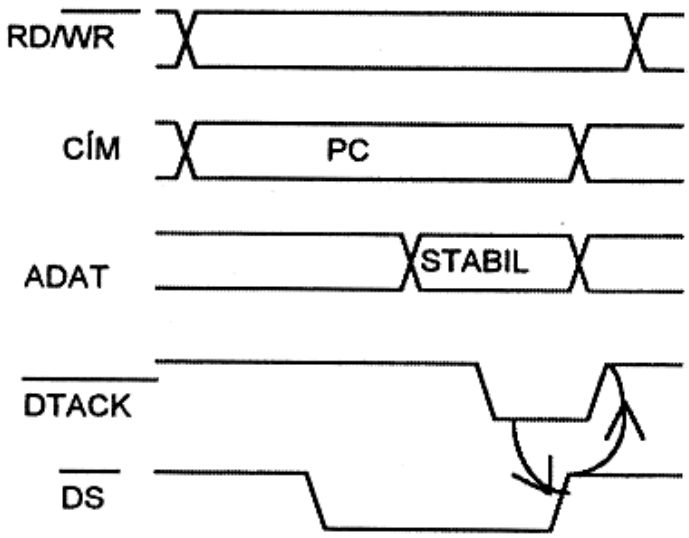
4.10. ábra. WAIT előállító áramkör idődiagramja

A RD+WR jel bizonyos CPU-k esetében elhagyható mivel a PSEL-ben szereplő IORQ jel pótolja, ha biztosított, hogy az IORQ csak periféria írás ill. olvasás esetén aktív. (Pl. a Z80 esetén ez nem biztosított, mert M1*IORQ az INTA ciklust jelzi, s ez alatt a címbuszon akár a periféria címe is megjelenhet.)

Aszinkron busz és vezérlőjelei

Aszinkron busz esetén egy adatátvitel csak akkor fog befejeződni, ha a slave visszajelezte, hogy az adatátvitel kész. Tehát ilyenkor az összes slave az átvitel során

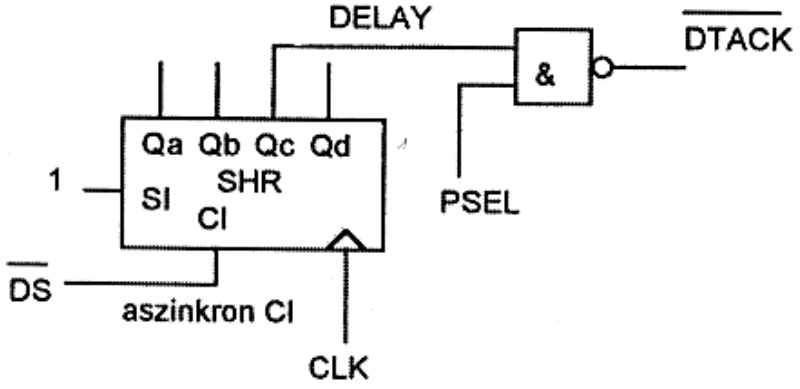
(akár írás, akár olvasás) köteles meghajtani az adatátvitel végét jelző DTACK (Data Acknowledge) vonalat (nyitott kollektorosan). A folyamat lezajlását mutatja a 4.11. ábra.



4.11. ábra. Aszinkron busz idődiagramja

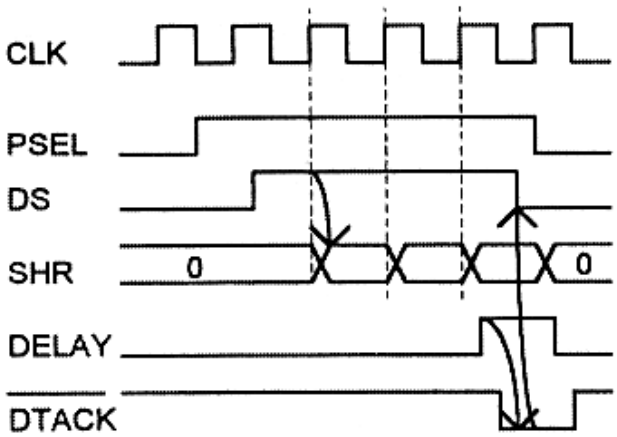
Amikor a slave elkészült (írás esetén beíródott az adat, olvasás esetén a buszra tette az adatot), a DTACK-al jelzi a busz masternak. A busz master ennek hatására visszaveszi a DS (Data Strobe) jelet. A slave a DS megszűnésének hatására veszi vissza a DTACK jelet. Az ilyen típusú adatátvitelt nevezik **hand-shake**-es (kézfogásos) adatátvitelnek.

A 4.12. ábra egy DTACK előállító áramkört mutat be.



4.12. ábra. DTACK jel előállítása

A shiftregiszter a törölve van, amíg nem aktivizálódik a DS jel, így nincs DTACK sem. Ha a DS aktív, a shiftregiszterbe balról 1 kezd beshiftelődni. Amint ez elérte a DELAY kimenetet, a DTACK aktivizálódik (ha a periféria címe van a buszon, amit a PSEL jelez). Ezt észreveszi a master és megszünteti a DS jelet. Ekkor törlődik a shiftregiszter s ezért megszűnik a DTACK jel is. A buszciklus órajelben mért hossza attól függ, hogy a shiftregiszter mely kimenetéről vesszük le a DELAY jelet. Az áramkör működését szemlélteti az 4.13. ábra idődiagramja.



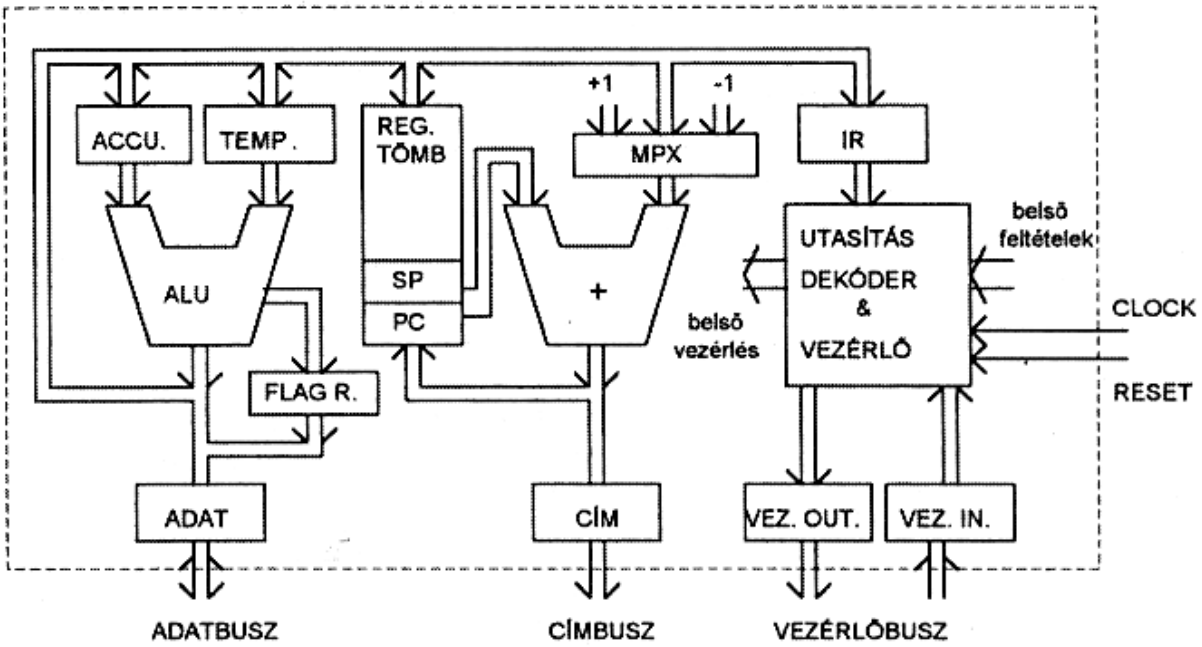
4.13. ábra. DTACK előállító áramkör idődiagramja

Az aszinkron busz előnye, hogy a busz master rugalmasan képes alkalmazkodni a slave sebességéhez, egy gyors egységet rövid idő alatt képes kiszolgálni, de a lassú egységet is képes megvárni.

Hátránya, hogy egy meghibásodott egység a rendszer "lefagyását" okozhatja (ha nem adja vissza DTACK jelet). Ez ellen - a wait-es esethez hasonlóan - egy időzítő watch-dog áramkörrel lehet védekezni. Ez az áramkör minden átvitel során elindít egy időzítést, s amennyiben a DTACK jel nem jön meg egy időkorláton belül, automatikusan generálja azt, és (pl. az NMI aktivizálásával) biztosítja, hogy a CPU értesüljön a hibáról.

4.2. A mikroprocesszor belső felépítése

A mikroprocesszor belső felépítésének tanulmányozásához újra elővesszük a blokkvázlatot, mely egy egyszerű mikroprocesszor belsejét mutatja (4.14. ábra).



4.14. ábra. Egyszerű CPU belső blokkvázlata

A CPU architektúrájának legfontosabb elemei a következők:

Utasítás regiszter (IR)

Ebbe a regiszterbe olvassa be a CPU a külső programot tartalmazó memóriából a végrehajtandó utasítás kódját.

Utasítás dekóder

Az utasítás regiszterben levő utasítás értelmezését végzi.

Vezérlő és időzítő egység

Ez az egység vezérli a CPU belső adatstruktúráját (az adatutakat, a belső regiszterek beírását, ALU-t stb.) a felismert utasításnak megfelelően, ill. reagál a külső jelekre. A vezérlő CPU-tól függően huzalozott logikás, vagy mikroprogramozott. Létezik olyan CPU, amelynek utasításkészletét a felhasználó rendelheti meg. (Ilyenkor a mikroprogramot a felhasználó igényeinek megfelelően írják meg.)

Aritmetikai logikai egység (ALU)

Feladata az aritmetikai (összeadás, kivonás stb.), logikai (bitenként ÉS, VAGY stb.) és néhány egyéb (shiftelés, rotálás stb.) művelet elvégzése a bemeneteire kerülő operandusokon. Az ALU a belső buszon keresztül a CPU belső regisztereivel és az adatbusszal van kapcsolatban. Egy művelet operandusai ill. az eredmény helye a belső regiszterek valamelyike (ez sok processzornál nem lehet tetszőleges), vagy a külső RAM lehet. Azt a belső regisztert, amelybe az eredmény kerül, akkumlátornak nevezik. Egyes CPU-k több akkumlátorral is rendelkeznek.

Regiszterek

A CPU belső regisztereit funkciójuk alapján két csoportra osztják, általános célú regiszterekre és speciális funkciójú regiszterekre.
Az **általános célú regisztereket** belső adattárolásra használják. Ezek sokkal gyorsabban hozzáférhetők, mint a külső memória. Az általános célú regiszterek mérete többnyire a mikroprocesszor szószélességével vagy annak néhányszorosával egyezik meg.

A **speciális funkciójú regiszterek** a következők:

PC (Program Counter) program számláló
Ez a regiszter tartalmazza a végrehajtandó utasítás címét. Normál utasítás végrehajtási sorrend esetén minden utasítás végrehajtása után inkrementálódik (1-el nő). A normál szekvenciát megváltoztató (ugró, szubrutin hívó stb.) utasítások esetén a megfelelő címmel feltöltődik. A PC mérete többnyire a CPU címbuszának szélességével megegyező.

IR utasítás regiszter
Az utasítás elővételi (fetch) ciklusban beolvasott utasítást tárolja. Ennek tartalmát dolgozza fel az utasításdekódér.

SP verem mutató (stack pointer)
Ez a többnyire külső RAM memóriában kialakítható verem memória (szinonímái: LIFO, Last in First Out, stack, zsák memória) jellegű memóriakezelés megvalósításához szükséges mutató értékét tárolja. Ez az ún. szubrutinok (egy utasítás segítségével aí program tetszőleges helyéről hívható program részek) megvalósításához és az interrupt működéséhez szükséges. Az SP mérete többnyire megegyezik a PC méretével.

Mikrokontrollereknél a belül kialakított kis méretű stack miatt sokszor csak 8 vagy kevesebb bites. Egyes mikrokontrollerekben (pl. PIC16C84) teljesen különálló, véletlen címzéssel nem elérhető hardver stacket alakítottak ki.

Akkumlátor

A műveletek egyik operandusa és eredménye többnyire ebben a regiszterben helyezkedik el. Mérete a mikroprocesszor adatbuszának szószélességével egyezik meg.

Jelzőbit (flag) regiszter

Többnyire a műveletek eredményétől függő jelzőbitek (flagek) és egyéb, a CPU állapotára vonatkozó információk helyezkednek el ebben a regiszterben. A műveletek eredményeitől függő flageket az ALU állítja elő (összeadás és kivonás esetén az átvitel bit, az eredmény előjele stb.).

A flagek értéke alapján feltételes elágazásokat és egyéb összetett műveleteket végez a processzor, ill. aritmetikai műveleteknél használja fel azokat.

A műveletek által állított flagek processzor függők, de a következőkben felsoroltak szinte minden CPU-nál megvannak:

Signum: Az eredmény előjele pozitív vagy negatív.

Zero: Az eredmény 0.

Carry: Aritmetikai *átvitel* és rotáló/shiftelő műveleteknél ez a bit is bekapcsolódhat az operandus adatbitjei közé.

Auxiliary carry vagy **Half carry:**

A 3. és 4. bit közötti átvitel (BCD műveletek esetén)

Parity: A művelet eredményében az 1-esek száma páros.

N: Az utolsó művelet kivonás volt.

Az alábbi regiszter funkciókat nem implementálják minden CPU-ban. (A hivatkozott címzési módokat később részletezzük):

Bázis regiszterek

Az adatok bázisrelatív címzésénél az adatterület bázis címét tartalmazza.

Index regiszterek

Az adatok indexelt címzésű hozzáférését teszik lehetővé.

Szegmens regiszterek

Szegmens szervezésű memória kezelést alkalmazó CPU-knál (pl. I8086) a memória szegmens kezdőcímét tartalmazzák.

4.3. A mikroprocesszor működése (utasítás ciklus, gépi ciklus, ütem)

A processzor feladata, hogy végrehajtsa a program memóriában található programot. A program utasítások sorozatából áll. Az utasítások végrehajtását kisebb egységekre, gépi ciklusokra ill. annak elemeire, az ütemekre lehet bontani. Magának az utasításnak a végrehajtását utasítás ciklusnak nevezik.

A CPU működéséhez órajelre van szükség, hiszen az egy szinkron sorrendi hálózat. A processzorok sebesség növelését a gyártók részben az áramkörök sebességének növelésével tudják elérni, ami nagyobb megengedett órajel frekvenciát eredményez.

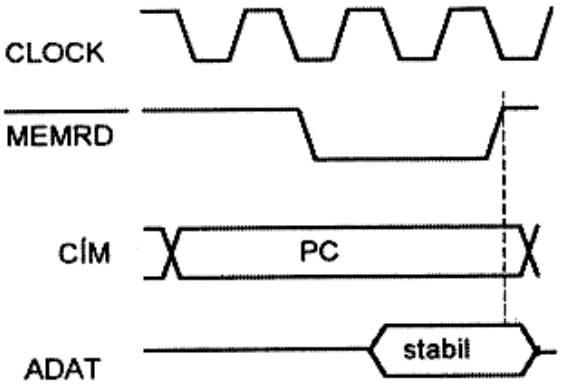
A processzor belső áramköreinek órajele adja a működés alapegységét a **gépi ütemet**. Ez a legkisebb időtartam, amely alatt a CPU-ban valamely elemi működés végrehajtódik, ez tulajdonképpen a CPU belső vezérlőjének órajel periódus ideje.
A processzor egy utasításának végrehajtása ún. **gépi ciklus**okból áll, melyek több ütemből tevődhetnek össze. Az utasítás végrehatása során a gépi ciklusok egy része belső processzor működést végez (pl. az ALU az adaton elvégzi a kijelölt műveletet). Másik része adatátvitelt hajt végre a buszon. Az utóbbiakat **buszciklus**oknak nevezzük.

4.3.1. A mikroprocesszor buszciklusai

A következőkben a különféle buszciklusokat ismertetjük. (Az idődiagramok egy fiktív CPU-ra vonatkoznak, mely szinkron busszal rendelkezik. Az idődiagramokon a CPU órajelét is feltüntettük, utalva arra, hogy egy gépi ciklus gépi ütemekből áll és az órajellel szorosan szinkronban van.)

FETCH ciklus (utasítás elővételi ciklus)

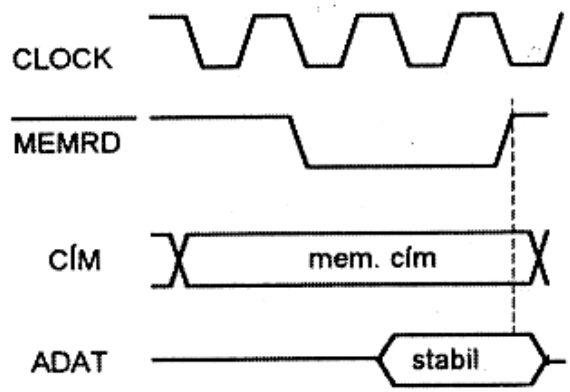
Ez a ciklus (4.15. ábra) szolgál a memóriában található utasítás kódjának az utasítás regiszterbe (IR) való beolvasására. Az utasítás kódjának címét a PC tartalmazza. Egy utasításkód beolvasása annak hosszától (hány szóból áll) függő számú FETCH ciklust igényel. Az utasítás végrehajtása (execute) az utasítás dekódolása alapján történik. Az utasítás olvasási ciklus a processzorok egy részénél időzítésben némileg különbözik az ún. memória olvasási ciklustól és azt egy külön vonal jelzi (pl. a Z80 M1 jele).



4.15. ábra Fetch ciklus

Memória olvasási ciklus

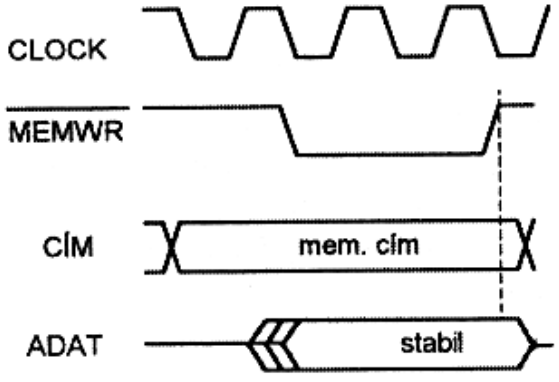
A memória megcímzett regiszterében található adatnak a processzor valamely regiszterébe való beolvasása (4.16.ábra).



4.16. ábra Memória olvasási ciklus

Memória írási ciklus

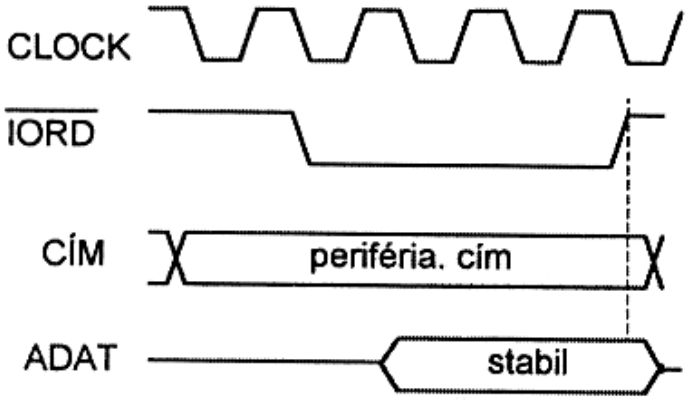
A processzor valamely regiszterében található adat valamely memória címre írása. Az adat beírása a beíró jel alacsony szintjére vagy hátsó élénél történik (4.17. ábra). (Ekkor stabil az adat.)



4.17. ábra. Memória írási ciklus

Periféria olvasási ciklus

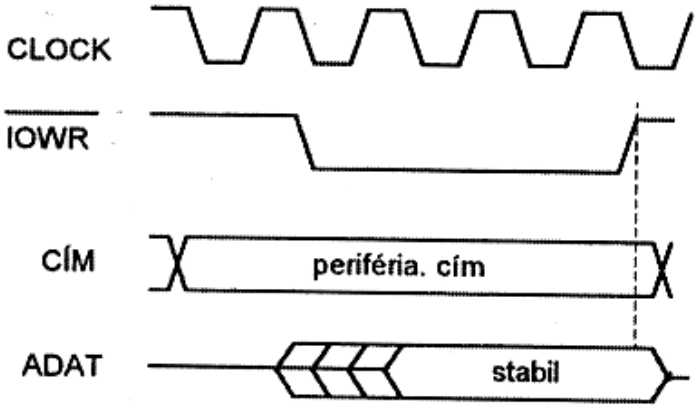
Egy periféria címen található adatnak a CPU valamely regiszterébe olvasása (4.18. ábra).



4.18. ábra. Periféria olvasási ciklus

Periféria írási ciklus

A processzor valamely regiszterében található adatnak az adott periféria címre írása (4.19. ábra).



4.19. ábra. Periféria írási ciklus

Az utóbbi két gépi ciklust I/O ciklusoknak is nevezik. Az utasítás végrehajtás gépi ciklusain kívül speciális gépi ciklusok is léteznek, ilyen az interrupt elfogadási (INTA) ciklus, és a busz vezérlés-átadási (arbitrációs) ciklus, de ezekről később lesz szó.

4.3.2. Egy utasítás végrehajtása

Egy komplett utasítás végrehajtási (execute) ciklus az előbb ismertetett gépi ciklusokból áll össze. Konkrét mintapéldaként megmutatjuk, hogy miképp történik a Z80 CPU, LD DE,(A150H) utasításának a végrehajtása. Az utasításra a program írásakor annak mnemonikjával (nevével) hivatkozunk. Ezeket egy fordító program lefordítja a processzor által értelmezhető bináris számokra, melyet be kell programozni a ROM-ba

(vagy háttér tárolóról betölteni a RAM-ba), melyből a CPU előveszi, értelmezi és végrehajtja azokat.

A fenti utasítás az A150 hexadecimálisan megadott címtől kezdve beolvas két byte-ot, az elsőt a Z80 CPU E regiszterébe, a másodikat pedig a D regiszterébe teszi.

Tegyük fel, hogy az utasítás kódja a hexadecimális 8000-es címen kezdődik. Az alábbiakban látható, hogy az utasítás kód byte-jai hogyan helyezkedik el a memóriában, és hogy azokhoz milyen gépi ciklusok kapcsolódnak.

cím	adat		gépi ciklus típusa
8000	ED	az utasítás kód 1. byte-ja,	1. FETCH ciklus
8001	5B	az utasítás kód 2. byte-ja	2. FETCH ciklus
8002	50	a cím alsó fele	3. memória olvasási ciklus
8003	A1	a cím felső fele	4. memória olvasási ciklus,
			5. memória olvasási ciklus, E=(A150H),
			6. memória olvasási ciklus D=(A151H).
8004	A következő utasítás 1. byte-ja.		

Amikor a processzor ehhez az utasításhoz ér, a PC-ben 8000H van. Erről a címről beolvassa az utasítás kód első byte-ját az utasítás regiszterbe, ez FETCH ciklus, melynek a végén a PC-t inkrementálja (PC=PC+1). Az utasítást dekódolva megállapítja, hogy az utasításnak van még egy byte-ja, s ezt is beolvassa a következő FETCH ciklus alatt (PC=PC+1). A teljes utasítást dekódolva megállapítja, hogy be kell olvasnia még két byte-ot, melyeket majd memória címként kell értelmezni. Ez két memória olvasási ciklus, melynek végén a PC a következő utasítás címére mutat. Ezután az A150H címről beolvassa az adatot az E regiszterbe, majd az A151H címről a D regiszterbe, ez két memória olvasási ciklus. (A memória címen levő adatot zárójelbe tett memória címmel jelöltük.) Ezután következhet a következő utasítás elővétele.

A processzor a vezérlés átadó utasítások és program megszakítás kivételével mindig szekvenciálisan (és növekvő memória címek felé) hajtja végre az utasításokat, vagyis az utasítás végén a PC a következő utasítás címére mutat.

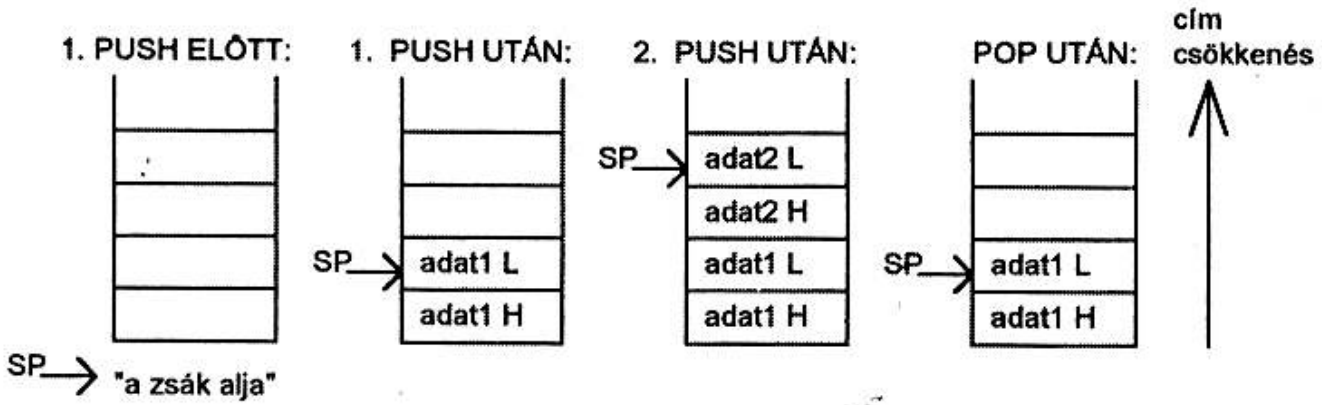
A verem tár (stack) kezelése

A stack (LIFO, verem tár, zsák memória) a RAM memóriának egy speciálisan kezelt része, melynek a program végrehajtása során fontos szerepe van. A kezelésének lényege, hogy az adatokhoz a beírással ellentétes sorrendben lehet hozzáférni, vagyis mindig a legutoljára beírt adatot lehet legelőször kiolvasni. A beírás ill. kiolvasás címét a stack pointer tartja nyilván.

Két olyan utasítás van, amely csak a stackbe való írást (PUSH forrás) ill. az onnan történő olvasást (POP cél) szolgálja. Azonban egyéb utasítások is kezelik a stacket, ilyenek a szubrutin hívó ill. abból visszatérő utasítások, melyeket később részletezünk.

A stack pointer értéke a legtöbb CPU-nál egy-egy adat betétele során csökken (a tartomány a csökkenő memóriacímek felé hízik), de van ellentétes példa is. A POP és PUSH utasítások által egyszerre betett ill. kivett adat byte-ok száma CPU és utasítás

függő (pl. Z80 esetén mindig 2 byte). A 4.20. ábra mutatja, hogy mi történik, ha az eredetileg üres stackbe két egymást követő PUSH utasítással 2 byte-os adatokat teszünk, majd egy POP utasítással kiveszünk adatot.



4.20. ábra. A stack kezelése

4.4. Utasításrendszer

A mikroprocesszorok működését az utasítások vezérlik. Az utasítások processzor specifikusak, minden processzorhoz tarozik egy utasítás halmaz, amelyet az végrehajtani képes. Ezt az utasítás halmazt nevezik a processzor *utasításkészlet*ének. Az utasításkészlet definíciójában leírják az utasítás kódját (egy utasítás kód több szóból állhat), az utasítás hatását (mit csinál), az utasítás mnemonikját (nevét). Az utasítás a műveleti kódból (op. code, ez kódolja, hogy mit kell csinálni) és az operandus elhelyezkedésére vonatkozó adatokból áll.

Az utasításkészletben szereplő utasításokat az elvégzett művelet típusa alapján szokás csoportosítani, az egyes processzorok programozási kézi könyvében is ily módon szerepelnek. Egy szokásos csoportosítás: adat mozgató, adat kezelő, vezérlés átadó, CPU vezérlő. Itt is ezt a csoportosítást követjük.

Az µP-k gyártói többnyire ún. mikroprocesszor családokat hoztak létre. A család újabb elemeinél (amíg az a fejlesztést nem gátolja túlzottan) arra törekednek, hogy az újabb típusok szoftver (esetleg hardver) kompatibilisek legyenek a régebbiekkel. A szoftver kompatibilitás azt jelenti, hogy a régebbi CPU-k utasítás kódját felismeri és hibátlanul végrehajtja az újabb processzor. (Természetesen többnyire újabb utasításokkal is bővül az utasítás készlet.) Ilyenkor a régi készülékek szoftvere változtatás nélkül (vagy minimális változtatással) futtatható az újabb processzorokon, amivel sok fejlesztési munkát lehet megspórolni.

4.4.1 Az utasítás csoportok

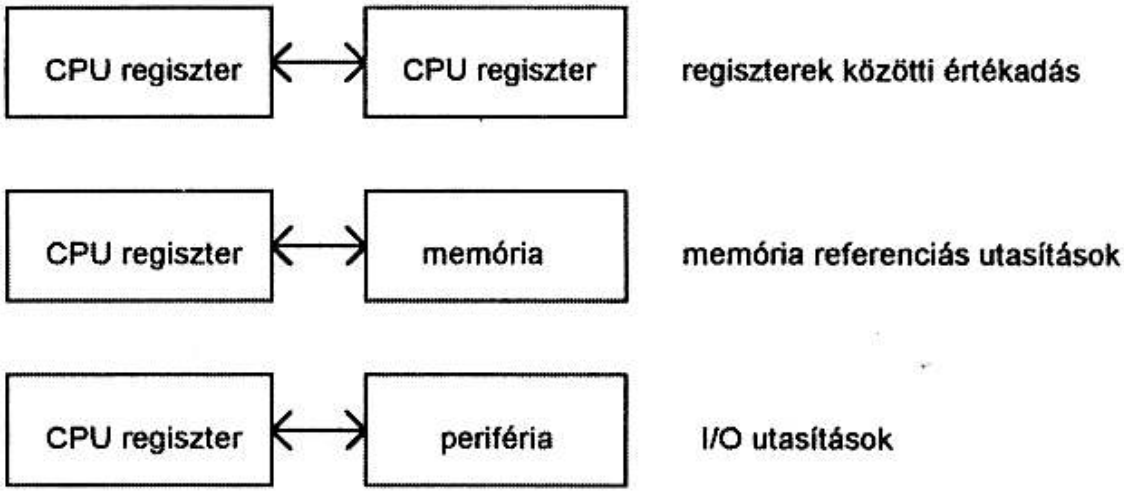
A különféle mikroprocesszorok utasításkészlete többé-kevésbé eltérő, de a konkrét utasítások többnyire besorolhatók a következőkben megadott kategóriákba. Az

utasításokra azok ún. mnemonikjával (nevével) szokás hivatkozni assembly nyelven. Ez az a string, amelyet feldolgozva egy assembler (az assembly nyelv fordítója) előállítja az utasítás kódját (azt a bináris adatot, amelyet a processzor értelmez).

Az alább felsorolásra kerülő utasítások közül vannak olyanok amelyeket egyik másik processzor nem ismer, ill. a mnemonikjuk nem egészen az itt megadottnak megfelelő! Az óriási választék miatt nem soroljuk fel az utasítások változatait, és olyanok is vannak, amelyeket meg sem említünk. A konkrét utasításokat és azok mnemonikját minden esetben megtalálhatjuk az adott CPU kézikönyvében.

Adatmozgató utasítások

Az adatmozgató utasítások adatokat visznek át (4.21. ábra.):
- a CPU regiszterei között
- a CPU valamely regisztere és a memória között (memória referenciás utasítások)
- a CPU valamely regisztere és a periféria között (I/O utasítások)



4.21. ábra. Adatmozgató utasítások forrás-cél szerinti csoportosítása

Az adatmozgatás vonatkozhat egy adatra, vagy egy adat blokkra, az utóbbiakat *blokkmozgató, tömb kezelő ill. string kezelő utasításoknak* nevezzük.

Az egyszerre átmozgatott adat szószélessége lehet byte, szó (2 byte), hosszú szó (4 byte) stb. processzortól és utasítástól függően.

Sok processzor rendelkezik olyan utasítással, amely két regiszter tartalmát felcseréli (exchange).

Az memória referenciás adatmozgató utasítások mnemonikja többnyire:

MOV cél, forrás

vagy LOAD cél, forrás

vagy LD cél, forrás stb.

A stack kezelő utasítások is ide sorolhatók:

PUSH forrás a stack tetejére ír

POP cél a stack tetejéről olvas

(A cél és forrás megadásáról, vagyis a különféle címzési módokról később esik szó.)

Az I/O utasítások mnemonikja:

IN cél, forrás beolvas a perifériából
OUT cél, forrás kiír a periférába

Adatkezelő utasítások

Az adatkezelő utasítások valamilyen manipulációt hajtanak végre a forrás operanduson. A művelet eredménye (op1-el jelölve) többnyire a CPU akkumlátorában keletkezik, így az akkumlátor az operandus tárolására is szolgál és célregiszter is.
A művelet típusa szerint a következő csoportok állíthatók fel:

Aritmetikai utasítások

Összeadás: ADD op1, op2
Kivonás: SUB op1,op2
A fenti utasítások carry flag-et figyelembe vevő változata is mindig létezik.

Szorzás: MUL op1,op2
Osztás: DIV op1,op2

Logikai utasítások

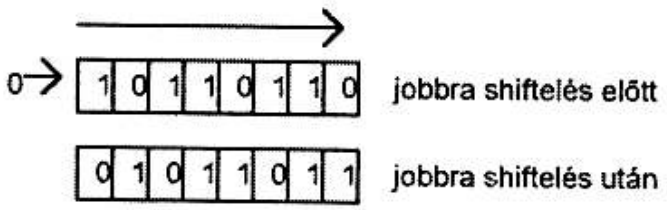
Bitenkénti ÉS: AND op1, op2
Bitenkénti VAGY: OR op1,op2
Bitenkénti XOR: XOR op1,op2

A bitenkénti logikai utasítások az operandusok azonos sorszámú bitjei között hajtódnak végre, párhuzamosan.

Shiftelő, rotáló utasítások

A shiftelő utasításoknál az adat i-edik bitje jobbra shiftelésnél az i-1-edik bit pozícióba másolódik s a legnagyobb helyiértékű bit törlődik (4.22. ábra).

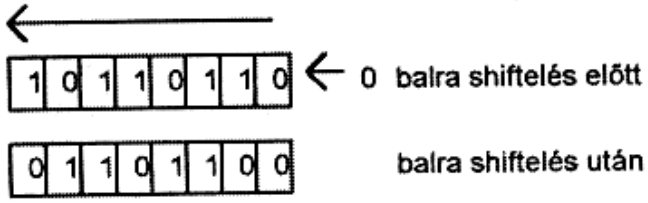
SHR op



4.22. ábra. Jobbra shiftelés

Balra shiftelésnél az i+1-edik bit pozícióba másolódik az i-edik és a legkisebb helyiértékű bit törlődik (4.23. ábra).

SHL op

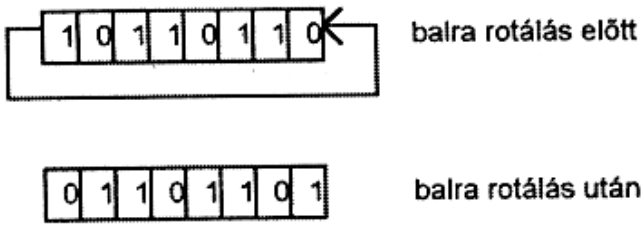


4.23. ábra. Balra shiftelés

A rotálás olyan shiftelés. ahol a legnagyobb és legkisebb helyiértékű bitet egymás folytatásának kell tekinteni.

RR op jobbra rotálás

RL op balra rotálás (4.24. ábra)

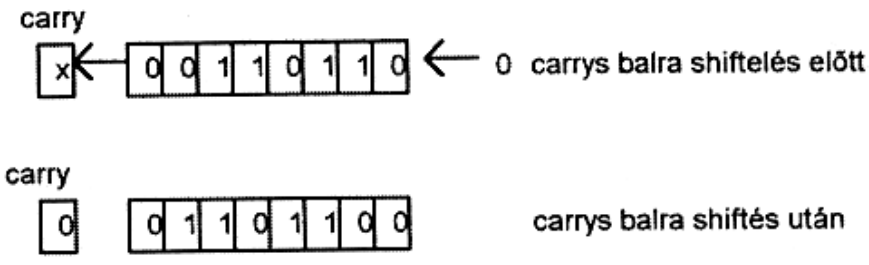


4.24. ábra. Rotálás

Többnyire létezik olyan utasítás, ahol a shiftelés/rotálás irányától függően a legkisebb vagy legnagyobb helyiértékű bit bemásolódik az átvitel bitbe (carrybe).

SLC op jobbra shiftelés carryvel

SRC op balra shiftelés carryvel (4.25. ábra)

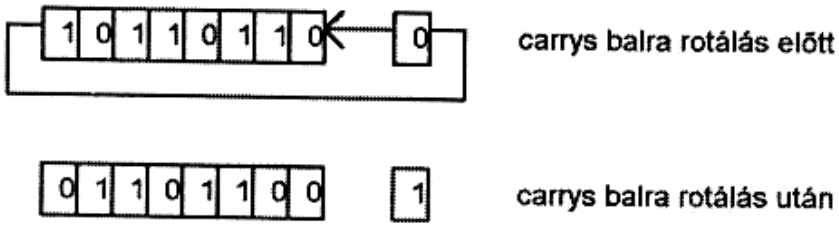


4.25. ábra. Shiftelés carryvel

Általában létezik olyan utasítás, ahol a rotálás a carryn keresztül történik.

RRC op jobbra rotálás carryn keresztül

RLC op balra rotálás carryn keresztül (4.26. ábra)



4.26. ábra. Rotálás carryn keresztül

Bit kezelő utasítások

A bit kezelő utasítások valamely regiszter, memória esetleg periféria bitet képesek 1-be ill. 0-ba állítani, invertálni, esetleg valamely flagbe átmásolni.

SETB	bit	1-be állítja a bitet
CLR	bit	törli a bitet
TEST	bit	a bitet bemásolja a Z flagbe
MOV	C,bit	a bitet bemásolja a carrybe

Vezérlés átadó utasítások

Ezen utasítások hatására a valamely feltétel flagtől függően vagy attól függetlenül a következő végrehajtandó utasítás nem a következő címen levő utasítás lesz.

Feltétel nélküli ugrás:

JUMP cím

vagy JMP cím

vagy JP cím stb.

Hatására a megadott címen folytatódik a program végrehajtás (PC=cím).

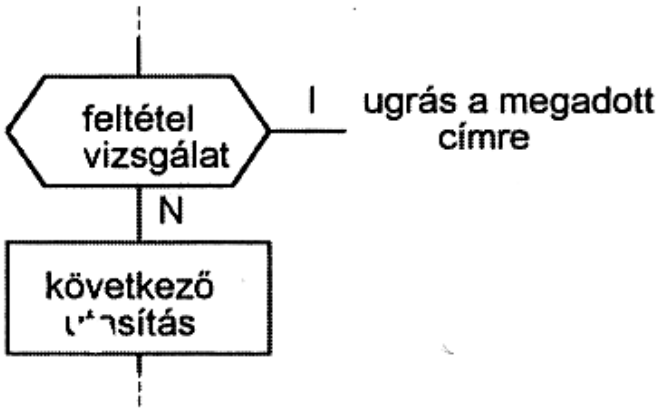
Többnyire létezik olyan típusuk is, ahol az ugrási távolság korlátozott (pl. +127, -128 byte), ezek a relatív ugrások. Ilyenkor a következő utasítás helye a PC aktuális értékéhez képesti távolsággal van megadva (PC relatív cím). Feltételes változatai is léteznek.

BR relatív cím

Feltételes ugrás:

JUMP feltétel, cím

Itt az ugrás csak akkor hajtódik végre, ha a feltétel teljesül, egyébként a következő utasításra adódik a vezérlés (4.27. ábra). Feltételként többnyire egy flag 0, vagy 1 értékét lehet előírni.



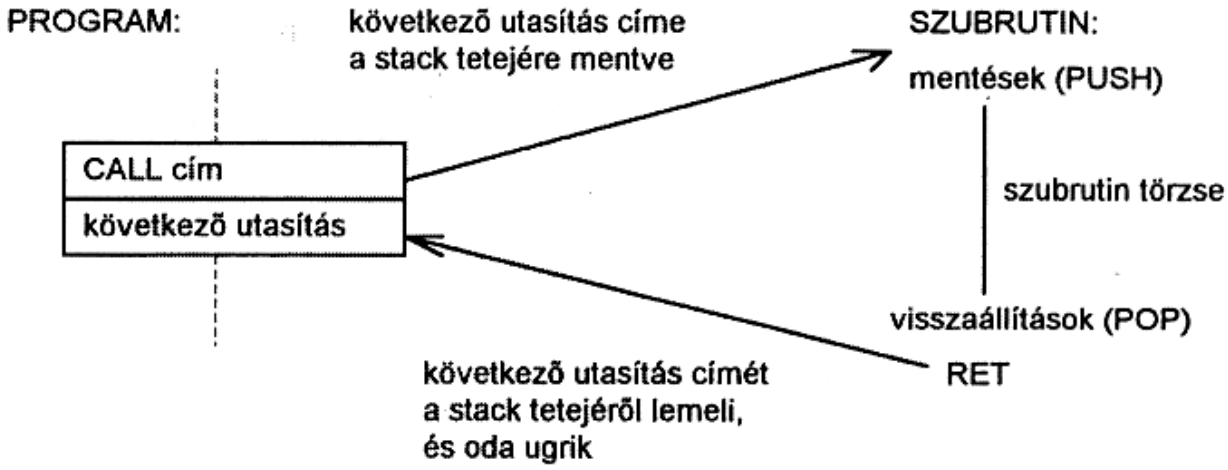
4.27. ábra. Feltételes ugrás

(Pl: JUMP NZ, cím ugorj, ha a zero flag értéke 0.)

Szubrutin hívások

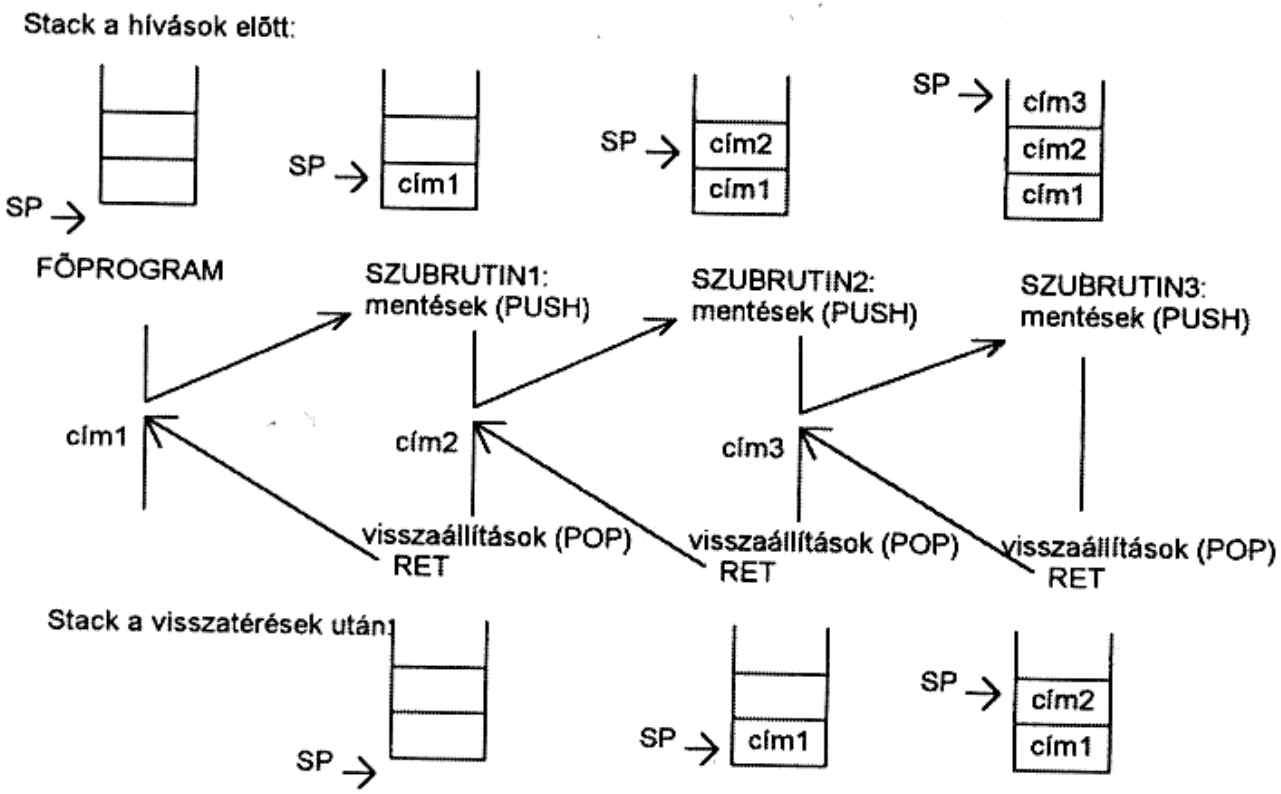
A programozás során sokszor előfordul, hogy bizonyos programrészeket azonos formában a program több különböző helyén kell végrehajtani. Ilyenkor, ha a programrészlet mérete nem túl kicsi, azt szubrutinként célszerű megírni. A szubrutinra a program tetszőleges helyéről lehet hivatkozni az ún. szubrutin hívással (CALL). Ez az utasítás a stack tetejére menti a következő utasítás címét, majd a szubrutin elejére ugrik. A szubrutinból visszatérő utasítás (RET) leveszi a stack tetejéről a visszatérési címet, majd oda ugrik, vagyis visszatér a félbeszakított program folytatásához (4.28. ábra).

A szubrutin használattal memóriát lehet spórolni, ahhoz képest, mintha minden egyes hívás helyett az egész programrészletet leírnánk. Azonban a szubrutinban használt regisztereket annak elején célszerű elmenteni (pl. PUSH utasításokkal a stack tetejére), majd a visszatérés előtt visszaállítani, mert akkor nem kell minden hívásnál külön arra figyelni, hogy mely éppen használt regisztereket ront el a szubrutin. A mentések és a visszatérő (RET) utasítás a legkisebb szubrutinban is ott vannak, ezért a túl kis méretű szubrutinok nem hatékonyak.



4.28. ábra. A szubrutin működése

A szubrutin hívásokat a stack terület által korlátozott mértékben egymásba lehet ágyazni, vagyis egy-egy szubrutinból újabbakat lehet hívni (4.29. ábra). Ezt a mechanizmust a stack teszi lehetővé. Ha a szubrutin által használt összes változót, bemeneti paramétert, visszatérési értéket és regisztert a stack tetején tároljuk, akkor ún. *rekurzív* (önmagára hivatkozó) szubrutin hívások is lehetővé válnak. Bizonyos algoritmusok nagyon röviden kódolhatók ilyen módon. A visszatérés ilyenkor mindig feltételes. A szubrutin mindaddig meghívja önmagát, amíg nem teljesül a visszatérési feltétel, ettől kezdve pedig minden hívásból visszatér. Az ilyen szubrutin tervezése, belövése nagy odafigyelést igényel, mert a stack nagyon meghízhat a visszatérési feltétel teljesülése előtt, s nem megfelelő tervezés esetén felülírhatja a nem stack területen elhelyezkedő változókat. Ezért a rekurzív szubrutinok alkalmazását inkább kerüljük el.



4.29. ábra. Szubrutin hívás-visszatérés folyamata

Feltétel nélküli szubrutin hívás:

CALL cím

Az utasítás először a stack tetejére menti a következő utasítás címét, majd a megadott címre ugrik.

Feltételes szubrutin hívás:

CALL feltétel, cím

A szubrutin hívás csak a feltétel teljesülése esetén következik be.

Feltétel nélküli visszatérés szubrutinból:

RET

Az utasítás hatására a STACK tetejéről a PC-be kerül a visszatérési cím, és a következő utasítást erről a címről veszi a CPU. Feltételes változata is létezik. Külön meg kell említeni az interruptból való visszatérést (RETI) és a nem maszkolható interruptból való visszatérést (RETN), de ezekről részletesebben majd az interrupt tárgyalásánál szólunk.

Feltételes visszatérés szubrutinból:

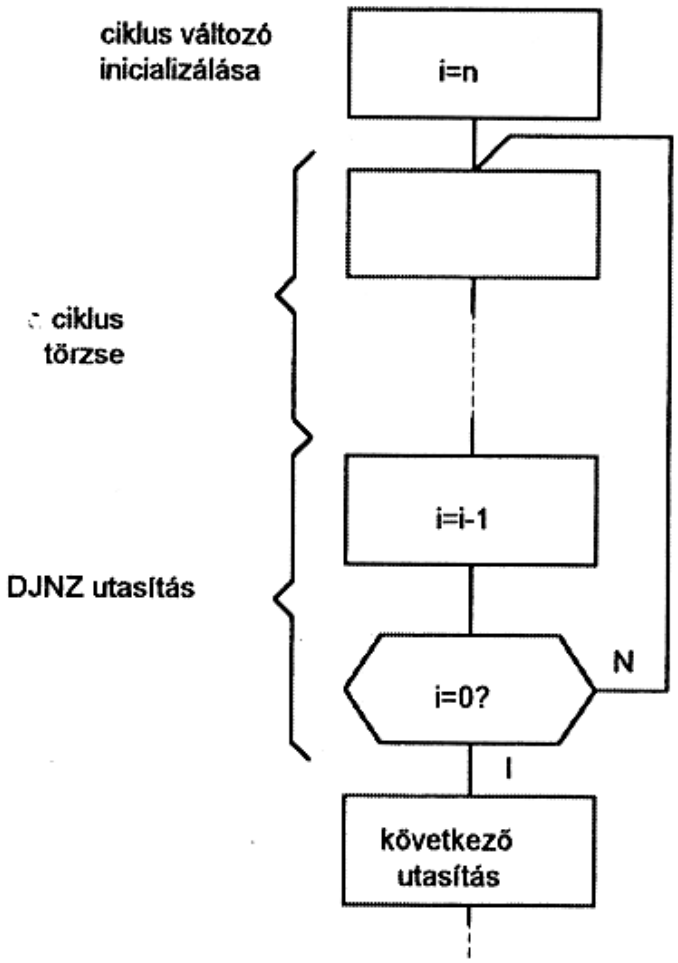
A feltétel teljesülése esetén a CPU visszatér a szubrutinból, egyébként a következő utasításnál folytatja a program végrehajtását.

RET feltétel

Ciklus szervező utasítások

Ezekkel az utasításokkal egy program részlet előre megadott számú ismétlése oldható meg. Az utasításhoz tartozik egy ciklus változó, melynek helye előre definiált (pl. egy adott regiszter). A ciklikusan ismételendő program részlet előtt a ciklus változót be kell állítani. Az utasítás a ciklus végén helyezkedik el. Dekrementálja a ciklus változó értékét, s ha az még nem 0, akkor a megadott címre (a ciklus elejére) ugrik (4.30. ábra).

DJNZ relatív cím



4.30. ábra. Ciklus szervező utasítás működése

CPU vezérlő utasítások

NOP: Időhúzó utasítás, nem csinál semmit (No operation). Ciklusban, vagy többször egymás után alkalmazva programból lehet időzítést előállítani. A processzor órajelének és az utasítás gépi ciklus igényének (hosszának) ismeretében pontosan számítható az időzítés hossza. Az utasítás órajel periódusokban mért hosszát a katalógusok megadják.

HALT: Hatására interruptra várakozó állapotba kerül a processzor, NOP-okat hajt végre, amíg egy interrupt be nem következik, vagy RESET jelet nem kap a CPU.

EI: Globális interrupt engedélyező utasítás.

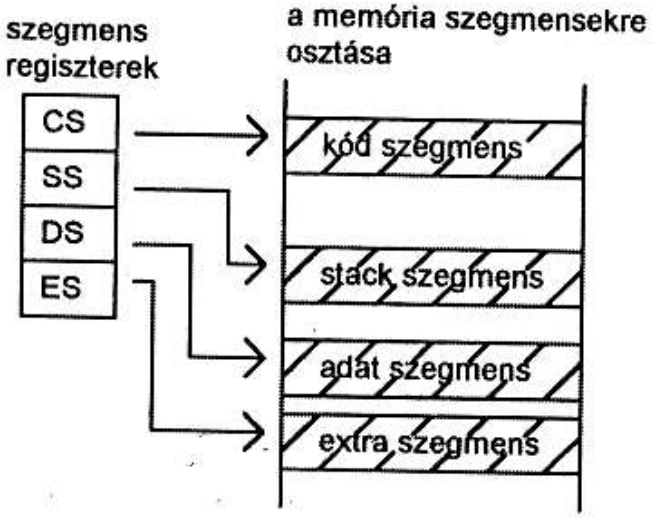
DI: Globális interrupt tiltó utasítás. Hatására a CPU az NMI-n (nem maszkolható megszakításon) kívül nem fogad el megszakítást.

CPU-tól függően egyéb vezérlő utasítások is léteznek.

4.5. Címzési módok

Azt, hogy hogyan lehet elérni az operandust, a címzési módok határozzák meg. A processzorok egy részénél a címbuszra kiadott fizikai cím a címzési mód alapján közvetlenül keletkezik (pl. Z80), más részénél egy ún. szegmens regiszter is résztvesz a címképzésben (pl. I8086). Ez utóbbi esetben a fizikai cím címzési módtól függő részét *effektív cím*nek nevezik.

Szegmens szervezésű memória esetén az egyes szegmensek egy kisebb memória tartományt jelölnek ki (I8086 esetén 64 kbyte), s csak az ezen belüli címekhez lehet hozzáférni a szegmens regiszter megváltoztatása nélkül (4.31. ábra).



4.31. ábra. Szegmens szervezésű memória

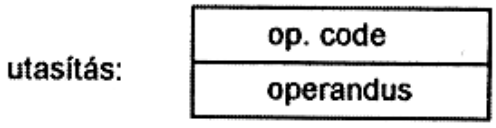
A címzési módoknál a következő alaptípusokat szoktuk megkülönböztetni.

Implied (magában foglalt)

Az operandus címét az utasítás kódja imlicit módon határozza meg. (Pl. a művelet csak az akkumlátorral végezhető el.)

Közvetlen (immediate) címzés

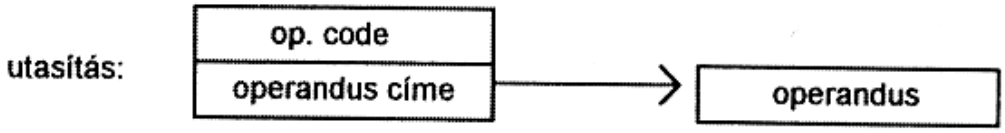
Itt maga az operandus az utasítás része (4.32. ábra). (pl. Z80-nál a LD a, #55 utasításban az utasítás kódja 3E 55.)



4.32. ábra. Az operandus elhelyezkedése közvetlen címzésnél

Direkt címzés

Itt az operandus címe (vagy effektív címe) az utasítás része, tehát az operandus csak a megadott címről való kiolvasás után áll rendelkezésre (4.33. ábra). (pl. Z80-nál LD a, (1C00) kódja 3A 00 1C)



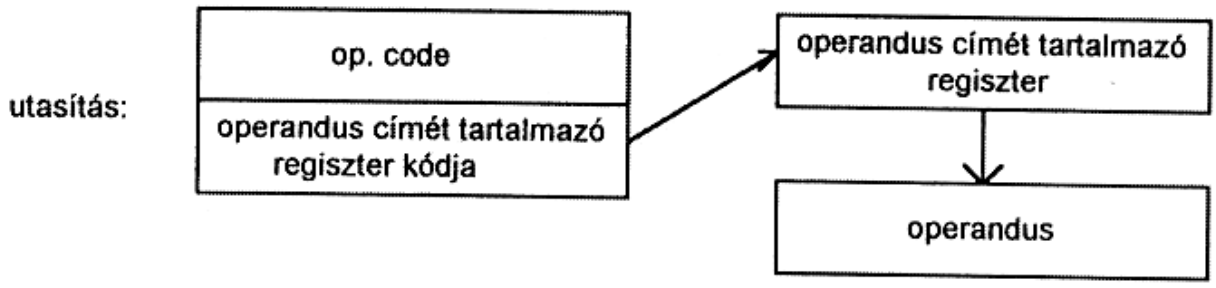
4.33. ábra. Az operandus elhelyezkedése direkt címzésnél

Regiszter (direkt) címzés

Itt az utasítás kódja (annak adott bitjei) jelöli ki a regisztert vagy regisztereket, amelyek az operandust tartalmazzák. (pl. Z80-nál LD a,b)

Regiszter indirekt címzés

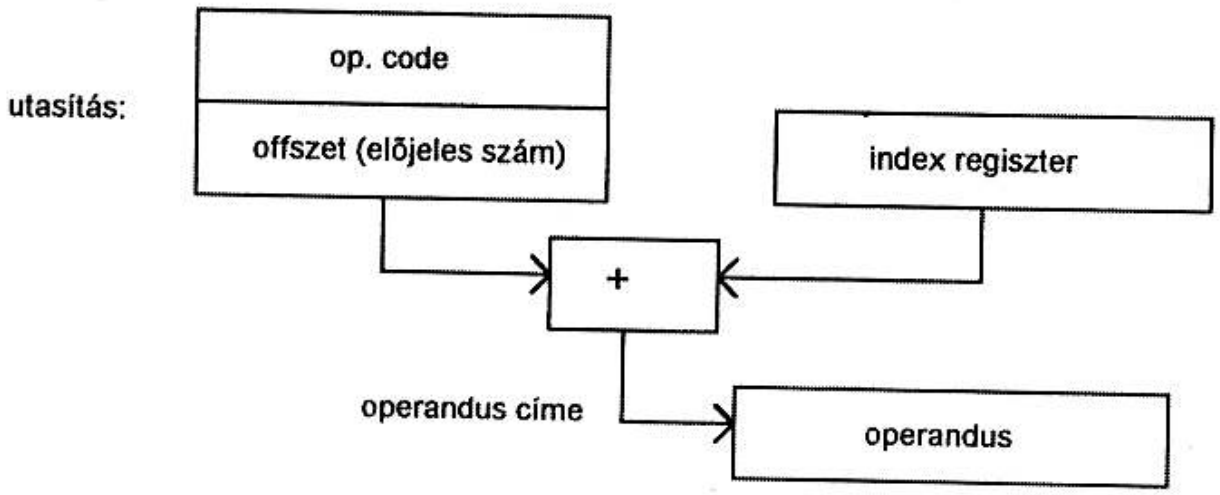
Ez az előbbihez hasonló, de a kijelölt regiszter nem az operandust, hanem annak a címét (vagy effektív címét) tartalmazza (4.34. ábra). (pl. Z80-nál LD a,(HL))



4.34. ábra. Az operandus elhelyezkedése indirekt címzésnél

Indexelt címzés

Itt az utasítás kódja egyrészt kijelöl egy index regisztert, másrészt tartalmaz egy 2-es komplemensben értelmezett konstanst (displacement, eltolás). A CPU a konstanst és a kijelölt indexregiszter tartalmát összeadja, s ez az összeg lesz az operandus címe (vagy effektív címe). Ezt mutatja a 4.35. ábra. Az adatblokkok (vektorok, mátrixok) kezelését könnyíti meg. (pl. LD a,(IX-3)



4.35. ábra. Az operandus elhelyezkedése indexelt címzésnél

Az utasításokban a különféle címzési módok vegyesen fordulnak elő. Mindig létezik forrás és rendeltetési cím, de olykor a kettő megegyezik.

4.6. A program megszakítás (interrupt)

Bizonyos esetekben a periféria kiszolgálása kényelmesebb, ha maga a periféria jelzi a kiszolgálási igényét. Ilyenkor a főprogramban nem kell a perifériával törődni (időnként lekérdezni), hanem a jelzés hatására időlegesen abbamarad a főprogram futása és automatikusan a perifériát kezelő program rész (periféria handler) aktivizálódik.

A perifériák a processzor IT kérő bemenetén keresztül jelzik, ha valamilyen kiszolgálást igénylő esemény történt (pl. befejeződött valamely működés, kiolvasandó adat keletkezett, újabb adat küldhető ki, hiba történt stb.).

Ha teljesülnek az interrupt elfogadás feltételei, s a CPU észlel egy megszakítás kérést, akkor *az aktuális utasítás végrehajtását befejezi*, s a következő utasítás végrehajtása helyett, *az interrupt kiszolgáló szubrutinra adja a vezérlést.*

A perifériát kiszolgáló szubrutin címének meghatározása processzor ill. hardver függő, és előfordul, hogy egy processzoron belül is több módszer közül lehet választani (pl. Z80). Maga az interrupt rutin némileg hasonlít egy közönséges szubrutinhoz, a különbség az aktivizálás módjában és az induláskor ill. visszatéréskor elvégzendő feladatoknál van.

Ezek a feladatok a következők:

- Az interrupt rutin első utasításának végrehajtása előtt a CPU automatikusan a stack tetejére menti a visszatérési címet és esetleg a CPU interrupt előtti állapotára vonatkozó információkat tartalmazó állapotszót, továbbá többnyire letiltja az interrupt elfogadást a CPU-ban. (Ez a tiltás processzor függő, pl. a Z80, I8086 letiltja, de a MC8031/51 nem. Az utóbbinál az interrupt program első utasítása lehet egy interrupt tiltás, melynek végrehajtása után már nem szakítható meg a program futása újabb engedélyezésig.)
- A periféria kezelését az IT rutin elvégzi, majd egy speciális (return from interrupt, RETI) utasítás végrehajtása után visszatér a megszakítás észlelését követő utasításhoz. A RETI utasítás egyrészt a CPU automatikusan elmentett állapotszavát állítja vissza (ha volt ilyen), másrészt a visszatérési címet veszi le a stack tetejéről, s erre a címre adja a vezérlést. (Bizonyos esetekben a RETI az IT-s perifériákra is hatással van.)

Az IT rutin írójának kell gondoskodnia a *CPU állapotának* (rutinban felhasznált CPU regiszterek) *elmentéséről* az IT rutin elején és *visszaállításáról* a rutin végén. A mentés elmaradása be nem látható következményekhez vezet, mivel az IT rutin elronthatja a megszakított program által használt regiszterek, flagek értékét. A regisztereket fix memóriahelyre, vagy a stackre lehet elmenteni, az utóbbi programozási szempontból kényelmesebb. Egyes processzoroknál (Z80, MC8031/51) egy alternatív regiszter készletet is fel lehet használni a mentésre, azonban ekkor egyrészt ezt a regiszter készletet csak az IT rutinok használhatják, másrészt ilyenkor nem megengedhető, hogy az egyes IT rutinok egymást megszakítsák, csak ha a főprogramhoz, és minden megszakítási rutinhoz külön regiszter készlet rendelhető.

Az interrupt kérést annak kiszolgálásáig fenn kell tartani, csak akkor garantált a kiszolgálása. Az IT kérés általában a megszakítási rutinban, a perifériának adott paranccsal szüntethető meg.

A megszakítás engedélyezése, tiltása

A megszakítás kérés általában az alábbi helyeken engedélyezhető:
- CPU: Itt globálisan lehet engedélyezni vagy tiltani az interrupt elfogadását.
- Interrupt vezérlőben is lehetőség van az interrupt kérés tiltására.
- Periféria: Egyedileg is lehetséges a perifériák interrupt kérésének engedélyezése és tiltása.

Az NMI és funkciója

A processzorok általában rendelkeznek egy nem tiltható interrupt bemenettel is (**NMI** No Maskable Interrupt). A processzor ezen bemenetén érkező kérés a normál IT-től eltérően a processzoron belül nem tiltható le, az mindenképpen érvényre jut.

Az NMI-hez többnyire a rendszer működését veszélyeztető események kezelését szokás rendelni. Ilyen a tápfeszültség jelentős csökkenése, észlelhető hardver hibák (RAM paritás hiba) stb. Az NMI rutin ilyenkor egyrészt jelzést ad a felhasználónak,

másrészt menti a menthetőt (pl. tápfeszültség csökkenés esetén nem felejtő RAM-ba menti azokat az információkat, amelyek újraindításkor a hiba előtti állapot visszaállításhoz szükségesek).

Az NMI vonal élérzékeny, vagy impulzus érzékeny. (Ha szint érzékeny lenne, folyamatosan megszakítaná önmagát az NMI rutin, amíg aktív az NMI bemenet.) Ebből következik, hogy nem szükséges a kiszolgálásig fenntartani az NMI kérést.

A következőkben az interrupt rendszerek különféle csoportosításáról lesz szó. Ennek egy részét foglalja össze a 4.36. ábra.

4.36. ábra. Interrupt rendszerek csoportosítása

4.6.1. Az interrupt források csoportosítása

Külső IT források

A külső források a CPU valamely bementén keresztül váltanak ki interruptot. Ilyenek az IT (maszkolható külső interrupt kérő vonal) és az NMI (nem maszkolható IT kérés).

Belső IT források

Egyes processzorokban belső események is okozhatnak megszakítást (pl. I8086):

- osztási hiba (0-val történő osztás),

- túlcsordulás (számábrázolás határának átlépése),

- Bizonyos processzoroknál lehetőség van lépésenkénti program végrehajtás (single step) beállítására. Ebben az esetben minden utasítás végrehajtás után megszakítás történik.

- Vannak olyan CPU-k, amelyek valamely címen elhelyezkedő utasítására ún. töréspontot (break point) lehet tenni. Ebben az esetben az adott utasítás végrehajtása után szintén interrup következik be.

A lépésenkénti program végrehatást és töréspont funkciókat a program fejlesztést segítő hiba kereső (debugger) programok használják.

Belső megszakítások esetén a különböző IT okokhoz különböző interrupt megszakítási rutin kezdőcímek rendelhetők.

Szoftver megszakítások

Némely processzorban a megszakítások szoftverből is kiválthatók, speciális utasítások segítségével (pl. I8086, INT i utasítás). Az ilyen megszakítások azonban inkább speciális szubrutin hívásnak tekinthetők, hiszen nem szakítják meg egy program futását. Az elnevezés inkább arra utal, hogy a meghívott rutin címének előállítása hasonló módon történik, mint a szokásos interrupt esetén.

4.6.2. Az IT rutin kezdőcímének meghatározása

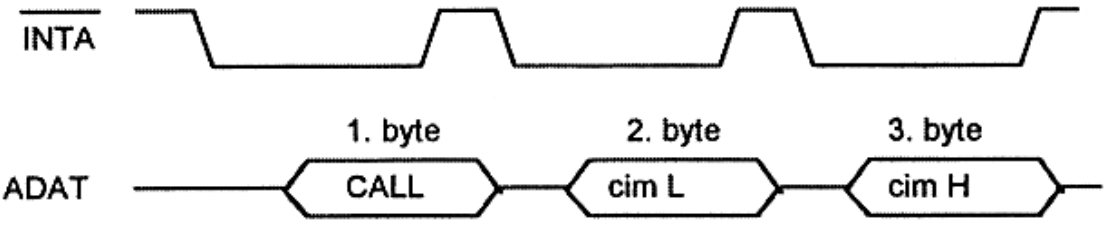
Egy rendszerben több interrupt forrás (interrupt kérő eszköz) lehet. Ezért egy interrupt bekövetkezésekor először meg kell határozni, hogy az aktuális interrupt kérés hatására melyik (milyen címen kezdődő) interrupt rutin induljon el. Ennek módszerei a következők:

Fix újraindítási cím

A CPU minden interrupt kérő vonalához egy előre definiált interrupt rutin cím tartozik, azon nem lehet változtatni (pl. I8085, MC8051).

Utasítás beolvasás

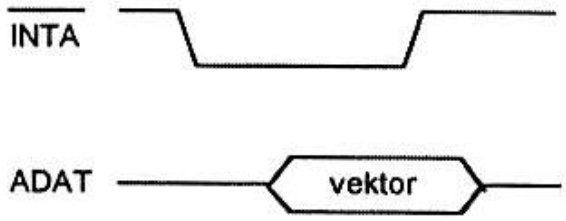
A CPU az ún. interrupt elfogadási ciklus (INTA) alatt az adatbuszon egy utasítás kódot vár. Ez célszerűen az IT rutin címére szóló szubrutin hívó utasítás kódja (4.37. ábra). (A CPU bármilyen más utasítás kódot is elfogad és végrehajt).



4.37. ábra. Utasítás beolvasásos INTA ciklus (3 byte-os CALL)

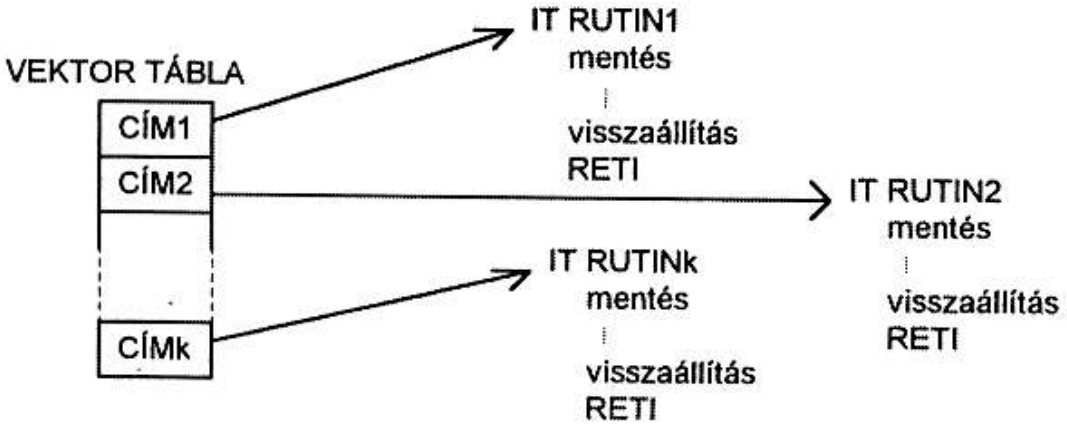
Vektor beolvasás

Hasonló a parancs beolvasáshoz, de itt a CPU az INTA ciklus alatt az adatbuszon egy ún. interrupt vektort vár (4.38. ábra). A vektor alapján a CPU valamilyen módszerrel meghatározza az IT rutin kezdőcímét és oda ugrik.



4.38. ábra. Interrupt vektor beolvasása

A vektorból először egy memóriacím generálódik. A vektorból előállítható címek egy összefüggő, kis méretű (0.5 - 2 kbyte) tartományt alkotnak, amit **interrupt vektor táblának** neveznek. A tartomány kezdőcíme többnyire programozható az interrupt vezérlőben (pld. I8259A), vagy a processzorban (pld. Z80). A vektor táblában vannak eltárolva az egyes interruptokhoz tartozó rutinok kezdőcímei. A processzor a táblázatból automatikusan előveszi a megfelelő címet, s a következő utasítás címének stackre mentése után oda adja a vezérlést (4.39. ábra).



4.39. ábra. Interrupt vektor tábla

Az utasítás beolvasásos és vektor beolvasásos rendszert összefoglaló néven gyakran *vektoros IT rendszer*nek nevezik.

4.6.3 Az interrupt kérő eszköz azonosítása

Az interrupt kiszolgáló program kezdőcímének azonosítása nem minden esetben jelenti, hogy egyben az interrupt kérő eszköz is ismert. Ebből a szempontból három féle interrupt rendszert különböztetünk meg.

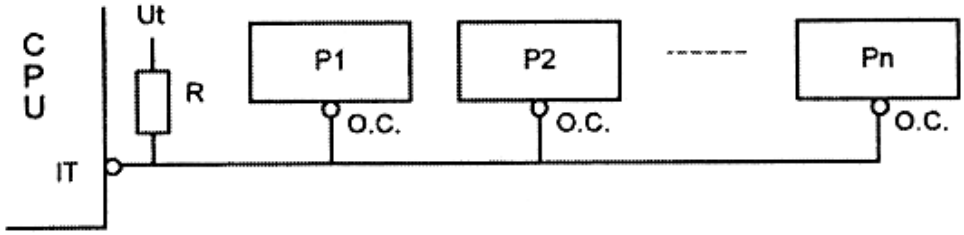
Programozott lekérdezéses IT forrás azonosítás

Ebben a rendszerben az IT forrás megállapítása a program feladata. Erre elterjedten használják az **egyszerű IT rendszer** elnevezést, mivel ez a megoldás igényli a legegyszerűbb hardvert több IT-s periféria kezelésére.

Erre példa, ha egy CPU egyetlen, fix újraindítási címhez rendelt IT vonalára több interrupt kérő eszköz csatlakozik (4.40. ábra). Ilyen interrupt rendszernél minden IT

forrás kérésére ugyanarra a kezdőcímre adódik a vezérlés. Ekkor perifériák állapotának lekérdezésével azonosítható az IT kérő eszköz, s ez alapján ágazik el a program az adott eszközt kiszolgáló részre. Így, ha több egység kér egyszerre interruptot, a kiszolgálás sorrendjét vagyis az egyes perifériák prioritását is a program határozza meg.

Ebben az esetben az IT vonalat nyitott kollektorosan kell meghajtani így a VAGY kapcsolat automatikusan létrejön. Nem szabad elfeledni, hogy a processzor IT bemenetére egy felhúzóellenállás szükséges, ha nincs belső felhúzás beépítve. Az egyszerű IT rendszer egyik hátránya, hogy a tényleges interrupt rutin végrehajtása előtt mindig lefut az IT forrás azonosítását végző program rész, ami megnöveli a perifériák kiszolgálási idejét. Másik hátránya, hogy nem alakítható ki igazi többszintű IT rendszer (lásd később), ami bizonyos esetekben elengedhetetlen.



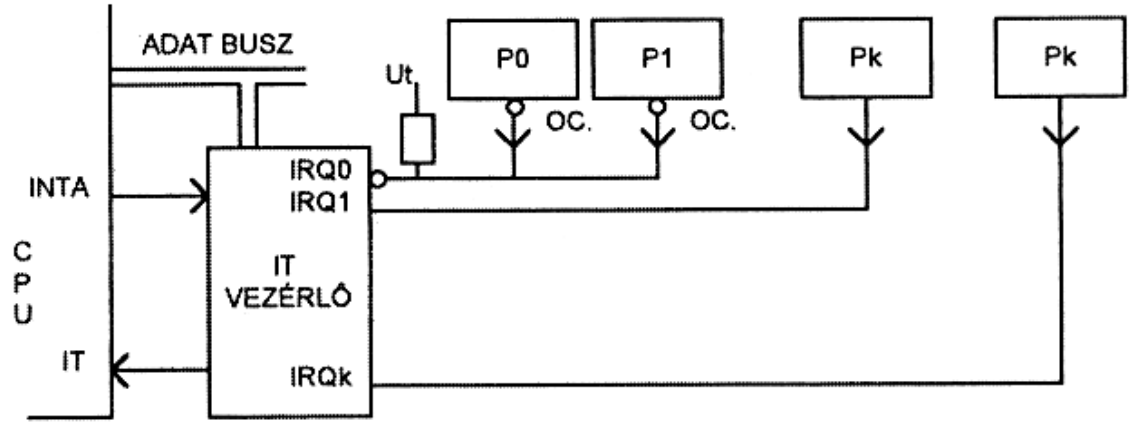
4.40. ábra. Példa egyszerű IT rendszerre

Automatikus megszakítási forrás azonosítás

Ebben az esetben az interrupt forrás felismerése automatikusan történik meg. Ekkor minden perifériához saját interrupt rutin kezdőcím van rendelve. Ilyen a vektoros IT rendszer.

Vegyes megszakítási rendszer

Az interrupt forrás azonosítása szempontjából vegyes megszakítási rendszerről beszélünk, ha egyes periféria csoportokhoz külön IT rutin kezdőcím van rendelve, de a csoport tagjainak azonosítását programból kell elvégezni. Ez a rendszer áll elő, ha egy processzoros rendszerben több IT kérő vonal van, külön interrupt rutin kezdőcímekhez rendelve, de egy-egy vonalra több eszköz is kapcsolódik (nyitott kollektorosan) Pl. a 4.41. ábra egy ilyen rendszert mutat.



4.41. ábra. Vegyes IT rendszer

4.6.4. Az interrupt rutinok megszakíthatósága

A több interruptos perifériát tartalmazó rendszerekben az egyes IT kérő eszközök jelentősen eltérő tulajdonságokkal rendelkezhetnek. Egyes perifériák kiszolgálása lehet halasztást nem tűrő (pl. nagy sebességű soros vonalon adatblokk érkezik), de lehet hosszabb kiszolgálási időt elviselő is (pl. msec-onként bekövetkező timer interrupt). A halasztást nem tűrő eset lehet olyan sürgős, hogy egy kevésbé fontos interrupt megszakítására is szükség van. Ezért az interrupt rutin megszakíthatóságának szempontjából két lehetőség közül választhatunk, s ez alapján kétféle rendszert különböztethetünk meg.

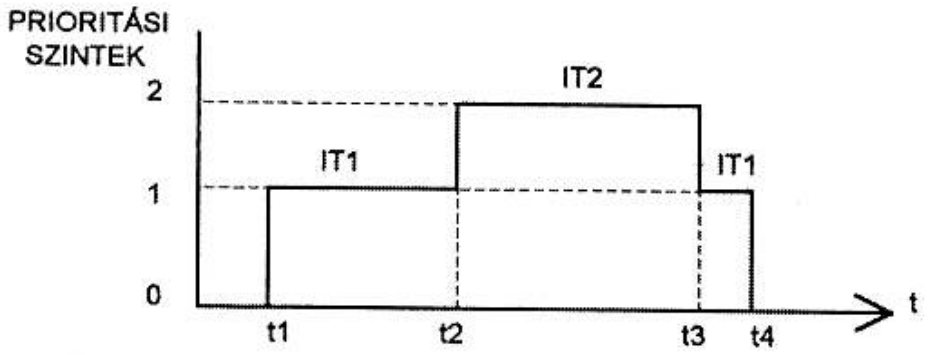
Egyszintű interrupt rendszer

Az éppen futó interrupt rutint nem szakíthatja meg egy másik interrupt. Ilyen rendszerben előfordulhat, hogy egy nagyon sürgős kérést nem szolgálnak ki időben. Az egyszerű interrupt rendszer egyben egyszintű is.

Többszintű interrupt rendszer

Ebben az esetben az egyes interruptos perifériák ún. *prioritási szintek*hez vannak rendelve. Az aktuálisan futó interrupt rutint megszakíthatja egy magasabb prioritású interrupt.

A 4.42. ábrán 2 IT forrás szakíthatja meg a főprogramot, ill. a magasabb prioritású a kisebbet (a főprogram a legkisebb prioritású, azt mindenki megszakíthatja). A prioritás növekvő sorrendben IT1, IT2. Itt t1 időpontban IT1 megszakítja a főprogramot, t2-ben IT1-et megszakítja IT2, t3-ban befejeződik IT2 és folytatódik IT1, t4-ben befejeződik IT1 és folytatódik a főprogram.



4.42. ábra. Többszintű IT rendszer működése

Ha az interrupt elfogadás hatására az interrupt engedélyezés a processzorban letiltódik (pl. Z80, I8086) akkor az interrupt rutinban kell gondoskodni annak engedélyezéséről, amennyiben többszintű rendszert akarunk létrehozni.

Az IT forrás azonosítása szempontjából vegyes rendszerben a különböző IT vonalak különböző szintekhez tartoznak, közöttük az IT vezérlő által meghatározott prioritás érvényesül. Azon eszközök között azonban, amelyek egy IT vonalra csatlakoznak (nyitott kollektorosan), programból lehet prioritást kialakítani (az IT forrás azonosítása is programból történik).

Prioritási stratégiák

Ha egyszerre több kérés érkezik akkor el kell dönteni, hogy melyik legyen először kiszolgálva. Ez a probléma a mikroprocesszoros környezetben a megszakítás és a busz arbitráció esetén is felmerül. A megoldására különféle stratégiák terjedtek el.

Fix prioritás

Fix prioritás esetén minden kérő eszköznek előre definiált, megváltoztathatatlan prioritása van. Ennek a sémának hátránya, hogy esetleg egy vagy több nagyon aktív, magas prioritású kérő nem hagyja szóhoz jutni az alacsonyabb prioritásúkat.

Változó prioritás

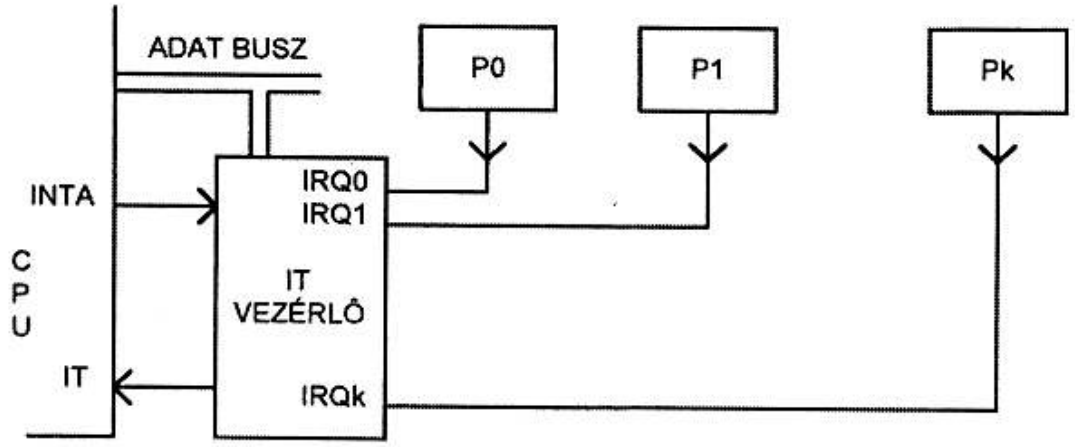
Ebben az esetben valamilyen algoritmus szerint változtatják a kérők prioritását, így előbb-utóbb mindenki sorra kerül. Az egyik leggyakrabban alkalmazott algoritmus, a *körben forgó* (round robin) prioritási stratégia. Ennél a kiszolgálandó eszközök egy sort alkotnak. A sorban legelöl álló a legnagyobb, a leghátul álló pedig a legkisebb prioritású. A legnagyobb prioritásúból kiszolgálása után legkisebb prioritású válik, (a sor legvégére kerül), s az utána következő lesz a legnagyobb prioritású.

4.6.5. Interrupt busz struktúrák

Ha több interrupt kérő eszközt engedünk meg egy mikroprocesszoros rendszerben, melyek prioritását nem programozott lekérdezéssel kívánjuk eldönteni, akkor ehhez interrupt vezérlő (hardver) szükséges.
Az interrupt vezérlő lehet önálló egység (centralizált), vagy elhelyezkedhet a perifériákban elosztva (decentralizált). A következőkben a legelterjedtebb struktúrákat mutatjuk be.

Centralizált vektoros IT rendszer

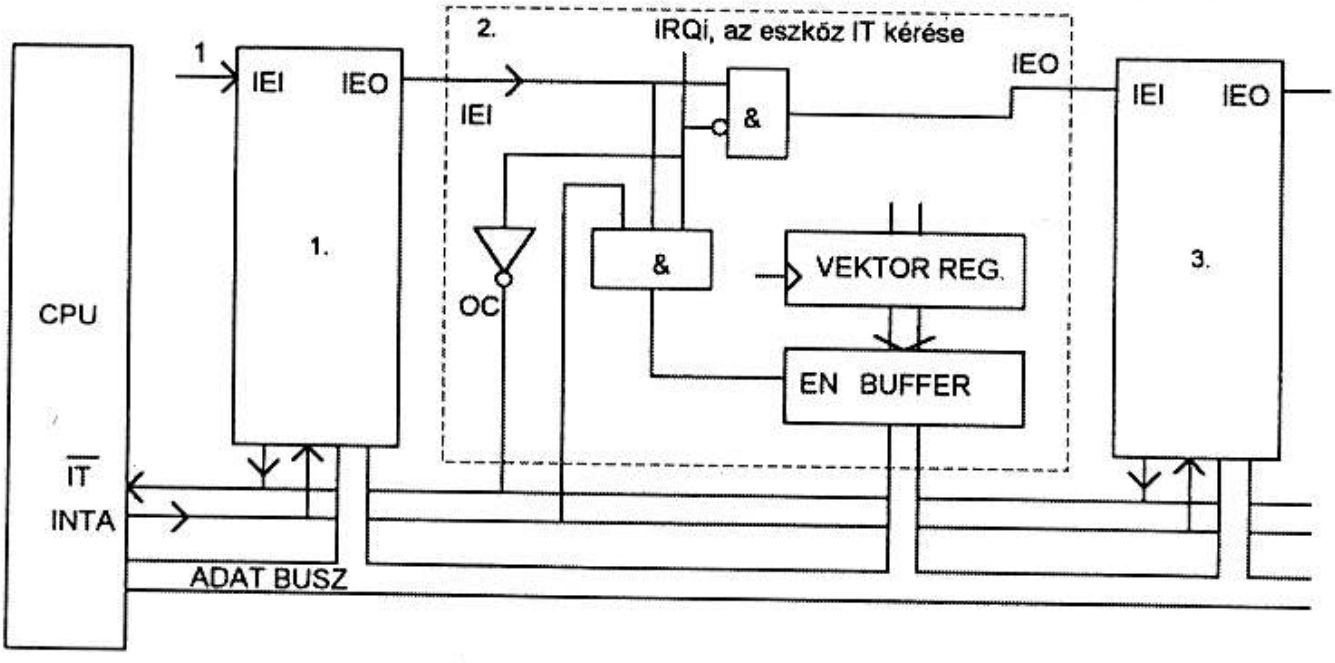
Ebben a rendszerben minden egyes IT kérő eszköztől külön interrupt vonal megy a vezérlőbe (4.43. ábra). A vezérlő minden egyes interrupt vonalhoz külön vektort generál. A prioritási algoritmus a vezérlőtől függ, általában programozható. Az ilyen struktúra hátránya, hogy sok interrupt vonalat igényel.



4.43. ábra. Centralizált vektoros IT rendszer

Vektoros daisy-chain interrupt rendszer

Ez egy decentralizált IT vezérlős rendszer. Minden eszköznek van egy interrupt engedélyező bemenete (IEI) és egy interrupt engedélyező kimenete (IEO). Ezeken a vonalakon keresztül az IT kérő eszközök láncba vannak fűzve (4.44. ábra). Így fix prioritási séma alakul ki. Az egyes eszközök prioritását a láncban elfoglalt helyük határozza meg. A láncban legelöl levő állandóan engedélyezve van, ennek legnagyobb a prioritása, az utána levőknek pedig egyre csökken. Az eszközöknek egyetlen közös interrupt kérő vonala van, melyre nyitott kollektorosan kapcsolódnak.Ha egy eszköznek kiszolgálási igénye van, akkor egyrészt letiltja a láncban utána állókat az IEO vonalán keresztül, másrészt lehúzza az IT vonalat. Ha az engedélyezés a láncon keresztül eljut hozzá, vagyis nincs nála nagyobb prioritású kérés, akkor az INTA ciklus alatt az adatbuszra kapuzza a vektorát.



4.44. ábra. Vektoros daisy-chain interrupt blokkvázlata

Az INTA ciklus alatt a prioritási láncot "befagyasztják" a vezérlők, vagyis az IEI-IEO lánc az INTA ciklus alatt nem változik (ez a logika nincs feltüntetve az ábrán). Erre azért van szükség, mert különben, ha egy INTA ciklus alatt, miközben egy alacsonyabb prioritású egység a vektort az adatbuszra helyezi, egy magasabb prioritású egység IT-t kér, akkor az alacsonyabb prioritású rögtön lekapcsolná a vektort a buszról, s a másik helyezné rá. Ez az INTA ciklus lefolyása közben tetszőleges időpontban bekövetkezhetne, s így semmi sem garantálná, hogy amikor a CPU az INTA ciklus végén mintavételezi az adatbuszt (beolvassa a vektort), akkor az stabil.

A 4.45. ábra idődiagramja azt az esetet mutatja, amikor az i-edik egység interrupt kérésének (IRQi) elfogadási ciklusa alatt bejön egy magasabb prioritású (a láncban előrébb levő) j-edik egység interrupt kérése (IRQj). Ekkor a következők történnek:

- Az i-edik egység interrupt kérése aktivizálja az IT vonalat és letiltja a láncba alatta levőt (IEOi).

- A processzor elfogadja a kérést és INTA jelet generál, amelyre az i-edik egység az adatbuszra teszi a vektorát. Ezután az i-edik egység megszüntetheti a kérését.

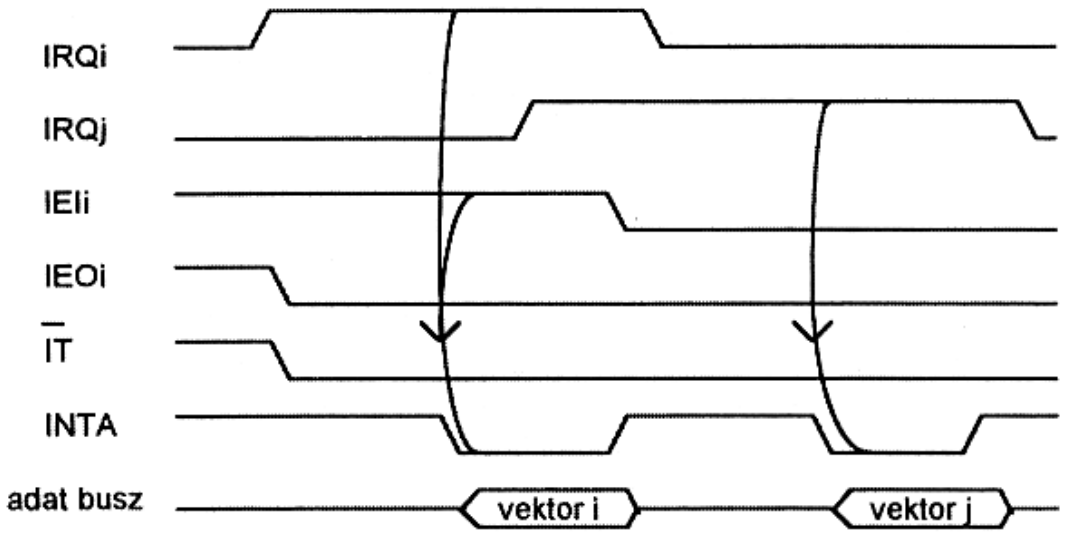
- A j-edik egység az i-edik egységnek szóló INTA alatt megszakítást kér (IRQj), de ennek nincs hatása addig, míg ez az INTA ciklus be nem fejeződött, mert a prioritási lánc ezalatt be van fagyasztva.

- Az i-edik egységnek szóló INTA után az i-edik egység engedélyező bemenete (IEIi) letiltódik.

- A processzor a stackre menti a visszatérési címet és az i-edik egység interrupt programjára adja a vezérlést.

- A processzor az i-edik egység megszakítási rutinjának első utasítása után elfogadja a j-edik egység interrupt kérését (INTA), mire az az adatbuszra teszi a vektorát. Ezután a j-edik egység megszüntetheti kérését.

- A CPU a visszatérési cím stackre mentése után j-edik egység interrupt kiszolgáló programjára adja a vezérlést.

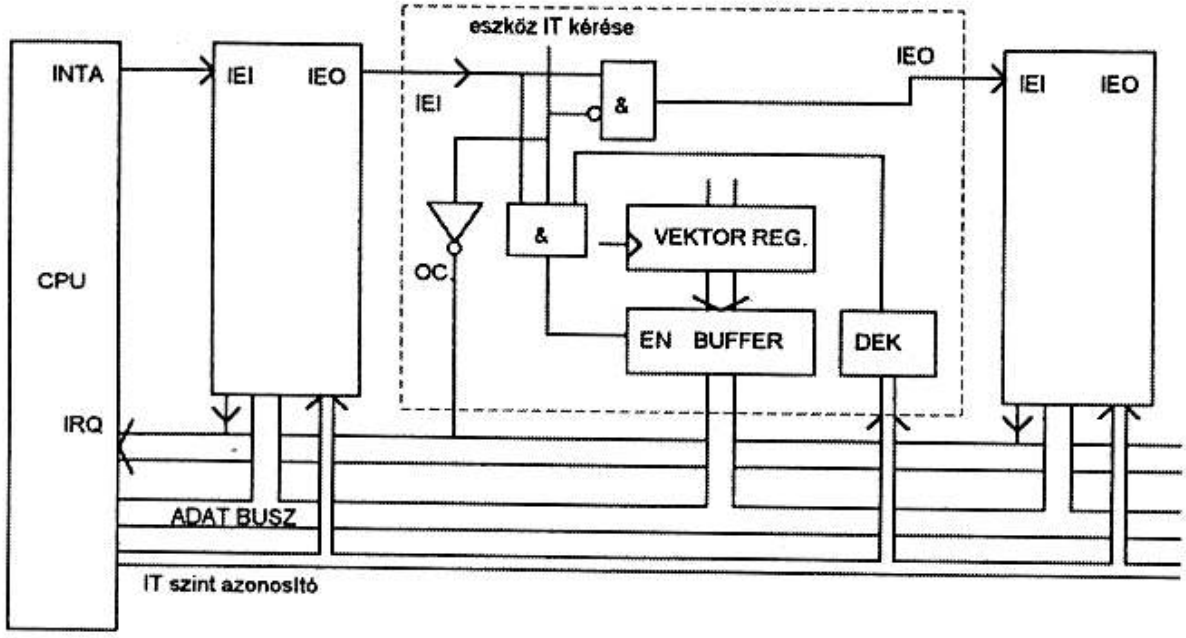


4.45. ábra. Vektoros daisy-chain interrupt idődiagramja

Többszintű vektoros decentralizált interrupt rendszer

Az interrupt engedélyezés itt is daisy-chain jellegű, azonban több interrupt kérő vonal van. A különböző interrupt vonalak különböző megszakítási szintekhez tartoznak. Az interrupt kérő vonalakon kívül, ún. interrupt szintet azonosító kódot továbbító jelek is vannak, erre a célra egyes rendszerek a címbusz néhány vonalát használják. Ezek a címvonalak kettős funkciót látnak el, általában címként funkcionálnak, INTA ciklus alatt pedig a szintazonosító kódot továbbítják.

Ha valamely eszközök interruptot kérnek és a CPU elfogadja, akkor kiadja az INTA-t és a szintazonosító kódot. A szintazonosító kód, az aktuálisan legnagyobb prioritási szintű interrupt vonal sorszámával egyezik meg.



4.46. ábra. Vektoros daisy-chain interrupt blokkvázlata

Az IT kérő eszközök figyelik a szint azonosító kódot, és amelyik egyezőnek találja a saját szintjével (az azonos interrupt vonalra kapcsolódó eszközök ugyanazt a kódot figyelik), és a láncon keresztül engedélyezést kap (a szinten belül nincs nagyobb prioritású kérés), az ráteszi a vektort az adatbuszra.

Ez a rendszer (4.46. ábra) lehetővé teszi, a interrupt vonalak közötti változó prioritást, a CPU-ban ill. CPU mellett levő interrupt logika segítségével. Az azonos IT kérő vonalra (IT szintre) kapcsolódó eszközök között pedig a daisy-chain lánc biztosít fix prioritást.

Példaként két konkrét IT rendszert ismertetünk, a Z80 rendszerét és az IBM PC/XT-ben kialakított rendszert.

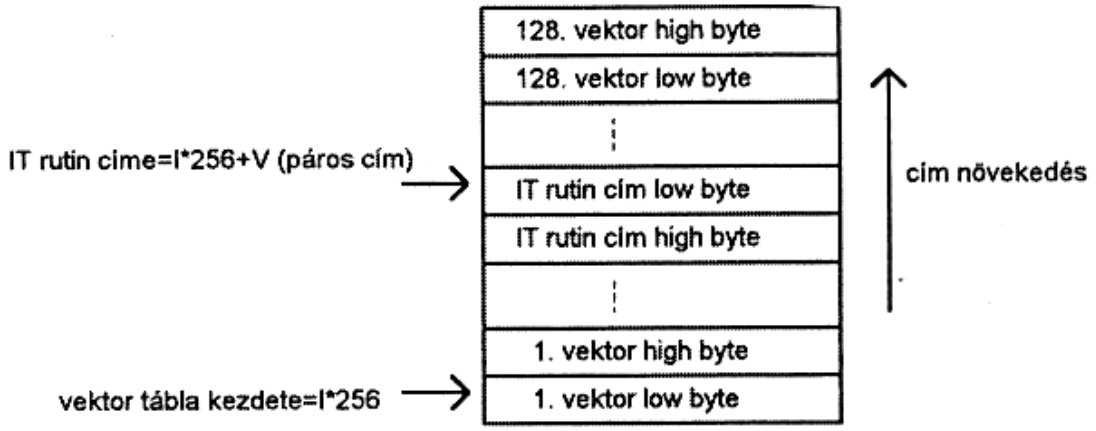
4.6.6. A Z80 interrupt rendszere

A Z80-nak 3 interrupt módja van. Az egyes módok az IM0, IM1, IM2 utasításokkal állíthatók be. A Z80 IT bemenete (mint a legtöbb CPU-nak) szintérzékeny.

Az **IM0** mód az ún. I8080 mód. Ekkor az IT elfogadás után a Z80 az INTA ciklusban, amit az IORQ (periféria ciklus) és M1 (FETCH ciklus) jelek egyszerre történő aktivizálása jelez, egy tetszőleges utasítást vár az adatbuszon (*utasítás beolvasásos IT rutin cím meghatározás*). Ezt az IT vezérlő, vagy a periféria kapuzhatja az adatbuszra, az utasítástól függő számú INTA ciklusok alatt. A 2 byte-os restart, vagy 3 byte-os szubrutin hívó utasítást szokás alkalmazni (CALL <IT rutin cím>).

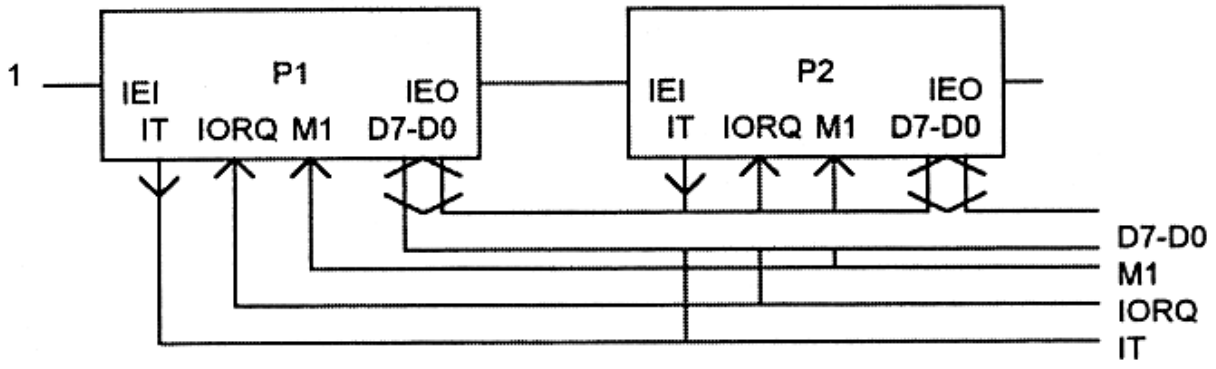
IM1 módban a Z80 az interrupt hatására a 38H címen található interrupt rutint hajtja végre (*fix újraindítási cím*).

Az **IM2** mód a Z80 saját interrupt módja, a Z80-as perifériák erre vannak felkészítve. Itt az INTA ciklus alatt a periféria egy 8 bites vektort (V7-V0) tesz az adatbuszra (*vektor beolvasás*). A vektor páros szám (V0=0). A Z80 I regisztere (amely az LD I,A utasítással tölthető fel) és a beadott vektor együtt egy memóriacímet határoz meg. Az I regiszter tartalma jelöli ki a vektor tábla helyét a memóriában, míg a vektor egy a táblán belüli páros című memória helyet. Itt található az IT rutin kezdőcíme. A processzor a következő utasítás címének stackre mentése után innen veszi elő az IT rutin kezdőcímét, ahogy a 4.47. ábrán látható.



4.47. ábra.

A Z80 perifériák prioritási láncba vannak fűzve (*vektoros daisy chain*). Egy-egy periféria prioritását a láncban elfoglalt helye határozza meg, a lánc elején van a legnagyobb prioritású eszköz. A láncot a minden Z80-as periférián megtalálható IEI és IEO vonalak összekötésével lehet létrehozni, (4.48. ábra).



4.48. ábra. A Z80 perifériák láncba fűzése

Ha egy periféria interruptot kér, akkor az IEO vonalán keresztül letiltja az alatta elhelyezkedő (alacsonyabb prioritási szinten levő) perifériák IT kérését, és meghajtja a CPU IT vonalát. Az interrupt elfogadási ciklus elején, INTA (M1*IORQ) első élére a Z80-as perifériák befagyasztják a prioritási láncot. Egy IT kérő periféria az INTA alatt ráteszi az felprogramozásakor megadott IT vektort az adatbuszra, ha az IEI bemenete engedélyezi (nincs magasabb prioritású kérés). A vektor beolvasása után a prioritási lánc befagyasztását megszüntetik a perifériák. Ezután a már ismertetett módon a megfelelő IT rutinra adja a processzor a vezérlést.

Ha többszintű IT rendszert akarunk létrehozni, a megszakítási rutin elején ki kell adni egy interrupt engedélyező (EI) utasítást. Egyszintű IT rendszer esetén közvetlenül az IT rutinból visszatérő (RETI) utasítás előtt kell kiadni az EI-t, ugyanis ennél az utasításnál a megszakítás kérés észrevételét egy utasításnyit késleltetik, így az EI alatti interrupt kérés csak a RETI után érvényesülhet. A Z80-as perifériák dekódolják a RETI utasítást, s az aktuális megszakítás kérő periféria csak a neki szóló RETI (IEI=1) hatására fogja az IEO kimenetén engedélyezni az alatta levőket (IEO=1). Ezért *nem szabad* az IT rutint RET utasítással befejezni.

Külön érdemes megemlíteni a Z80 esetén az NMI kezelését. A Z80-nál az NMI lefutó él érzékeny. A CPU-n belül az ún. IFF1 flip-flop tárolja azt az információt, hogy az IT engedélyezett-e vagy sem, ezt állítja az EI (enable IT) és DI (disable IT) utasítás, továbbá ezt törli egy IT bekövetkezése. Egy NMI ezt a flip-flopot tiltott állapotba billenti. Ezért, hogy a IFF1 állapota ne vesszen el, az NMI előtt az átmásolódik az IFF2 flip-flopba. Ha ez nem történne meg, akkor egy NMI hatására az esetleg engedélyezett állapot letiltódna, így ezután a megszakítások nem jutnának érvényre. Az NMI rutin (66H címen kezdődik) végére mindenképpen RETN utasítást kell tenni, ugyanis ennek hatására mentődik vissza az IFF2-ből az IFF1-be a régi állapot.

4.6.7. A PC-XT/AT interrupt rendszere (I8086/286)

A 8086/286 processzornál 4 csoportba osztják a megszakítást, melyek prioritása az alábbi felsorolás sorrendjében csökken. Zárójelben a vektor sorszáma áll. A csoportokon belül a kisebb sorszámhoz tartozik a nagyobb prioritás:

Belső hardver okok, mint az
- osztási hiba (0)
- lépésenkénti program végrehajtás (1)
- töréspont (3)
- túlcsordulás (4)

NMI (2)

Szoftver interrupt

Külső hardver interrupt (interrupt vezérlő).

Az interrupt vektor, a Z80-hoz hasonlóan itt is egy vektor táblában jelöl ki egy címet, amelyen az IT rutin kezdőcíme található. A vektortábla fixen a 0-ás címen kezdődik. Egy-egy IT rutinhoz 4 byte tartozik a táblázatban, ezek tartalmazzák az IT rutin

kódszegmensének címét (CS) és a szegmensen belüli címet (PC). A vektor tábla 256 elemű, így 1 kbyte-ot foglal el.

Az első öt (0-4) vektor a belső hardver interruptokhoz és az NMI-hez van rendelve. Az IBM PC-ben a 8H-FH és 70H-77H sorszámú vektorokat foglalták le hardver megszakítások számára.

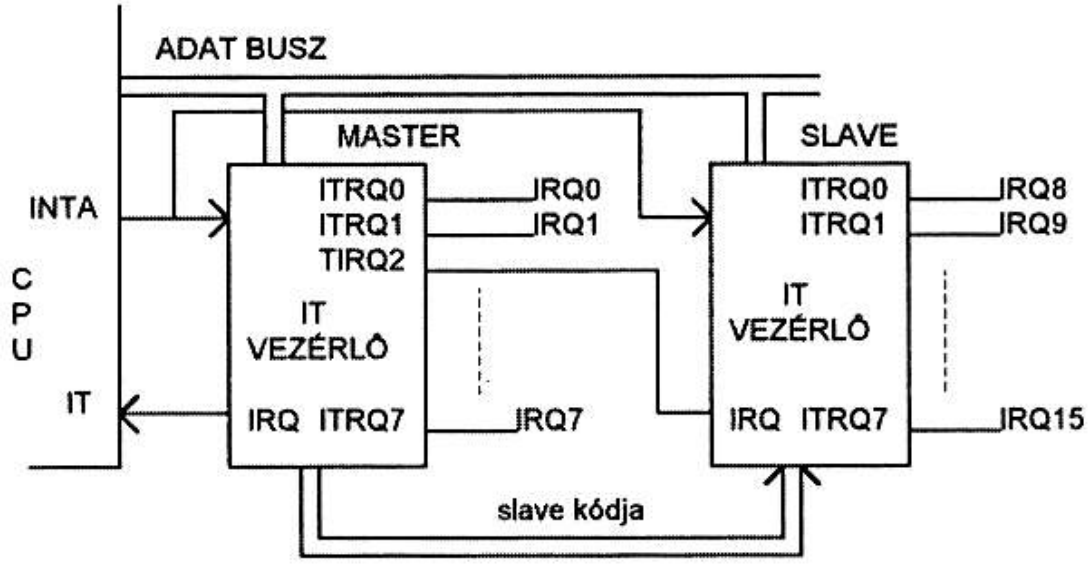
Egy megszakítás elfogadása után, a CPU a stackre menti a flag regisztert, a kód szegmenst (CS) és a következő utasítás kódszegmensen belüli címét (PC). A további interruptok letiltódnak (az IF flag törlődik). Az interruptból visszatérő utasítás (RETI) visszaállítja ugyanezeket a regisztereket, így mivel a flagek is visszaállítódnak (az IT engedélyező bit is), nem kell a visszatérés előtt külön engedélyezni a további IT-ket (STI utasítás).

Az IBM PC-ben centralizál IT rendszert alakítottak ki, dedikált interrupt kérő vonalakkal. (Többszintű, vektoros megszakítási rendszer.)

Az IBM PC-XT rendszer 8 hardver megszakítást képes fogadni. A megszakítás kérő vonalakat (IRQ0..7) egy Intel 8259A interrupt vezérlő IC fogadja.

Az interrupt vezérlőben az interrupt bemenet felfutó él érzékenyre van programozva. Az interrupt vezérlő az IRQi vonalon kapott megszakítás kérést továbbítja a processzor INT vonalára (magas szint), hacsak nincs letiltva a 8259A-ban az adott IRQi vonal fogadása, és nem egy magasabb prioritású IT rutin fut éppen. Ennek hatására a processzor, ha szoftverből engedélyezve van az IT elfogadás, két INTA jelet (interrupt elfogadás) ad ki. Az első INTA jelre a I8259A befagyasztja a pillanatnyi állapotot, hogy csak a magasabb prioritású interruptok érvényesülhessenek. A második INTA hatására pedig egy 8 bites adatot (vektort) továbbít a processzornak. Ennek alsó 3 bitje a 8 IRQ vonal valamelyikét kódolja (az elfogadott kérést), a felső öt bit pedig a felprogramozáskor beállított érték, ami a vektortábla kezdőcímét határozza meg. A DOS operációs rendszer a felső 5 bitet 00001-re állítja, ebből adódik, hogy IBM PC-ben az IRQ0 vonalhoz a 8-as sorszámú vektor tartozik.

A PC-AT alaplapon két 8259A interrupt kontroller található, melyek kaszkádba vannak kapcsolva (4.49. ábra).



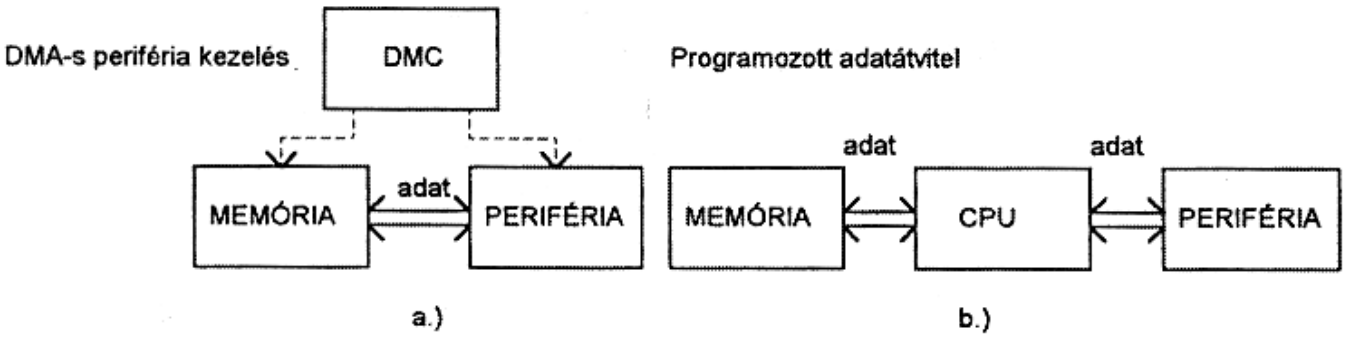
4.49. ábra. I8259A kaszkádosítása

Ez további 8 hardver megszakítás kérő vonal (IRQ8-IRQ15) fogadását teszi lehetővé. A két interrupt vezérlő közül az IRQ0-IRQ7-hez tartozó masternak van programozva, a másik slave-nek. (A 8259A maximálisan 8 interrupt vezérlő kaszkádosítását teszi lehetővé.) A masternak és a slave-nek is meg kell adni egy-egy parancs szóban, hogy a master mely bemeneteire kapcsolódnak a slave-ek. Az AT esetében a kaszkádosításhoz az IRQ2 vonalat használják fel (ide csatlakozik a 2. interrupt vezérlő IT kérő kimenete), ezért az IRQ2 vonal a PC-AT-ben közvetlenül nem használható (az IRQ9 helyettesíti). Kaszkád módban a slave IT vezérlők a masternak jelzik megszakítási igényüket. Ha az aktuális igények közül egy salve-hez tartozó a legnagyobb prioritású, akkor a master továbbítja a kérést a CPU felé, majd 3 kaszkádosító kimenetén kódolva jelzi, hogy melyik slave teheti rá a vektort az adatbuszra.

Az interrupt rutin végén az IT vezérlőket (a mastert és slave-et) értesíteni kell annak befejeződéséről (EOI parancs bit). Ennek hatására veszik vissza az IT kérést a vezérlők.

4.7. A közvetlen memória hozzáférés (DMA)

Az ún. DMA vezérlő segítségével a processzort kikerülve, közvetlen adatátvitel lehetséges a memória és a periféria között (4.50a ábra). Ezt nevezik közvetlen memória hozzáférésnek (Direct Memory Access). Ennek egyrészt az az előnye, hogy mivel az adatátvitelt egy speciálisan erre a célra kialakított hardver végzi, az többnyire gyorsabb, mintha a CPU végezné (nem kell közben a memóriából utasításokat olvasni és nem kell az adatot a CPU-n keresztül áramoltatni, (4.50b ábra). Másrészt, a DMA-s kezelés tehermentesíti a processzort (egyszerűbb lesz program) A DMA-s kezelést sokszor nagy sebességű, blokkos adatátvitelt igénylő perifériáknál alkalmazzák (floppy disk, winchester).



4.50. ábra. Az adat útja DMA-s és programozott kezelés esetén

Az átvitel lebonyolítását az ún. DMA vezérlő (DMC) végzi. Ez egy speciális periféria, amely felprogramozása után busz master funkciót képes ellátni. A DMA vezérlő főként a periféria és a memória közötti átvitel CPU-nál gyorsabb elvégzésére készített speciális hardver elem.

A DMA-s átvitel lebonyolítása a következő fázisokból áll: felprogramozás, átvitel kezdeményezés, busz vezérlési jogának átvétele, adatátvitel lebonyolítása, busz vezérlési jogának visszaadása, átvitel végének jelzése. Ezeket részletezzük a következőkben.

Felprogramozás

Az átvitel lebonyolításához a DMA vezérlővel közölni kell a következőket:
- az átvitel **irányát** (periféria → memória, memória → periféria, és lehetőség van memória-memória átvitelre is),
- az átviendő adatblokk memóriabeli **kezdőcímét**
- az átviendő blokk méretét (**szószám**)
- az átvitel **módját** (lásd később)

A DMC a felprogramozási fázis alatt szokványos perifériaként viselkedik, s a fenti információk a megfelelő regisztereibe kiírt adatként kerülnek bele.

Az átvitel kezdeményezése

- A periféria kezdeményezi, a DMC megfelelő kérő vonalán keresztül, vagy
- a program kezdeményezi a DMC-nek adott parancsszóval.

Az utóbbi főként memória→ memória átvitel esetén szokásos, ill. olyan perifériáknál, ahol egy kérés hatására megtörténhet a teljes blokk átvitele.

CPU-DMC arbitráció

A felprogramozás után, ha DMA átvitel kezdeményeződik, a DMA vezérlő az arbitrációs ciklus folyamán átveszi a busz vezérlési jogát a CPU-tól (busz masterré válik). A busz elkérésére a processzorok két jele szolgál.

DRQ Ezen a bemeneten jelzi a buszt elkérő egység, hogy használni kívánja a buszt.

DACK Ezen a kimeneten jelzi a CPU, hogy lekapcsolódott a buszról.

A busz elkérésére a processzorok az aktuális buszciklus befejezése után reagálnak. Ilyen módon akár egy utasítás végrehajtást is meg lehet szakítani a busz elkérésével. A processzorok egy része a buszhoz fordulást nem igénylő (belső) műveleteket ilyenkor is folytatni képes.

Adatátvitel

Az DMA-s átvitel során a DMA vezérlő megfelelő buszciklusok generálásával átviszi az adatforrásból az adatokat cél egységbe. A buszciklusok hossza általában bizonyos határok között programozható, ill. lehetőség van külső jellel annak meghosszabbítására (a wait kéréshez hasonlóan).

DMC-CPU arbitráció

Az átvitel befejezése után a DMC visszaadja a busz vezérlésének jogát a CPU-nak.

Az adatátviteli módtól (később részletezzük) függően vagy a teljes adatblokk átvitele alatt a DMC-nél van a busz vezérlésének joga, vagy időnként visszaadja. Ezért a DMC felprogramozásakor megadott számú adat átvitele során esetleg többször ismétlődik a következő három fázis: CPU-DMC arbitráció, adatátvitel, DMC-CPU arbitráció.

Az átvitel befejeződésének jelzése

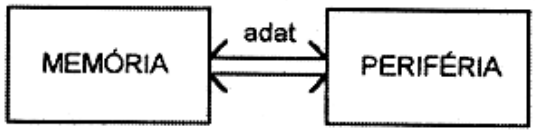
A teljes adatblokk átvitelének végét a DMA vezérlő vagy a periféria jelzi többnyire interrupttal.

Az átvitel befejezése után a program a bejövő adatokat feldolgozza ill. előkészíti a következő kimenő adat blokkot.

Attól függően, hogy egy szó átvitelét hány buszciklussal bonyolítja le a vezérlő, megkülönböztetünk egyciklusú és a kétciklusú átvitelt.

Egyciklusú DMA átvitel

Egyciklusú átvitelnél közvetlenül kapcsolódik össze a periféria és a memória (4.51. ábra) az adatbuszon keresztül, s egy byte átvitele egy ciklus alatt megtörténik. Ilyen átvitelre alkalmas az I8237 típusú DMC. (Az IBM PC DMA vezérlője is ilyen kialakítású.)



4.51. ábra. Adat áramlás egyciklusú DMA átvitel esetén

A 4.52. ábra egy egyciklusú memória-periféria átvitelre képes DMA vezérlő és a periféria kapcsolatát mutatja. A vastagon húzott jelek periféria → memória irányú átvitel esetén aktivizálódnak.

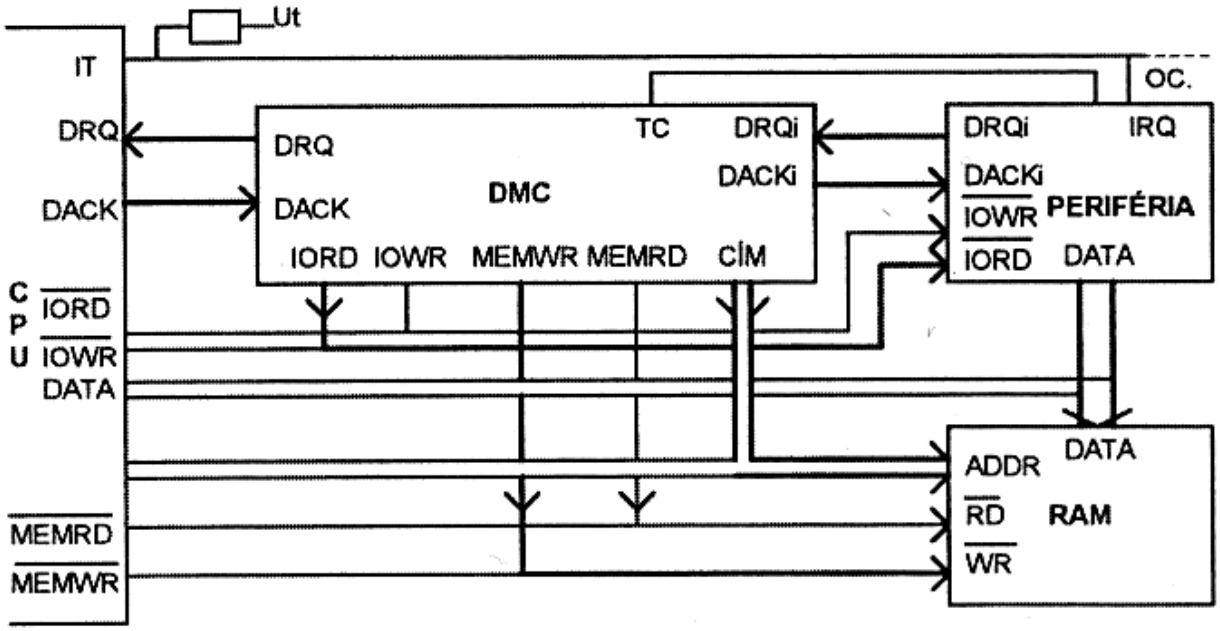
A DMA átvitel három részre bontható:

- a DMA vezérlő elkéri és megkapja a buszt (1. busz arbitrációs szakasz)
- elvégzi az átvitelt (adatátviteli szakasz, cím és vezérlőjelek kiadása)
- visszaadja a buszt (2. busz arbitrációs szakasz)

Az átvitel lefolyása részletesen:

- a periféria jelzi a DMC-nek, az átvitel igényét, a DRQi vonalon (egy vezérlő több DMA csatornával is rendelkezik, s mindegyik csatornához van egy DRQi-DACKi jelpár),

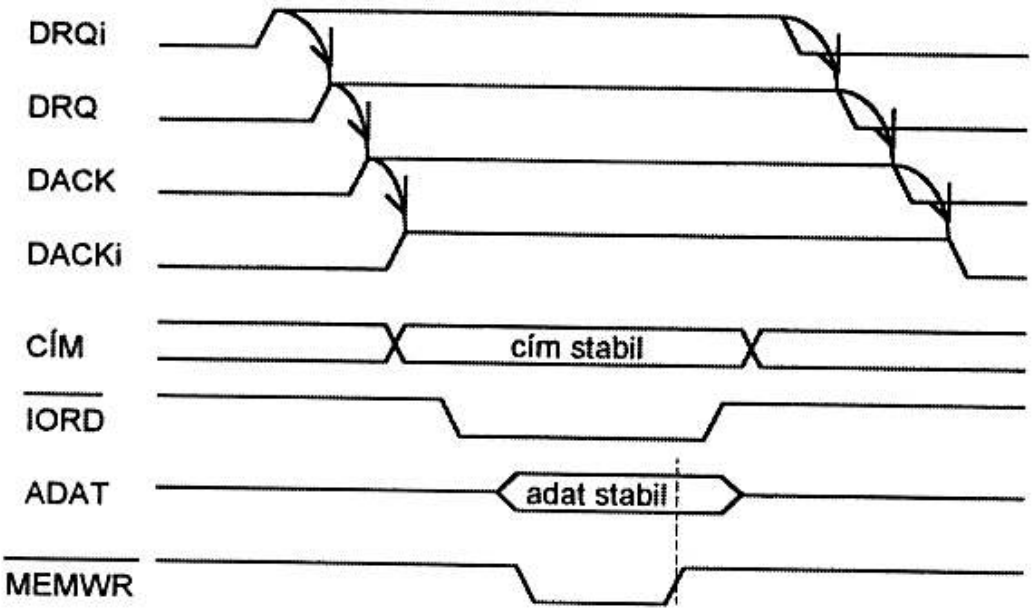
- a DMC a DRQ jel aktivizálásával elkéri a CPU-tól a buszt,



4.52. ábra. Egyciklusú DMA blokkvázlata

- a CPU befejezi az aktuális buszciklust, lekapcsolódik a buszról s ezt visszajelzi a DACK kimenetén,
- a DMC visszajelzi a DACKi vonalon a perifériának, hogy megkezdődhet az átvitel, s közben kiadja az aktuális memória címet, továbbá periféria → memória átvitel esetén aktivizálja az IORD, memória → periféria átvitel esetén pedig a MEMRD jelet,
- periféria → memória átvitel esetén a DACKi*IORD hatására a periféria az adatbuszra kapuzza az adatot, s annak stabilizálódása után a DMC a MEMWR jellel beírja azt a memóriába,
- memória → periféria átvitel esetén a megcímzett memóriarekesz tartalma a MEMRD jel hatására az adatbuszra kerül, melynek stabilizálódása után az DMC az IOWR jellel beírja azt a periféria regiszterébe,
- ha a teljes blokk átvitele megtörtént, azt többnyire egy interrupttal jelzi a DMC a processzornak (ezt a DMC blokkvéget jelző TC kimenete és a DACKi együttesen válthatja ki), és visszaveszi a DRQ jelet, (üzemmódtól függően a DRQ a teljes blokk átvitele előtt többször megszűnhet ill. aktivizálódhat),
- a processzor a DRQ megszűnését érzékelve visszaveszi a DACK jelet és folytatja működését a következő gépi ciklussal.
- a DMA folyamat a teljes blokk átvitele előtt megszakítható, TC vonal aktivizálásával, ugyanis ez kétirányú (belül nyitott kollektorosan meghajtott és figyelt jel)

A 4.53. ábra egy periféria → memória irányú egyciklusú DMA átvitel vázlatos idődiagramját mutatja.



4.53. ábra. Egyciklusú DMA idődiagramja (periféria→ memória)

A DMA vezérlők többféle átviteli üzemmódra képesek. Az átviteli módok a következők lehetnek:

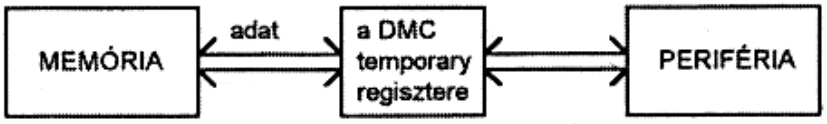
- *egyenkénti* (single), amikor minden egyes szó átviteléhez külön DMA kérés szükséges, folyamatos DMA kérés esetén minden DMA ciklust egy processzor ciklus követ,
- *blokkos*, amikor egyetlen DMA kérés hatására az egész blokk átvitele megtörténik,
- *demand*, amikor a DMA átvitel addig tart, amíg a kérés fennáll, egy teljes blokk átvitele esetleg csak több kérés hatására történik meg.

Az egyciklusú DMA esetén megfigyelhető, hogy a buszon egyszerre van memória cím, miközben a IORD vagy IOWR aktív. A perifériák a címbuszból csak kevesebb számú címbitet figyelnek, mint a memóriák (8-10-et). Ha egy közönséges (DMA-t nem használó) periféria címe megegyezik a címbuszon levő memória címmel a periféria által figyelt címbitekben, akkor ez a periféria azt hiszi, hogy megcímezték, és hibásan adat íródik bele, vagy meghajtja a buszt, a DMA átvitel irányától függően. Ennek elkerülésére a nem DMA-s perifériák címdekóderét le kell tiltani az átvitel alatt (pl. a DACK jellel, ezt a célt szolgálja az IBM PC ISA busz AEN jele is).

Kétciklusú DMA átvitel

Ebben az esetben a DMA vezérlő az első ciklusban megcímzi a forrást és egy belső ideiglenes (temporary) regiszterében tárolja az onnan kiolvasott adatot (4.54. ábra). A második ciklusban megcímzi a célt és a temporary regiszter tartalmát beírja cél tárolójába. A forrás ill. a cél lehet memória és periféria. Egy ilyen temporary regisztert

tartalmazó DMC-vel memória-memória átvitel is lehetséges (pl. gyors blokkmozgatás a memóriában). A kialakítás előnye, hogy a DMA-s perifériák kiválasztó áramkörének felépítése azonos a normál perifériákéval. Hátránya viszont, hogy kb. fele olyan gyors, mint az egyciklusú átvitel.



4.54. ábra. Adat áramlás kétciklusú DMA átvitel esetén

Pl. a Z80 saját DMA vezérlője ilyen kialakítású. Ennél a DMC-nél csak egy közös READY jel tart kapcsolatot a periféria és DMC között, melyen a periféria az átviteli készségét jelzi. Itt a perifériát a memóriához hasonlóan, a címbuszon keresztül éri el a DMC, szemben az egyciklusú átvitellel, ahol *a címzést a DACKi jel helyettesíti*.

4.8. Periféria kezelési módszerek

A perifériák egy része a működésének engedélyezése után autonóm módon végzi a feladatát s a megfelelő időpontokban kész jelzést ad (pl. az adat kiolvasható vagy újabb adat küldhető). Az ilyen perifériákat nem kell minden egyes kész jelzés után újraindítani, az folyamatosan termeli vagy igényli az újabb adatokat. Ezeket a perifériákat **aktív perifériák**nak nevezzük.

Más perifériák minden egyes működés elkezdéséhez külön indítást igényelnek (pl. bizonyos analóg-digitális (A/D) konverterek). Az ilyen perifériákat **indítást igénylő passzív perifériák**nak nevezzük.

A perifériák egy harmadik csoportja nem igényel indítást és nem végez autonóm működést továbbá nem ad visszajelzést ezeket **indítást nem igénylő passzív perifériák**nak nevezzük. (Pl. egy egyszerű kimeneti regiszter.)

A periféria és a processzor sebességkülönbsége miatt sok esetben a CPU és a periféria szinkronizálására van szükség. Túl gyors periféria esetén DMA-val lehet biztosítani az adatáramlást a periféria és memória között.

A CPU és periféria szinkronizálásának legalsó szintje a buszciklus szinkronizálása. Ezt az aszinkron és szinkron busznál már elemeztük.

A buszciklusnál magasabb szintű szinkronizálás megoldási módjai szerint különféle periféria kezelési módszereket különböztetünk meg. Ezeket mutatja 4.55. ábra. Ezek közül a DMA-s kezelést már részleteztük ezért a következőkben ez nem szerepel.

4.55. ábra. Periféria kezelési módszerek csoportosítása

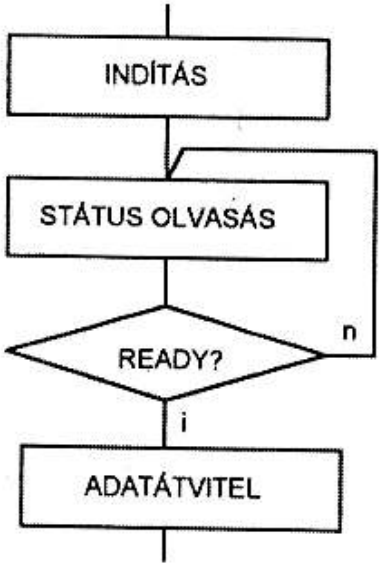
4.8.1 Programozott periféria kezelés

Közvetlen szoftver ütemezés

Egyes perifériáknál nincs szükség szinkronizálásra mert vagy nincs mihez szinkronizálódni (pl: egy kijelzőt meghajtó regiszter, egyszerű bementi port), vagy a periféria egy művelet elindítása után a következő utasítás végrehajtására már biztosan elkészül. Az ilyen perifériákhoz a processzor tetszőleges időpontban fordulhat.

Lekérdezéses ütemezés

Ha a periféria sebessége olyan, hogy egy perifériás művelet néhány utasítás végrehajtási időn belül fejeződik be, akkor a szinkronizálást státus figyeléssel célszerű megoldani. Ilyenkor a program, a periféria adatregiszteréhez fordulás előtt ciklikusan addig olvassa a státusz regisztert, amíg az azt nem jelzi, hogy kész a következő műveletre (olvasásra vagy írásra). Csak ezután történik meg az adatátvitel. Ennek folyamatábráját mutatja a 4.56. ábra indítást igénylő passzív periféria esetében.



4.56. ábra. Lekérdezéses ütemezés folyamatábrája

Ez a fajta kezelés lassú periféria esetében sok várakozással jár, ami alatt a CPU nem végez érdemleges működést. Ezért csak akkor engedhető meg, ha CPU egyéb feladatait így is képes elvégezni. Az igényelt hardver tekintetében általában a lekérdezéses ütemezés a legolcsóbb megoldás.

4.8.2 Program megszakításos periféria kezelés

Gyakran a periféria csak viszonylag hosszú idő vagy előre kiszámíthatatlan idő múlva lesz kész az újabb adatátvitelre. Ha ilyen esetben lekérdezéses ütemezést alkalmaznánk, akkor nagyon leromlana a processzor hatásfoka, hiszen az a státus figyelés időtartama alatt esetleg több száz egyéb utasítást is végrehajthatott volna. Ezért ilyenkor interruptos periféria kezelést célszerű alkalmazni.

A megszakításos kezelésnél a perifériához fordulást maga a periféria kezdeményezi egy logikai jellel a CPU vagy IT vezérlő interrupt kérő bemenetén. Az interruptot hardvertől függően az IT vonal szintje (szintérzékeny bemenetű hardver), vagy a jel valamely éle (élérzékeny bemenetű hardver) váltja ki. Esetleg a kettő együtt, vagyis valamelyik él és a szint kiszolgálásig történő fenntartása szükséges.

Az IT kezdeményezés hatására valamilyen mechanizmussal előbb-utóbb egy speciális szubrutinra (interrupt rutin) adódik a vezérlés, ami elvégzi a periféria kezelését, kiküldi ill. beolvassa az adatokat. Ez a programozottnál többnyire sokkal jobb CPU idő kihasználást eredményez. Mivel a CPU és a periféria szinkronizálása automatikus, nem kell feleslegesen státusz figyeléssel tölteni az időt.

Az interruptos kezelés a program áttekinthetőségét is növeli. Ha a CPU jó kihasználása mellett egy státus figyelést igénylő lassú perifériát is ki szeretnénk szolgálni az interrupt elkerülésével, akkor a program fő ciklusába be kell iktatni egy vagy több státus lekérdezést és feltételes szubrutin hívást. Ha a periféria kiszolgálása annak jelzése után nem tűr halasztást, akkor sűrűn be kell iktatni a státus figyelést. Ez egyrészt megnöveli a program méretét, másrészt az nehezen áttekinthetővé válik és a reagálása is lassabb lesz, mint az IT-s megoldásnak.
Mivel az IT rutinban nem a periféria kezelését szolgáló, ugyanakkor mindig végrehajtandó feladatok is vannak (regiszter mentés, regiszter visszaállítás, stack kezelés), túl sűrűn érkező IT-k szintén leronthatják a CPU kihasználtságát.

A fentieket figyelembe véve az IT-s periféria kezelést akkor célszerű alkalmazni, ha a CPU sebessége elegendő a periféria kezelésére és
- az interruptok várhatóan nem túl sűrűn jönnek, közöttük a CPU elég sok utasítást képes végrehajtani, vagy
- ha az interrupt bekövetkezésének időpontja véletlenszerű, de ha bekövetkezik, akkor gyorsan ki kell szolgálni,
- az interrupt alatt elvégzendő feladatok nem vesznek túl sok időt igénybe, vagy ha igen, akkor azt egyéb IT is megszakíthatja (többszintű IT rendszer) .
- az interruptos szervezés a program strukturáltságát, áttekinthetőségét szolgálja.

4.9. Busz illesztések

A mikroprocesszoros busz illesztések célja, hogy megoldjuk a buszra kapcsolandó egység felületének a busz jeleihez kapcsolását, ami az alábbiakat jelenti.

a. A buszjeleknek megfelelő logikai funkciójú és szintű (alacsony ill. magas aktív) jeleket (be- és kimenő jelek) kell kialakítani az illesztendő egység buszra kapcsolódó felületén.

b. A megfelelő funkciójú jeleknek időzítésben illeszkedniük kell a busz előírásaihoz.

c. A jeleknek terhelés ill. meghajtóképesség szempontjából illeszkedniük kell a busz előírásaihoz.

4.9.1 Memória egység felépítése és illesztése

A mikroprocesszoros rendszer programja a memóriában helyezkedik el. Minden mikroprocesszoros rendszerben kell egy olyan ROM memória, amely legalább a bekapcsolás után elvégzendő feladatok programját tárolja. A vezérlési célú μP-s rendszerek többségében a teljes program itt tárolódik. Ez a memória többnyire EPROM vagy EEPROM.

Szintén elengedhetetlen, hogy a mikroprocesszoros rendszer RAM memóriát is tartalmazzon, mert a processzorok többségének (egyes mikrokontrollerek kivételével) itt helyezkedik el a stackje, és többnyire a belső regiszterek nem elegendők a változók tárolására. Ez a memória elhelyezkedhet a processzor chipben (mikrokontrollerek esetén), vagy a processzor külső buszára illesztve. Egyes rendszereknél a RAM-ból is futhat program. Ilyenkor azt valamilyen háttértárolóból tölti be a processzor.

A memória illesztés esetén első feladat az illesztendő memória méretének meghatározása. A külső illesztendő ROM memória méretének meghatározásánál figyelembe kell venni a megoldandó feladat bonyolultságát és a választott programozási nyelvet. Assembly nyelven kisebb kódmérettel valósítható meg ugyanaz feladat, mint a magas szintű nyelvek esetében. A magas szintű nyelvek ezen a hátránya az egyre jobb kód optimalizálásnak köszönhetően csökkenő tendenciát mutat. Viszont a fejlesztési idő - bonyolult feladat esetén - sokkal hosszadalmasabb az assembly nyelv esetében.

A RAM méretnél, ha azt csak adattárolásra használjuk de programot nem futtatunk belőle, elsősorban az implementálandó algoritmus adatterület igénye a meghatározó, de figyelembe kell venni a szükséges stack területet is, amely szintén függ a programozási nyelvtől és a programozási stílustól. A stack terület meghatározásánál fontos figyelembe venni, hogy milyen mélységű szubrutin beágyazások fordulnak elő. Ebbe bele kell számolni az interrupt rutinokat is, melyek tovább mélyíthetik a stacket, főként többszintű interrupt rendszer esetében.

Természetesen egy felső korlátot ad a választott CPU címtartománya. Ez a korlát, ha nagyon szükséges elkerülhető, egyszerű lapozásos rendszerű memória illesztéssel (lásd később), de ha lehet, ezt inkább ne alkalmazzuk, mert elbonyolítja a memória kezelését. Inkább válasszunk nagyobb címtartományú CPU-t. Általánosan mondható, hogy a

memóriával nem érdemes túlságosan spórolni, mert a szoftver továbbfejlesztésének korlátja lehet.

A memória típust a RAM ill. ROM méret és a szükséges hozzáférési idő függvényében kell kiválasztanunk. A hozzáférési időre elsősorban a választott CPU és az alkalmazott órajel frekvencia alapján adódik egy felső korlát, persze figyelembe kell venni az illesztésnél alkalmazott áramkörök késleltetését is. Természetesen az ár is meghatározó szempont, a nagyobb ill. gyorsabb memóriák drágábbak.

A EPROM-ok hozzáférési ideje nagyobb a RAM-okénál. Ha az alkalmazható ROM olyan lassú, hogy az adott rendszerben a CPU nem lenne képes közvetlenül olvasni, (az olvasási ciklusban előírt időre az adat még nem stabilizálódik) akkor ezen ciklus hosszabbítással (WAIT, DTACK) lehet segíteni. Természetesen a RAM-oknál szintén.

Ha a program végrehajtása a ROM-hoz fordulási ciklus meghosszabbítása miatt elfogadhatatlanul lelassul, akkor ezen az ún. árnyék memória (shadow RAM) alkalmazásával lehet segíteni. Ez azt jelenti, hogy a ROM-ból át kell tölteni a programot egy olyan RAM-ba, amelyből az gyorsabban végrehajtható.

Ha a szükséges memória méret nem valósítható meg egy IC-vel, akkor azt több kisebb egységből rakjuk össze.

Változó szószélességű memória hozzáférés

Mint már említettük, egyes processzoroknál utasítástól függően változhat az adatbuszon átvitt adat szószélessége. Pl. I8086 esetén az ún. $\overline{BHE}$ jel és az A0 különféle kombinációi jelölik ki a byte-os vagy szavas hozzáférést.

Itt az utasítástól ill. az adat elhelyezkedésétől függően 5 lehetőség van, ahogy a 4.2. táblázat mutatja.

$\overline{BHE}$	A0	
1	0	byte páros címen
0	1	byte páratlan címen
0	0	szó páros címen
0	1	szó 1. byte-ja páratlan címen
1	0	szó 2. byte-ja páros címen

4.2. táblázat. Változó szószélességű hozzáférési lehetőségek

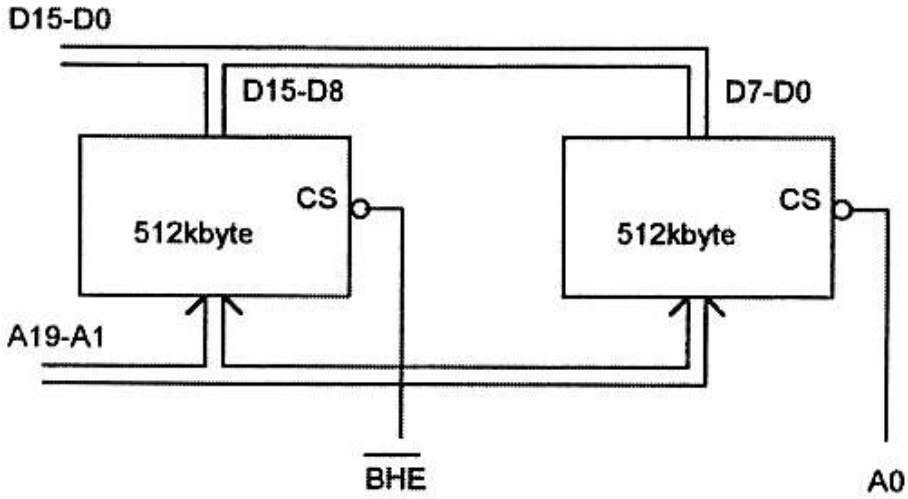
Egy ilyen módon hozzáférhető memória illesztésének elvét mutatja a 4.57. ábra, az I8086 esetére.

Ha az utasítás végrehajtása szavas hozzáférést igényel, akkor két lehetőség van.

- Ha a szó páros memória címen kezdődik, akkor egy buszciklussal hozzáférhető a teljes 16 bit.

- Ha a szó páratlan címen kezdődik, akkor csak 2 buszciklussal érhető el a 16 bit.

Ha az utasítás byte-os hozzáférést igényel, akkor ez a buszon ugyanolyan, mint a páratlan címen kezdődő szó páros ill. páratlan byte-jához való hozzáférés.



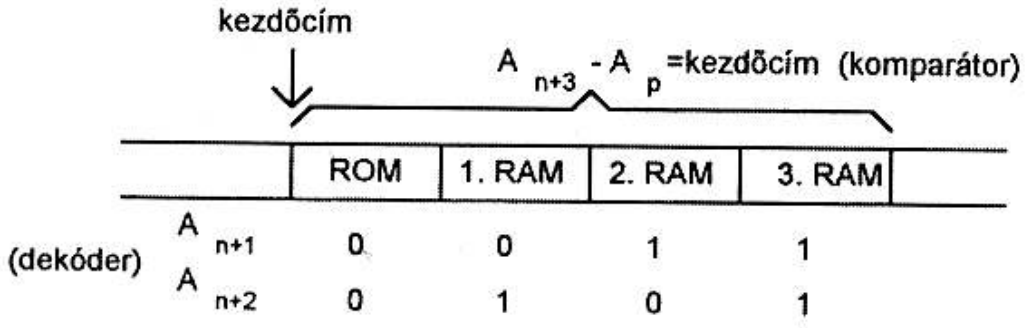
4.57. ábra. Változó szószélességű adat hozzáféréses memória illesztés elve

Egyszerű memória illesztés

A következő példában (4.59. ábra) egy szinkron buszra illesztett 4 chipből készített memória illesztést mutatunk be. A kapcsolásban a komparátor a teljes címtartományból egy kisebb egységet jelöl ki, ennek a tartománynak a mérete pontosan az illesztendő memóriának megfelelő, a komparátorra csatlakozó címvonalak száma szabja meg. Ha pl. a processzor címtartománya 64 kbyte-os (A15-A0), az illesztendő memória pedig 16 kbyte-os (A13-A0), akkor a komparátorra az A15-A14 címbiteket kell kapcsolni. E két címbit különféle kombinációi a teljes 64 kbyte-os tartomány egy-egy 16 kbyte-os részét jelölik ki. A komparátor másik oldalán kapcsolókkal beállított kombinációval ezek közül lehet kiválasztani egyet, s ez határozza meg az illesztett memória kezdőcímét.

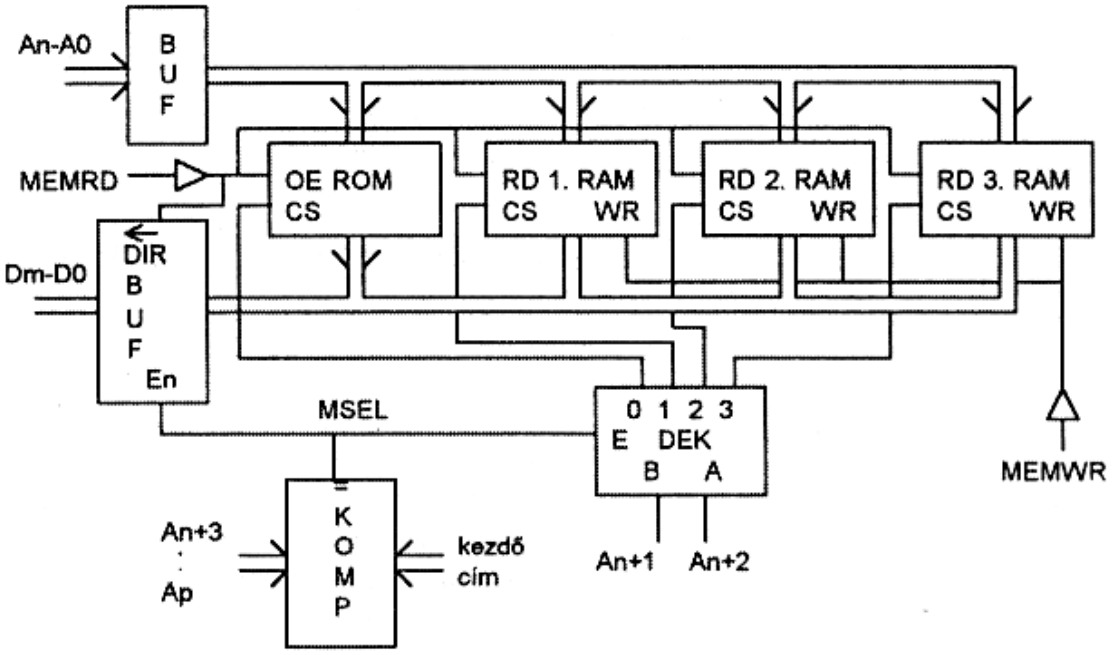
Ha memóriát egyforma méretű (pl. 4 kbyte) chipekből tervezzük, akkor tartományt (pl. 16 kbyte) elosztva a chipek méretével megkapjuk, hogy hány darab szükséges, ill., hogy hány kimenetű dekóder kell. A dekóder egy-egy kimenete egy chip mértének megfelelő tartomány jelöl ki (4 chip valamelyikét), az illesztendő tartományon belül. A chip-en belüli címet az A0-An határozza meg.

A memória kiosztást az ún. memória térképen szokás ábrázolni (4.58. ábra).



4.58. ábra. Memória térkép

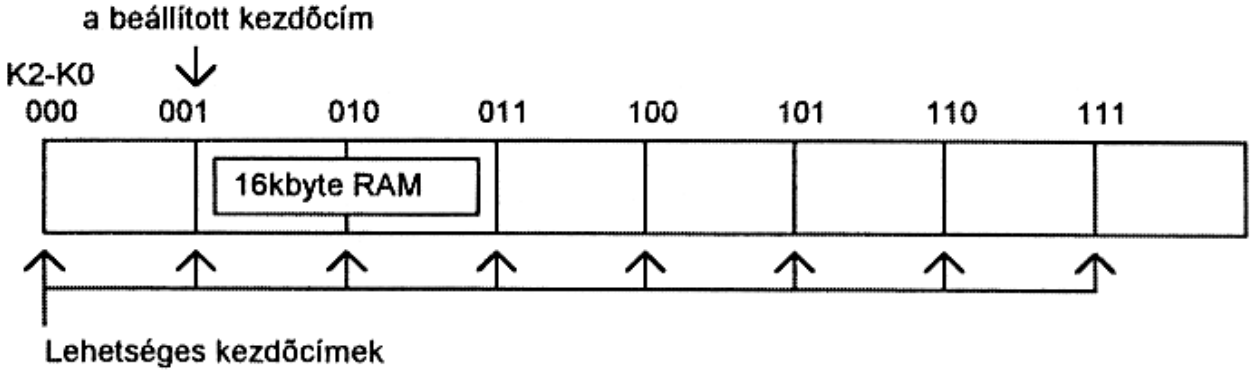
Az illesztéseknél figyelembe kell venni a CPU cím, adat és vezérlő vonalainak terhelhetőségét is. Amennyiben a közvetlen illesztés azt túlterhelné, akkor meghajtókra van szükség. A bemutatott illesztésben ezek is szerepelnek. Az adatbusz meghajtója kétirányú, ha RAM is van az illesztendő chipek között. Az irány akkor fordul a CPU felé, ha az olvasni akar a memóriából (MEMRD). A MEMWR jelet az adatbusz forgatására nem szabad használni, mert egy írási ciklusban, amikor a címet már kiadta a CPU, esetleg az adatot is kiadhatja, mielőtt a MEMWR jelet aktivizálná, s ez a CPU és a kétirányú meghajtó rövid idejű szembekapcsolódását eredményezheti! Az adatbusz meghajtót akkor lehet engedélyezni, ha az illesztett memória van kiválasztva, ezt jelzi az MSEL jel.



4.59. ábra. Memória illesztés blokkvázlata

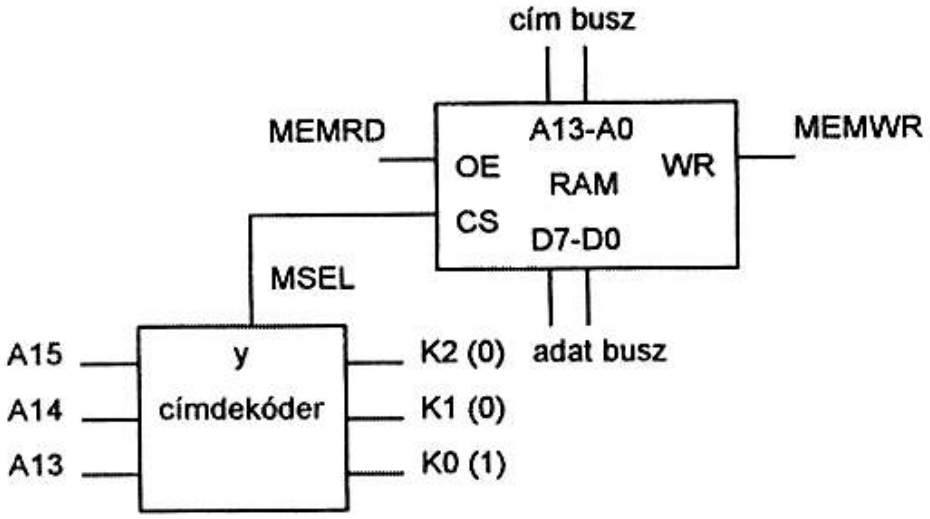
Ha egy memória kezdőcímét annak méreténél kisebb lépésekben is megadhatónak akarjuk, akkor a lépések finomságától függően az alacsonyabb címbitek figyelésére is szükség van a memória engedélyező jelének kialakításánál.

Ilyen esetben komparátor helyett PROM-ot vagy PAL-t célszerű alkalmazni. A 4.60. ábra egy olyan memóriatérképet mutat, ahol a 64 kbyte-os címezhető tartományba illesztett 16 kbyte-os RAM kezdőcíme 8 kbyte-os lépésekben állítható a K2-K0 jelekkel.



4.60. ábra. Memória térkép kis lépésekben állítható a kezdőcímű memóriához

A 4.61. ábra. mutatja az illesztés kialakítását, a 4.3. táblázat pedig a memória címdekóderének igazságtáblázatát. A kialakítás olyan, hogy a legnagyobb kezdőcím beállítása esetén (K2-K0=111), a memóriából csak 8 kbyte látszik. Több címbitet figyelembe véve természetesen lehetőség van finomabb kezdőcím beállítás kialakítására is.



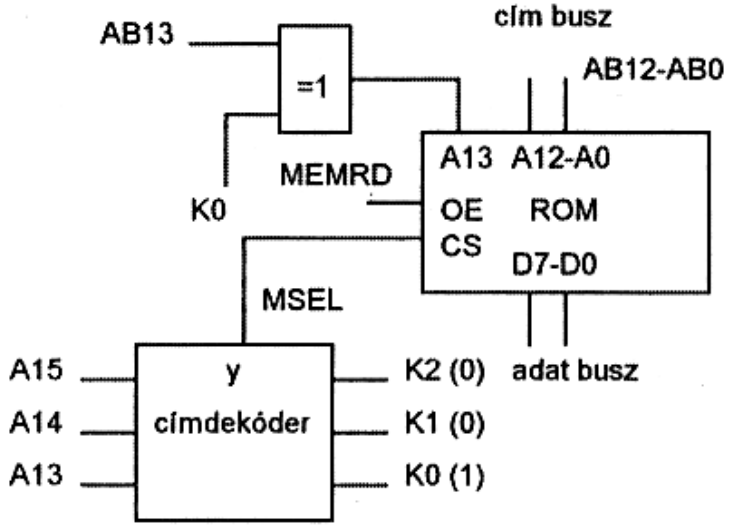
4.61. ábra. Kis lépésekben állítható kezdőcímű RAM illesztés

Felhívjuk a figyelmet, hogy különböző kezdőcím beállítások esetén, a memória tárolói eltérő helyen látszanak. Ennek az az oka, hogy a memória címdekódere olyan címbitet is figyel, amely a memóriához is csatlakozik. A konkrét esetben ha a K2-K0 páratlan, akkor a memória alsó és felső fele felcserélt sorrendben látszik a páros esethez képest, az A13 címbit miatt. RAM-ok esetén, ha a kezdőcímet fixen állítjuk be, akkor ez nem okoz problémát, hiszen egy adott címre írt adatot ugyanarról a címről olvassuk ki, s nem érdekes, hogy azt a RAM melyik konkrét regisztere tárolja. ROM esetében azonban ez problémát okozna, mert az előre beégettet program ill. adatok páratlan K2-K0 beállítás esetén a memóriában hibás címen jelennének meg.

A15-13	K2-0							
	000	001	010	011	100	101	110	111
000	1	0	0	0	0	0	0	0
001	1	1	0	0	0	0	0	0
010	0	1	1	0	0	0	0	0
011	0	0	1	1	0	0	0	0
100	0	0	0	1	1	0	0	0
101	0	0	0	0	1	1	0	0
110	0	0	0	0	0	1	1	0
111	0	0	0	0	0	0	1	1

4.3. táblázat. Címdekóder igazságtáblázata

A jelenség megszüntetéséhez ún. címtranszformációt kell alkalmazni. Címtranszformáció során a memória cím bemenetire egy kombinációs hálózaton (cím transzformátor) keresztül jut el a cím. Ennek hatására a címtranszformátor bemenetei felől nézve a memória bizonyos tartományai máshol helyezkednek el, mint normál esetben. A példában páratlan K2-K0 esetén meg kell invertálni a memóriára csatlakozó A13 címbitet. Ennek megvalósítását mutatja a 4.62. ábra.



4.62. ábra. Címtranszformáció

Egyszerű lapozásos memória illesztése (tömbkapcsolás)

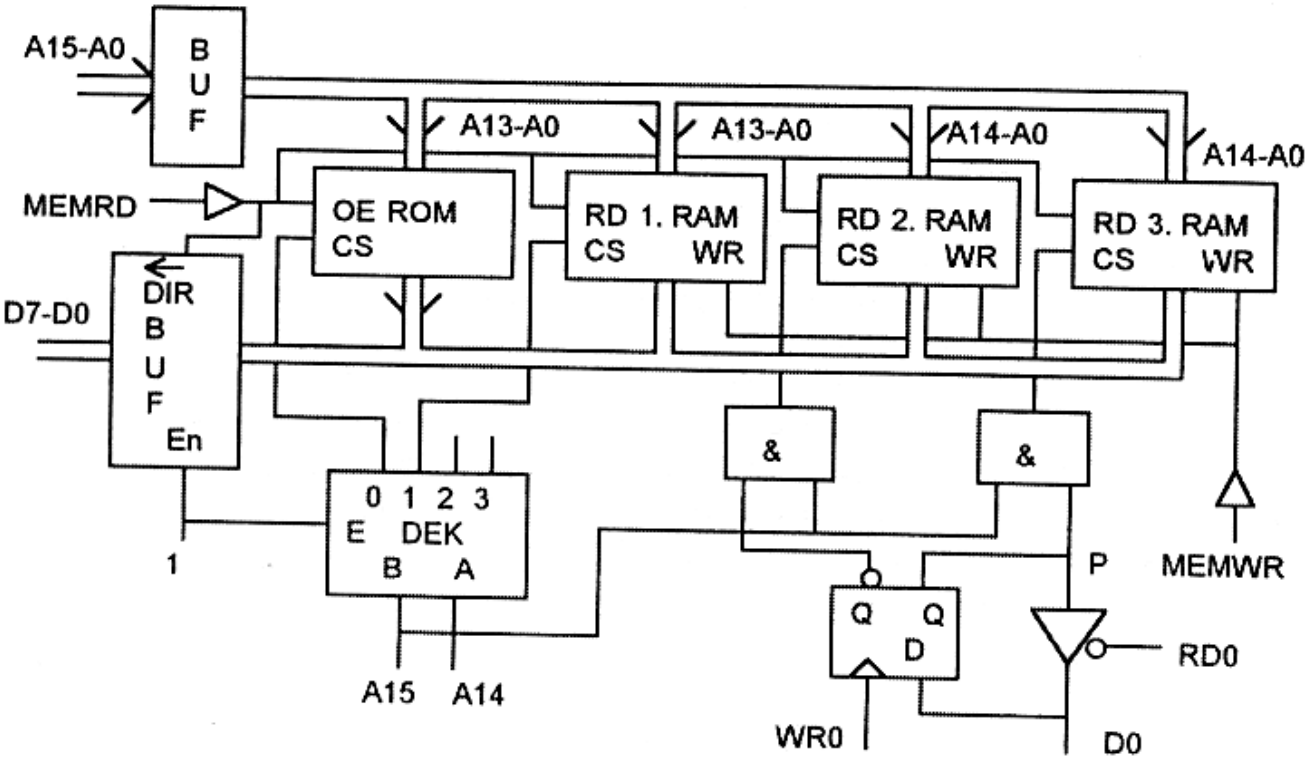
Ha a CPU címtartománya nem elegendő a szükséges memóriaméret kezelésére, akkor ún. egyszerű lapozásos memóriával (a szakirodalom *tömbkapcsolás*nak is nevezi) kerülhetjük el a problémát. (Ezt csak akkor alkalmazzuk, ha valamely okból nincs lehetőség megfelelő címezhető tartományú CPU-t választani).

A tömbkapcsolású memóriánál az igényelt memória területet olyan méretű részekre (lapokra) osztjuk, amelyek belefférnek a processzor címtartományába. Egyszerre mindig csak egy memória laphoz lehet hozzáférni, hogy éppen melyikhez, azt az ún. lapregiszterbe írt adat határozza meg. A lapregiszter többnyire az I/O tartományban helyezkedik el, és célszerűen visszaolvasható a tartalma.

A megvalósított egység memória térképét mutatja 4.63. ábra, blokkvázlatát a 4.64. ábra. A memóriát egy 64 kbyte-os címezhető tartományú (A15-A0) processzoros buszra illesztettük. Az alsó 32 kbyte-on levő ROM és RAM illesztése a megszokott. A felső 32 kbyte-on található a lapozható memória, melynek 2 lapja van. Az aktuális lapot a P flip-flop jelöli ki. Az ezt beíró WR0 és az állapotát visszaolvasó RD0 jelet előállító periféria dekódert nem tüntettük fel a rajzon. A felső 32 kbyte-ot A15=1 jelöli ki. Itt látszik a lapozható memória 0. lapja ha P=0, az 1. lapja, ha P=1.

	16kbyte ROM	16kbyte RAM	32 kbyte RAM	0. lap	P=0
			32 kbyte RAM	1. lap	P=1
A15	0	0	1		
A14	0	1	x		

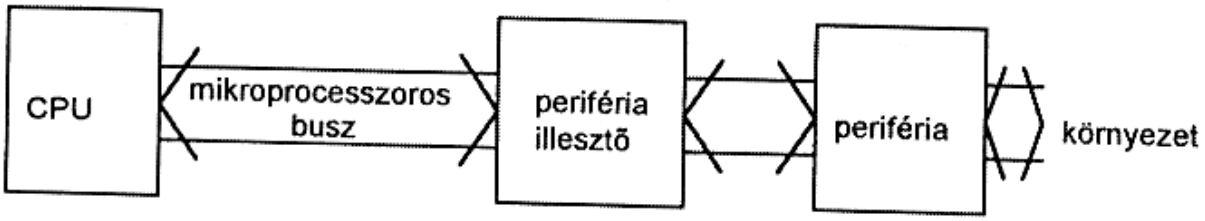
4.63. ábra. Memória térkép a lapozásos memóriához



4.64. ábra. Lapozásos memóriát is tartalmazó memória egység illesztése

4.9.2. Periféria illesztés

A perifériák biztosítják a processzor és a környezet közötti kapcsolatot. A periféria illesztő áramkör a periféria kommunikációs felületét illeszti a mikroprocesszor buszára (4.65. ábra).



4.65. ábra. A periféria illesztő funkciója

Egy általános periféria a beállítások elvégzésére és a kommunikáció segítésére különféle regiszterekkel rendelkezik.

A processzor az **üzemmód** regiszterben beállíthatja a periféria működési módját (pl. soros adónál az adás sebessége).

Írhatja ill. olvashatja a periféria **adat** regiszterét (pl. soros vevőnél a vett adatot).

Elindíthat egy folyamatot a **vezérlő** (parancs) regiszterbe való írással (pl: soros adónál az adás indítása).

A **státus** regisztert kiolvasva figyelheti, hogy mikor fejeződik be az elindított folyamat (pl. kiürült-e a soros adó adatregisztere).

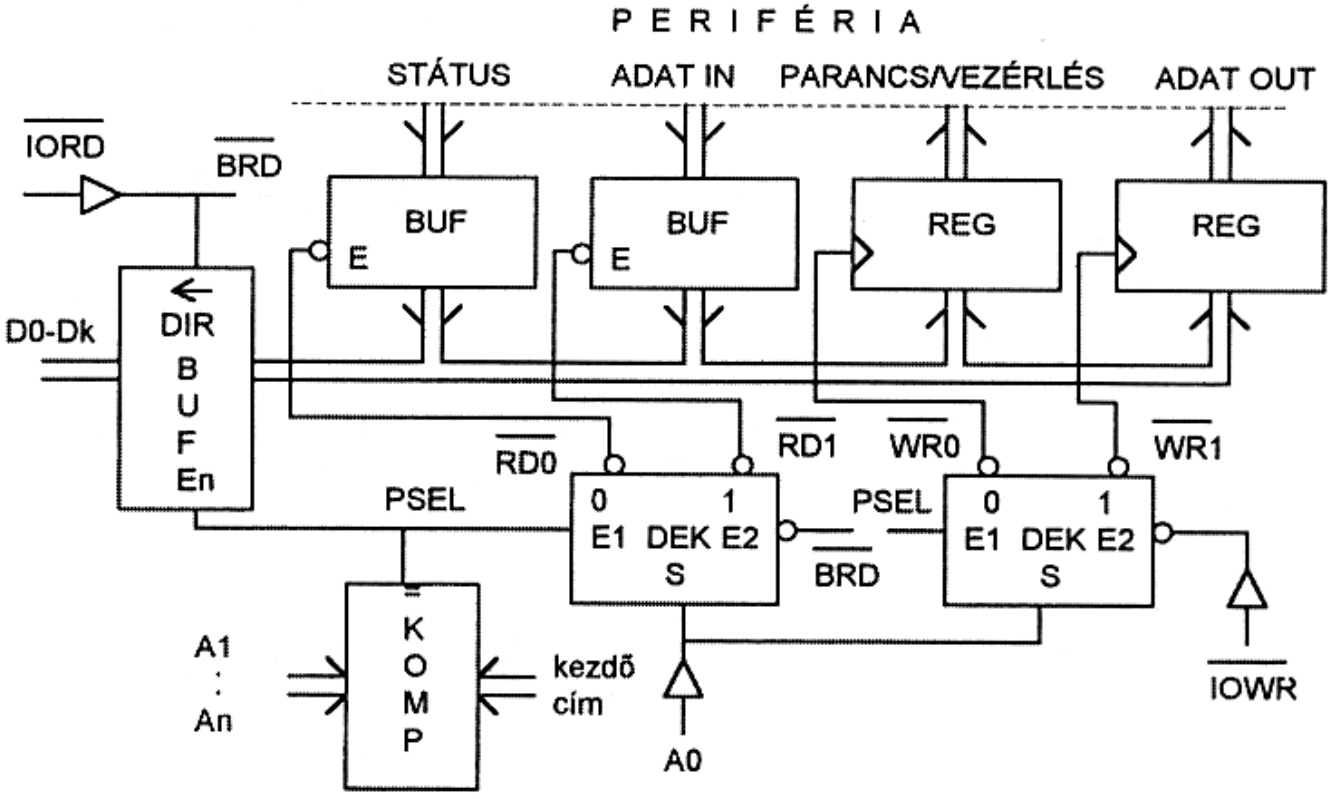
A következő periféria illesztési blokkséma (4.66. ábra) mindezeket a regisztereket tartalmazza, bár a parancs és vezérlő regisztert összevontuk.

A periféria címét a komparátor bemenetén fixen vagy kapcsolókkal beállított kezdőcím határozza meg. A CPU-k periféria címtartománya sokkal kisebb mint a memóriatartomány (256 byte-2 kbyte), ezért viszonylag kevés címbitet kell figyelni.

A periférián belüli címeket a dekóderre kapcsolódó címbitek száma határozza meg, a példában szereplő periféria mindössze 2 címet foglal el. Az író és olvasó jelek előállítását két külön dekóder végzi.

A státus és a parancs regiszter ugyanazon a címen van, vagyis erről a címről olvasva a státust kapjuk, ide írva pedig a parancsregiszterbe kerül az adat. Ugyanez a helyzet az adatregiszterrel is. Így kevesebb címet foglalunk el a periféria tartományból.

Ha a periféria olyan, hogy adatot nem termel (kimeneti periféria), akkor az adat visszaolvasását végző meghajtót vagy elhagyjuk vagy az adat regiszter kimenetére csatlakoztatjuk, s így visszaolvashatóvá válik a kiírt adat. Az ilyen jellegű visszacsatolások a tesztelhetőség szempontjából kedvezőek.



4.66. ábra. Periféria illesztési blokkvázlat

A periféria programjának írását megkönnyíti, ha a programozási tudnivalókat röviden összefoglaljuk. Ezt a célt szolgálja a **programozási felület**. Ebben feltüntetik a periféria regisztereinek címét, funkcióját és hogy írható ill. olvasható a regiszter.

A mintapélda programozási felületét mutatja a 4.4. táblázat.

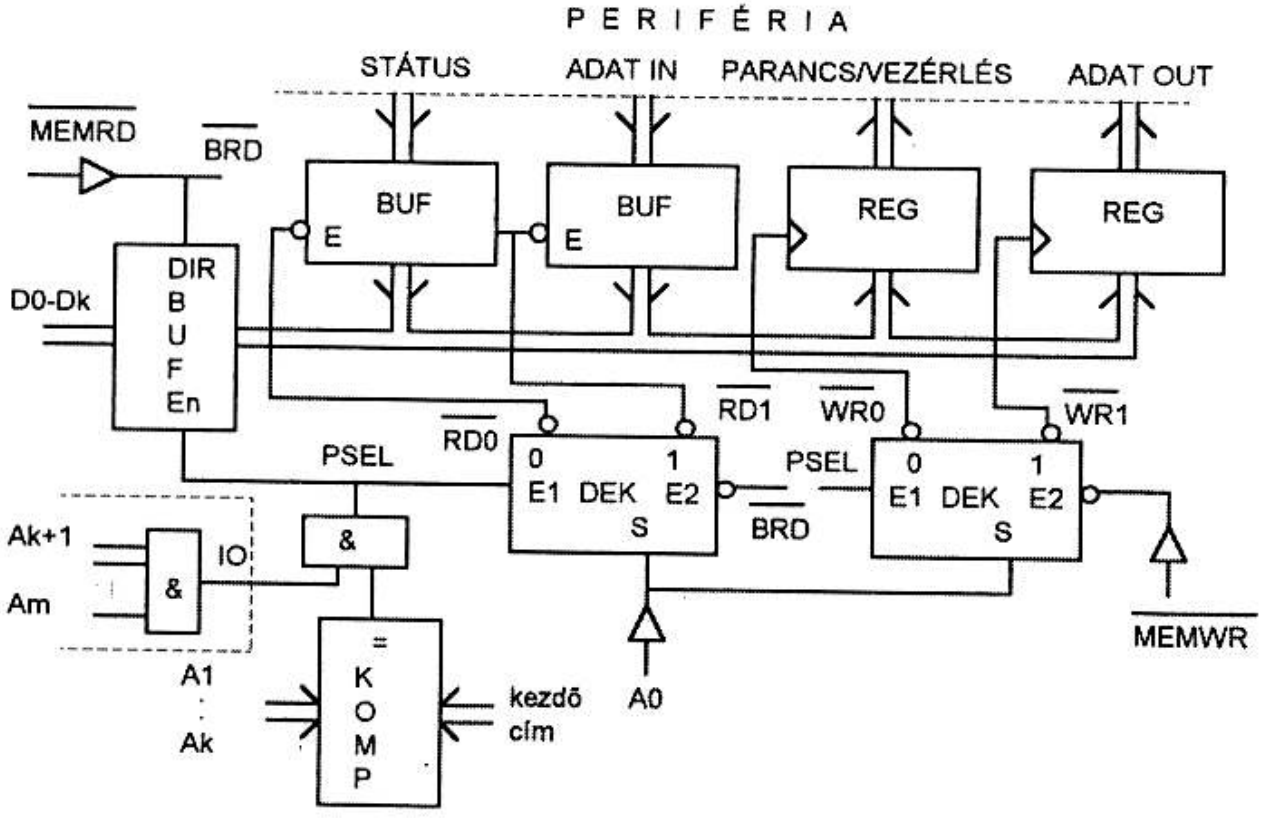
regiszter címe	funkciója	az egyes bitek szerepe	írható (W), olvasható (R), vagy mindkettő (R/W)
kezdő cím	parancs regiszter	x	W
kezdő cím	státus regiszter	x	R
kezdő cím+1	adat regiszter	x	R/W

4.4. táblázat. Programozási felület

Memóriába ágyazott periféria illesztés

Ha az az igény, hogy a perifériára is hatásosak legyenek a memória referenciás utasítások, vagy ha egy periféria sok címet lefoglal, akkor célszerű memóriába ágyazott periféria illesztést alkalmazni.

Ilyenkor kijelölünk egy memória tartományt a perifériák számára. Pl. egy 64 kbyte-os címtartományú processzornál a legfelső 1 kbyte-ot. Ezt a memória tartományt kikódolva (A15 A14 A13 A12 A11 A10 MEMRQ) előállítunk egy jelet, mely ugyanazt a szerepet játssza, mint az IORQ és célszerűen a vezérlőbusz része. A perifériának a megmaradt alsó címbiteket kell figyelni (A9-A0). Az előző példa memóriába ágyazott változata látható a 4.67. ábrán.



4.67. ábra. Memóriába ágyazott periféria illesztése

Egyszerű interruptos periféria illesztése

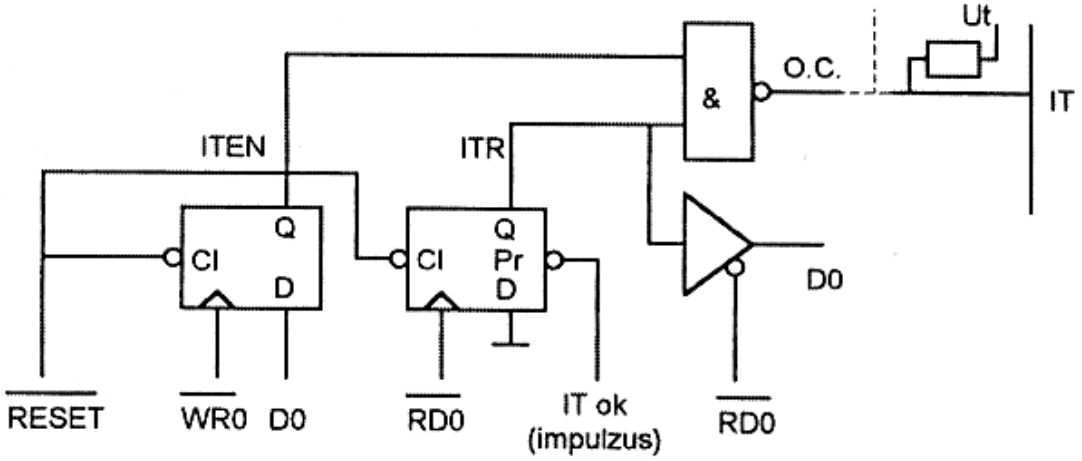
Az 4.68. ábra áramkör részlete akár egyszerű, akár vektoros IT rendszerhez alkalmazható. Utóbbi esetben az IT vonal az interrupt vezérlő egyik bementére csatlakozik és a NAND kimenete totem-pole jellegű.

Az interrupt kérés a periférián célszerűen tiltható, s az IT kérő flip-flop és esetleg az IT engedélyező flip-flop visszaolvasható. Így megmarad a lehetőség, a periféria programozott kezelésére is, továbbá egyszerű IT rendszer esetén csak így kereshető meg a megszakítást okozó periféria.

Itt az interrupt kérés feltételének teljesülésekor a periféria nem feltüntetett részének egy rövid impulzust kell generálni, ez billenti be az ITR flip-flopot. Ekkor, ha az ITEN-el előzetesen engedélyezzük, akkor az IT kimenet aktivizálódik.

Az interrupt rutinban, a státus olvasáskor generálódó RD0 jel hátsó éle törli az ITR flip-flopot. Ehelyett egy külön WR1 író impulzus is alkalmazható. (Természetesen az ITR flip-flopot bebillentő impulzusnak a törlésig meg kell szűnni, különben nem törlődik a flip-flop.)

Ügyelni kell arra, hogy bekapcsolás után az IT kérő és IT engedélyező flip-flop törölt állapotba kerüljön (RESET).

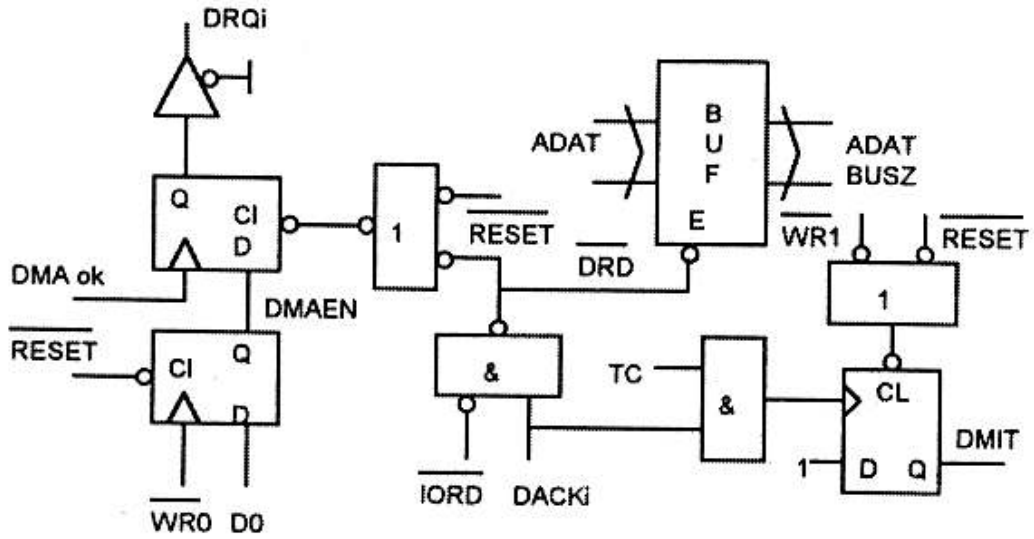


4.68. ábra. Interruptos periféria illesztés vázlata

Egyszerű DMA-s periféria illesztése

A 4.69. ábra áramkör részlete egy periféria → memória irányú DMA-s periféria illesztését mutatja be.

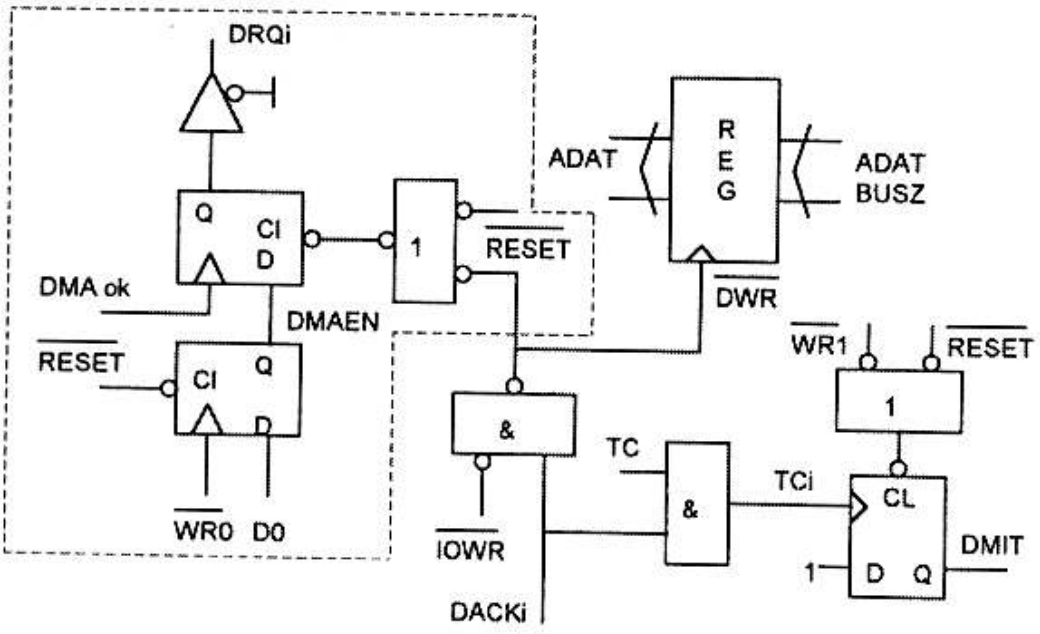
A DMA ok bebillenti a DMA-t kérő flip-flopot, ha a periféria DMA kérését előzőleg engedélyeztük. A DMA kérő flip-flop után, ha az hosszabb vonalat hajt meg, célszerű meghajtót tenni, mert a flip-flopok egy része a kimenet felől érkező zaj hatására hajlamos átbillenni. A DRQi után előbb utóbb megjön a visszajelzés a DACKi-n és az átvitel irányát és időzítését meghatározó IORD, melyek együttes megléte esetén (DRD) rákapuzódik az adatbuszra a periféria adata és visszatörlődik a DRQi. Ez blokkos átvitelnél célszerű, ahol nem kell a teljes blokk ideje alatt fenntartani a kérést.



4.69. ábra. Periféria memória irányú DMA-s illesztés vázlata

A 4.70. ábra egy memória → periféria irányú DMA-s illesztést mutat. Az áramkör bekeretezett DMA kérő része az esetek egy részében elhagyható, ha a periféria olyan, hogy az átvitel programból kezdeményezhető.

A DMA kérés elfogadása után a DMC kiadja DACKi-t és IOWR-t, s az ezekből előállított DWR jel hátsó éle írja be az adatot a periféria regiszterébe.



4.70. ábra. Memória periféria irányú DMA-s illesztés vázlata

4.9.3 Komplex memória és periféria illesztési példák

A következőkben gyakorlásképpen két komplett illesztési példát mutatunk be. Az áramköröket egy 64 kbyte címezhető tartományú szinkron busszal rendelkező CPU buszára illesztjük, melynek az 8 adat és 16 címbitjén kívül az MEMRQ, IORQ, RD, WR,

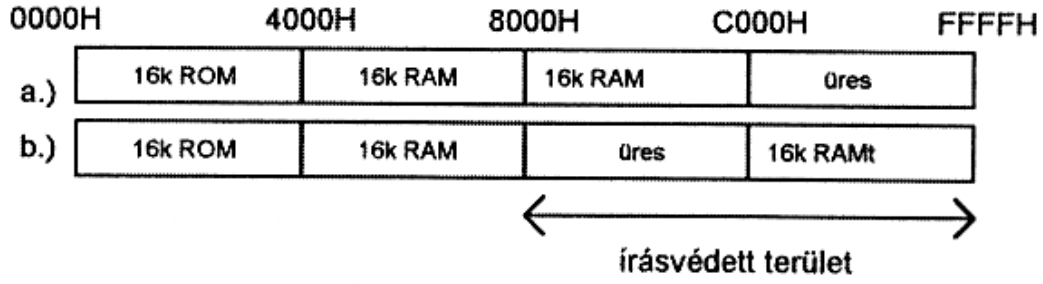
IT, RESET jeleit használjuk. A terhelési és időzítési viszonyok elemzésétől a terjedelmi korlátok miatt eltekintünk, azonban az a valós tervezési feladatoknál nem maradhat el. A feladatokat kisebb részekre bontva oldjuk meg, az egyes részeket megvalósító logikákat a megoldások blokkvázlatán bekeretezés jelöli.

Írásvédett, programozható kezdőcímű memória illesztése

Előfordul, hogy komplexebb feladatot ellátó memóriára van szükség egy rendszerben. Az alábbi példában a memória egy része írásvédett (nehezített az írás irányú hozzáférése). Erre nagyobb megbízhatóságú rendszerben lehet szükség. Egy esetleges tranziens hiba (külső zavaró jel hatására létrejövő, rövid ideig tartó hiba) esetén téves utasítást hajthat végre a processzor (pl. az adatbusz egy bitje egy ugró utasítás címrészének olvasása közben megváltozik). Ilyenkor a processzor "eltévedhet", és kiszámíthatatlan, hogy mit fog végrehajtani, ezért akár egy fontos adatterületet is felülírhat. Ha a fontos adatterület nehezített módon írható át, akkor ennek valószínűsége sokkal kisebb.

A feladat a cimtranszformációra is példa, Itt a transzformáció olyan egyszerű, hogy még kombinációs hálózat sem lesz szükséges.

A tervezendő memória programból konfigurálható, a 4.71. ábra *a* és *b* pontjaiban látható módon.



4.71. ábra. Memória térkép

A rendelkezésre álló memória egységek 1db 16 kbyte-os ROM (A0-A13, D7-D0, $\overline{CS}$, $\overline{OE}$), 1db 32 kbyte-os RAM (A0-A14, D7-D0, $\overline{CS}$, $\overline{OE}$, $\overline{WR}$).

- A legalsó 16k byte-on a ROM található.

- A 32 kbyte-os RAM a címmezőben 2db 16 kbyte-os részre van osztva (normál és írásvédett), melyből a normál rész kezdőcíme 4000H, a védetté P bittel programozhatóan a 8000H (P=0) vagy C000H (P=1) határokra esik.

Hogy a RAM melyik konfigurációban látszik azt az FEH *periféria* címre írt adat D0 bitje határozza meg (P jel).

Az írásvédett RAM terület tetszőleges címére csak az FFH periféria címhez fordulás (írásvédelem feloldása) után lehet egy byte-nyi adatot írni. Az adat beírása után a RAM automatikusan újra írásvédetté válik.

A 4.5. táblázatban megadjuk, hogy a memória térképen látható *a.)* (P=0) és *b.)* (P=1) konfigurációk esetén a RAM-ot mikor kell engedélyezni (CSRAM), ill. mely tartományt kell írásvédeni RAM (VT).

P	A15	A14	CSRA	VT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	1
0	1	1	0	1
1	0	0	0	0
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

4.5. táblázat.

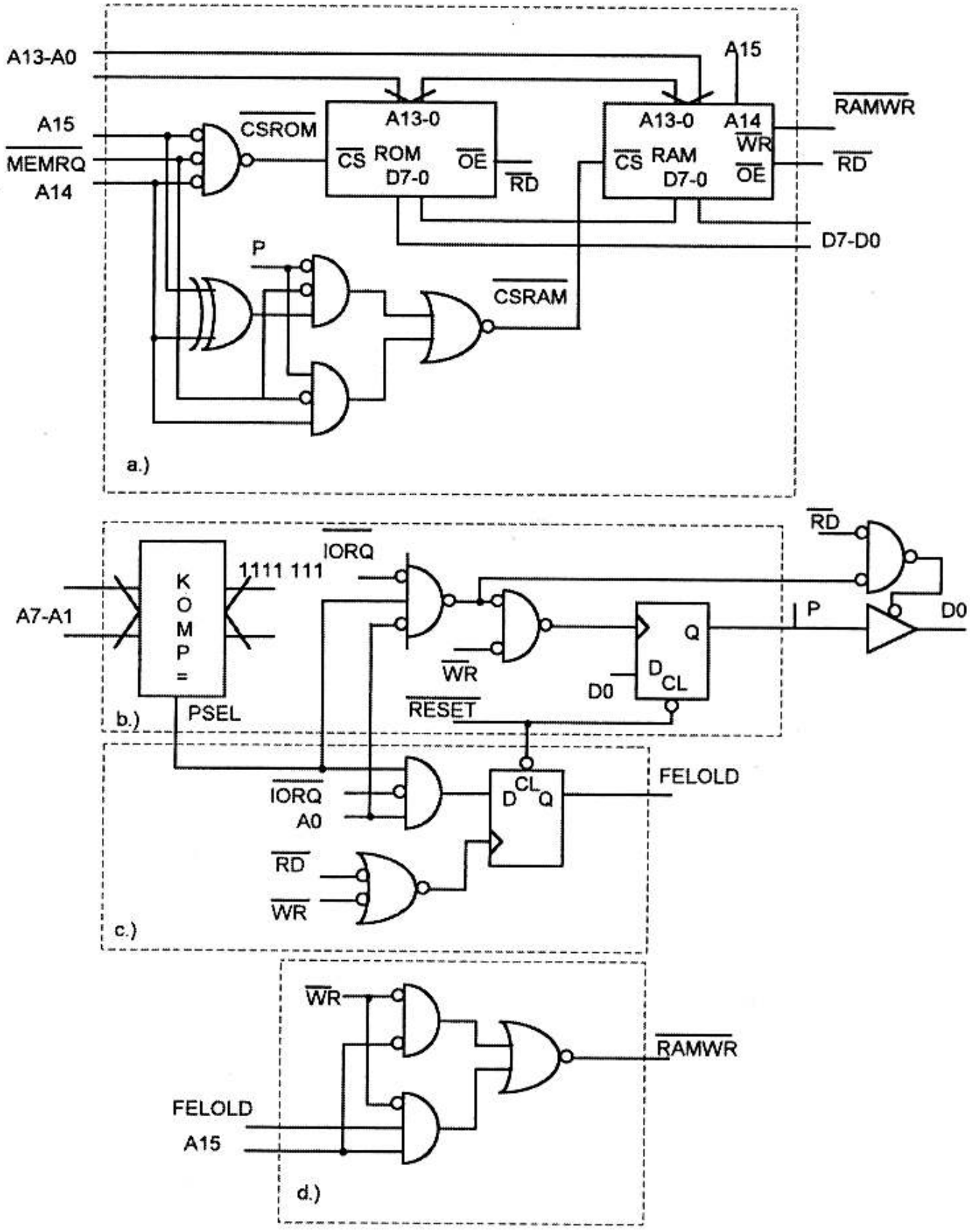
A ROM és RAM engedélyező jelének logikai függvénye a 4.5. táblázat alapján kiolvasható, csak a *MEMRQ* jellel kell a függvényeket ÉS kapcsolatba hozni. Ez a logika látható a 4.72. ábra *a*-val jelölt részén.

A RAM beírását akkor kell engedélyezni, ha nem írásvédett terület van megcímezve ($\overline{A15}\,A14$), vagy ha az írásvédett terület és az írásvédelem fel van oldva ($A15\,FELOLD$). Így a teljes függvény $RAMWR = WR\left(\overline{A15}\,A14 + A15\,FELOLD\right)$. Ez látható az ábra *d*-vel jelölt részén.

Az írásvédelem feloldásához az FFH periféria címhez kell fordulni. Ezt a periféria címet kikódoló logika egy D flip-flop adat bementére csatlakozik, mely a RD vagy WR megjelenésekor kap órajelet (ezek hátsó élénél). Így az FFH periféria címhez forduláskor 1-be íródik a flip-flop (FELOLD jel), de bármely más periféria vagy memória címre íráskor törlődik. Ezt a logikát az ábra *c*-vel jelölt része mutatja.

A konfigurációt meghatározó P jel előállítása látható az ábra *b* részén. Ez egy egyszerű visszaolvasható kimeneti port bit.

A RAM A14 bemenetére A15-öt kell kötni. Ha a RAM A14 bemenetére A14-et kötöttük volna, akkor a b.) konfigurációban a RAM ugyanazon 16 kbyte-os területe látszott volna kétszer (mint normál és mint védett RAM). A memóriatérképet megszemlélve kiderül, hogy a védett RAM terület esetén A15=1, normál esetben pedig A15=0. Így a processzor A15 jelét a RAM A14 bementére kötve (címtranszformáció), az előbbi probléma megoldódik, mert így a normál RAM mindig a chip alsó 16kbyte-ja, a védett pedig a felső 16kbyte-ja lesz.



4.72. ábra. Írásvédett, konfigurálható kezdőcímű memória illesztése

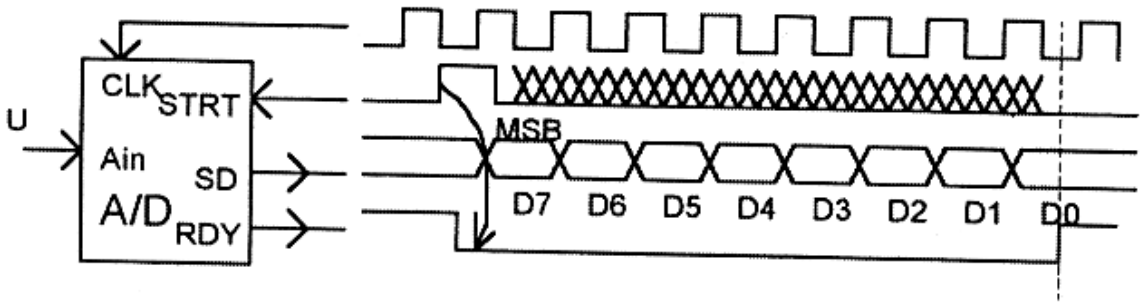
A blokkvázlaton látható logikai kapcsolást - esetleg a komparátor kivételével - programozható logikával célszerű megvalósítani (pl. ispLSI2032).

A/D konverter és automatikus feszültség határ figyelő

Az A/D konverterek analóg feszültséget alakítanak át digitális számmá. Bizonyos típusú konverterek egy indító jel hatására kezdik el a konverziót és visszajelzi ha kiolvasható az adat. IC láb spórolás és egyéb okok miatt a konverterek egy része sorosan adja ki az eredményt.

A 4.73. ábrán látható az illesztendő A/D konverter és kezelésének idődiagramja. Az A/D STRT bemenetén levő jel felfutó éle indítja az átalakítást, majd az ezt követő 8 órajel alatt (f=10kHz) az SD kimenetén jelenik meg az adat bitsorosan, az órajel *lefutó* élével szinkronban, *2-es komplemens kódban*. Az átalakítás végét a RDY vonal jelzi.

A feladat, az A/D felhasználásával egy programozható feszültség határ figyelő megtervezése. Az áramkör - programból való engedélyezés után (EN) - 1 msec-onként ciklikusan elindítja a konverziót. IT-t kér, ha az átalakított adat egy 8 biten programozható VL értéket meghalad. Egyszerű IT rendszert tételezünk fel. Rendelkezésre áll egy 10 kHz-es órajel. A periféria kezdőcíme *A0H*.



4.73. ábra. Soros A/D és kezelésének idődiagramja

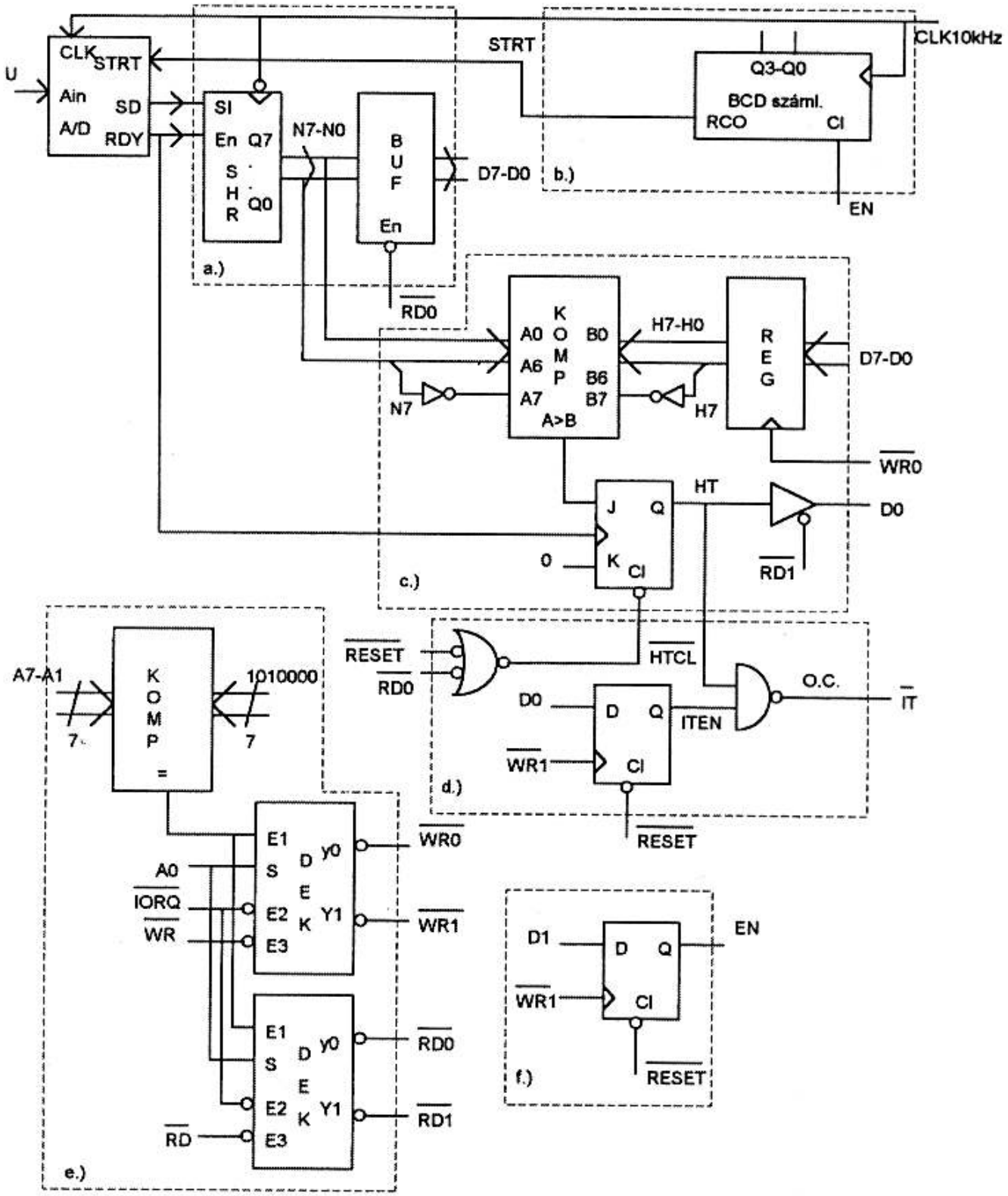
Először a soros párhuzamos átalakítást oldjuk meg, egy engedélyezhető shiftregiszter segítségével. Az idődiagramból látható, hogy az RDY jel közvetlenül felhasználható az engedélyezésre. A shiftregiszter órajele az A/D órajelének negáltja, mert a specifikáció szerint az adatok ezzel szinkronban jönnek. A shiftregiszter kimenetére egy három állapotú buffer kapcsolódik, így az adat visszaolvasható. Ezt 4.74. ábra *a*-val jelölt része mutatja.

Az 1 msec-os ciklikus újraindítást egy törölhető 10-es számláló kezdeményezi. (4.74. ábra *b* része). Ez leosztja a 10 kHz-es órajelet 10-el, így a RCO (ripple clock) kimenetén már előáll az 1 msec periódusidejű hazárdmentes impulzus. Az engedélyezést a számláló CI statikus törlő bemenetével oldottuk meg. A számláló egy kimeneti port bittel törölhető ill. engedélyezhető (EN). Ez a bit bekapcsolás után törölt állapotban tartja a számlálót.

A határ figyelést egy 8 bites komparátor végzi. Az A/D és a határértéket tároló regiszter felől érkező kettes komplemensben ábrázolt számokat az előjel bit invertálásával offszet kódúvá alakítjuk, így a komparátor helyesen jelzi a számok viszonyát. A komparátor kimenetét az átalakítás végét jelző RDY jel mintavételezi egy J-K flip-flopba. Ha a komparátor határ túllépést jelez, akkor a flip-flop bebillen (HT=1). A HT jelet (ill. az IT kérést) az adat kiolvasása és a RESET törli. A HT jelre egy három állapotú buffer kapcsolódik, így annak állapota programból lekérdezhető. Ezt a logikát mutatja 4.74. ábra *c* része.

A HT jel interruptot okoz, ha az programból engedélyezve van (ITEN). Az IT engedélyezést a RESET tiltja (4.74. ábra *d* része).

A címdekódert mutatja a 4.74. ábra *e* része. A periféria 4 címet foglal le. A programozási felülete a 4.6. táblázatban látható.



4.74. ábra. A/D konverter és automatikus feszültség határ figyelő blokkvázlata

regiszter címe	funkciója	bitek szerepe D7 D6 D5 D4 D3 D2 D1 D0	típusa R (olvasható), W (irh.), R/W (mindkettő)
A0H	adat	U7-U0 (a feszültség értéke)	R
A0H	adat	VL7-VL0 (határ érték)	W
A1H	státus	x x x x x x x IT	R
A1H	parancs	x x x x x x EN ITEN	W

4.6. táblázat. Programozási felület

5. MIKROKONTROLLEREK

A mikrokontrollerek (μC-k) olyan áramkörök, amelyeknél egy IC-ben megtalálható a mikroprocesszor, egy kisméretű RAM (kb. 64 byte - 2 kbyte), többségüknél ROM EPROM vagy EEPROM (kb. 2-16 kbyte) és több olyan periféria, amelyek segítségével egyszerűbb vezérlések alakíthatók ki, viszonylag kevés egyéb áramköri elem felhasználásával.

A mikrokontrollereket a mindennapi környezetünkben sok helyen megtalálhatjuk, alkalmazzák az audio és video magnók vezérlésére, kapcsoló órákban, a CD lejátszókban, a winchesterekben stb.

A különféle mikrovezérlőkben leggyakrabban előforduló perifériák:

- párhuzamos ki és bemeneti portok,
- időzítő áramkörök (timerek),.
- kétirányú soros vonali interfész (aszinkron és szinkron),
- A/D, D/A konverter,
- LCD meghajtó.

Az utasításkészletben egyrészt a processzoroknál megszokott általános célú utasítások (adat mozgató, ugró, aritmetikai stb.), másrészt a saját perifériák kezelését könnyítő speciális célú utasítások találhatók meg.

A továbbiakban az ipari standardnak számító MCS-51 család, néhány tagjával foglalkozunk röviden.

5.1. Az MCS 51 mikrokontroller család

A család fontosabb tagjai a 8751H, 80C51, 80C52, 80C31, 80C32.

4 db 8 bites párhuzamos port. Ebből a 80c31-nél kettő a külső memória illesztéséhez szükséges. A belső ROM-mal, EPROM-mal rendelkező változatoknál ezek portként is felhasználhatók.)

- 4 kbyte belső ROM (80C51) vagy EPROM (8751H),
- 8 kbyte belső ROM (80C52)
- 64 kbyte külső ROM és 64 kbyte külső RAM, illesztési lehetőség,
- 128 byte belső RAM. A 80C32/52 esetén 256 byte, (a stack a belső RAM-ban van)

- 2 db 16 bites időzítő/számláló (IT lehetőség). A 80C32/52 még egy 16 bites időzítőt tartalmaz.
- Kétirányú soros vonal (aszinkron vagy speciális szinkron üzemmód, IT lehetőséggel),
- 2 külső interrupt bemenet (programozhatóan szint vagy él érzékeny),
- 2 szintű interrupt rendszer,
- 4db 8 byte-os általános célú regiszterbank a RAM részeként, speciális funkciójú regiszterek,
- bit címzési lehetőség, byte szélességű egész számokat szorzó és osztó utasítások.

5.1.1. A mikrokontroller jelei és funkciójuk

A lábszámok dual in line tokozásra vonatkoznak.

lábsz.	jelnév	funkció
1-8	**P1.0-7**	8 bites kvázi kétirányú I/O port, belső aktív felhúzással, és 80C51/52-nél a cím alsó fele a program verifikálás (a belső EPROM programozásának ellenőrzése) alatt.
9	**RST**	Bekapcsolási reset (legalább 2 órajelnyi magas szint hatására áll alaphelyzetbe az áramkör).
10-17	**P3.0-7**	8 bites kvázi kétirányú I/O port, belső aktív felhúzással. A P3.0-P3.5 bitek alternatív (másodlagos) funkcióra is programozhatók, a P3.6, P3.7 viszont a 80C31/32-nél csak az alternatív funkcióra alkalmasak.

Az alternatív funkciók:

P3.0 **RXD/data** Aszinkron soros vonali bemenet, vagy szinkron soros vonali be/kimenet.

P3.1: **TXD/clock** Aszinkron soros kimenet vagy órajel kimenet a szinkron soros kommunikációhoz.

P3.2 $\overline{\textbf{INT0}}$ Külső 0. interrupt vagy a 0-ás időzítő/számláló kapuzó bemenete.

P3.3 $\overline{\textbf{INT1}}$ Külső 1. interrupt vagy az 1-es időzítő/számláló kapuzó bemenete.

P3.4 **T0** A 0-ás időzítő/számláló bemenete.

P3.5 **T1** Az 1-es időzítő/számláló bemenete.

P3.6 $\overline{\textbf{WR}}$ Külső memória beíró impulzus (80C51/52 esetén ki/bemeneti port bitként is használható, ha nincs külső memória).

P3.7 $\overline{\textbf{RD}}$ Külső memória olvasás jel (80C51/52 esetén ki/bemeneti port bitként is használható, ha nincs külső memória).

18	**XTAL1**	Kvarc kristály bemenet vagy külső óragenerátor bemenet.
19	**XTAL2**	Kvarc kristály bemenet vagy külső oszcillátor alkalmazása esetén VSS potenciálra kötendő.

20	**VSS**	Föld potenciál.
21-28	**P2.0-7**	Külső memória cím felső fele (80C51/52 esetén ki/bemeneti portként is használható ha nincs külső memória, illetve a belső EPROM címző bemenete, a program verifikálás alatt).
29	**$\overline{\text{PSEN}}$**	Program memória olvasás jel.
30	**ALE**	Cím latch engedélyező kimenet. Külső memóriához fordulás esetén a multiplexált adat/címbuszon megjelenő cím (a teljes cím alsó fele) latch-be tárolását vezérli.
31	**$\overline{\text{EA}}$**	Logikai alacsony szintre állításával kell jelezni a kontrollernek, ha a programot külső memóriából kell végrehajtani (80C51/52 esetén a magas szintre állításával kell jelezni, hogy a programot a belső EPROM (is) tartalmazza).
32-39	**P0.7-0**	Multiplexált adat/címbusz. (80C51/52 esetén ki/bemeneti portként is használható, ha nincs külső memória).
40	**VCC**	+5V-os tápfeszültség bemenet.

5.1.2. A memória szervezés

A mikrokontrollernek szétválasztott kód és adat memóriája van (ezt Harward architektúrának nevezik).

A külső program és adat memória

A buszára 64 kbyte külső program memória és ugyanennyi külső adat memória illeszthető. A 8051/52 esetében belső kód memória (EPROM) is van mely az AEN jellel engedélyezhető. Engedélyezett esetben a külső buszra csak a 64 kbyte-ból megmaradt kód memória rész illeszthető.

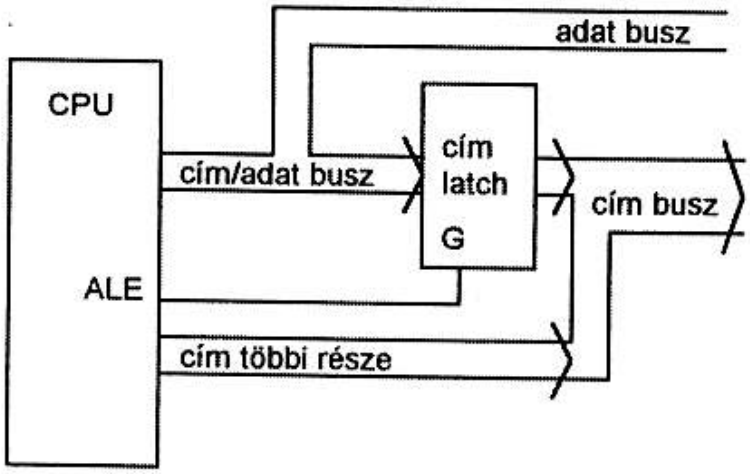
A külső program és adat memória ugyanazon a címtartományon helyezkedik el (0000h-FFFFh), azokat a vezérlő jelek különböztetik meg.

- A program memóriát (external ROM) a µC **$\overline{\text{PSEN}}$** jele engedélyezi.
- A külső adat memóriához (external RAM) a **$\overline{\text{RD}}$** és **$\overline{\text{WR}}$** jelek vannak rendelve.

A program és adat memória sohasem lesz egyszerre aktív (a cím- és adatbusz közös).

A µC adatbuszának alsó fele multiplexált cím/adat busz. Mint már volt róla szó, ilyenkor először a cím jelenik meg, ezt jelzi az ALE jel, majd az adat (a 4.7. ábra szerint). Külső memória vagy memóriába ágyazott periféria illesztéshez a cím alsó felét tárolni kell egy latchben az ALE jellel, így áll elő a külön cím és adatbusz.

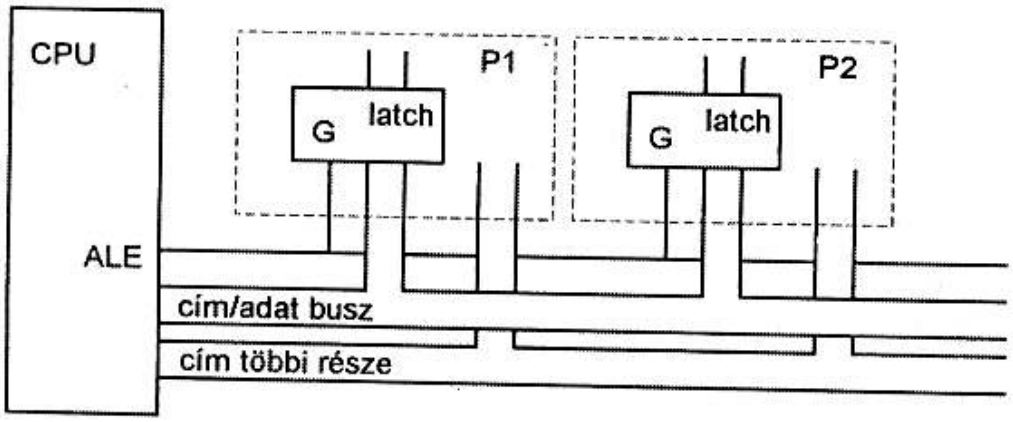
Multiplexált cím/adat busz esetén két lehetőség is van a rendszer buszfelületének kialakítására.



5.1. ábra. Központi cím latch a multiplexált cím/adat buszon

A gyakrabban használt lehetőség, hogy a processzor mellett helyezzük el a cím latchet, s a címbuszt innen vezetjük ki. Ekkor több vezetékből áll a busz, mint eredetileg (5.1. ábra).

Második esetben meghagyjuk az eredeti busz felületet, ekkor azonban minden megcímezhető egységben szükséges egy latch, a cím tárolására, ami hátrány. Ennek a kialakításnak előnye viszont, hogy kevesebb busz vezetéket igényel (5.2. ábra).



5.2. ábra. Elosztott cím latch a multiplexált cím/adat buszon

A program memória elérése

A program memóriában levő adatokhoz két utasítással lehet hozzáférni (a FETCH cikluson kívül csak ezek az utasítások generálnak $\overline{\textbf{PSEN}}$ jelet):

MOVC A,@A+DPTR az A-ba kerül az adatpointer és az A tartalmának összegeként adódó címről az adat.

MOVC A,@A+PC az A-ba kerül az aktuális utasításhoz képest az A-ban megadott offsetre lévő adat.

Az A-ban levő offsetet mindkét utasítás esetén kettes komplemensben értelmezi a processzor, így az eltolás +127 és -128 között lehet.

Az külső adat memória elérése

A külső adat memóriára 4 utasítással lehet hivatkozni:

A memória első 256 byte-ja az R0 vagy R1 regiszteren keresztül is elérhető. Ez főként akkor hasznos, ha a külső adat memória mérete nem nagyobb 256 byte-nál:

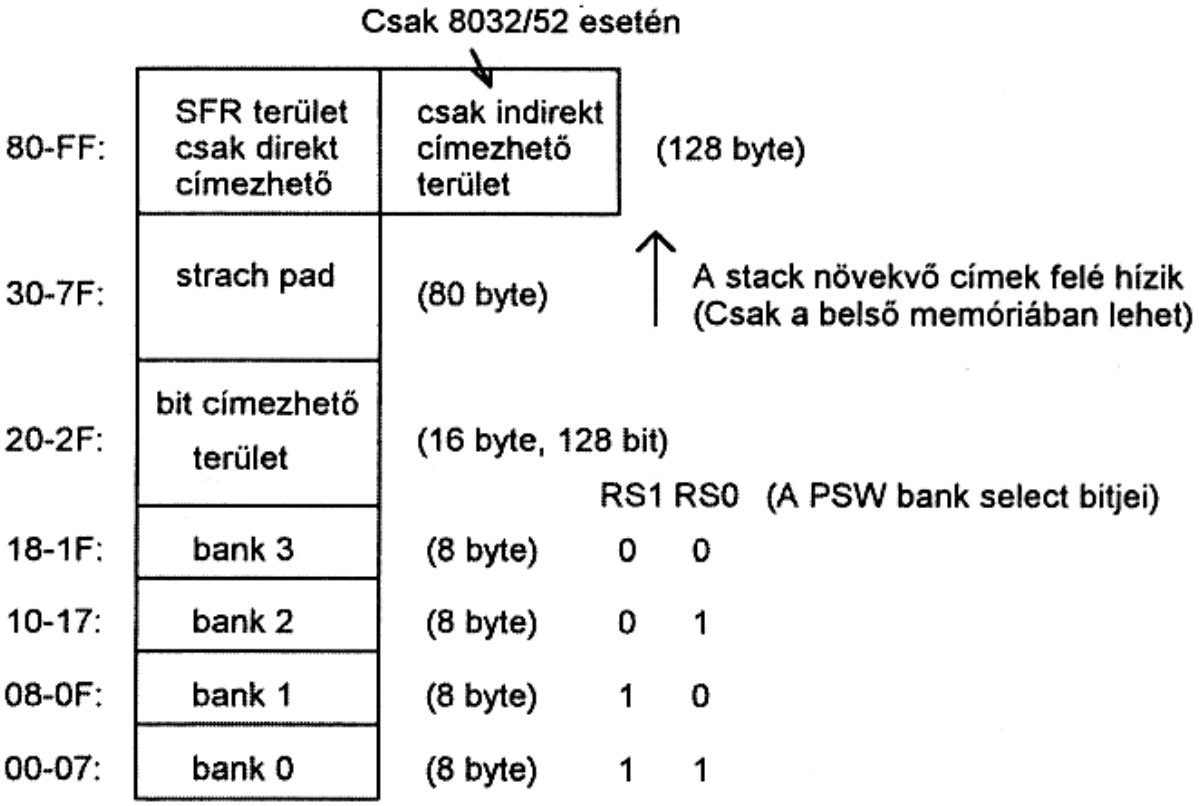
MOVX A,@Ri
MOVX @Ri,A

A teljes 64 kbyte az adat pointer segítségével címezhető.

MOVX A,@DPTR
MOVX @DPTR,A

A belső RAM

A μC-nek 128 byte ill. 80C32/52 esetén 256 byte belső adat memóriája is van (5.3. ábra).



5.3. ábra. A 8031/32/51/51 belső memóriájának felosztása

A belső RAM első 32 byte-jának (00h-1Fh) kitüntetett szerepe van. Itt helyezkedik el a 4 *regiszter bank*. Egy regiszter bank valamely elemére egy utasításban az R0, R1,.., R7 jelöléssel hivatkozhatunk. Egyszerre csak egy regiszter bank lehet aktív. Az aktív regiszter bankot a később ismertetendő program státus szó (PSW) két bitjével (RS1, RS0) lehet beállítani. *Indirekt címzésre csak az R0 és R1 regiszter használható (a @R0, @R1 hivatkozással)!*

Közvetlenül a regiszter bankokat követi a ***bit címezhető terület*** (20h-2Fh). Erre a területre a direkt és indirekt címzésen kívül *direkt bit címzés*sel is lehet hivatkozni. Itt a

bitcím 0-7F lehet, ezt az ugyanolyan belső RAM címektől maga a bit hivatkozású utasítás kódja különbözteti meg. A bit címzésére az assemblerek többsége a direktcím.bit jelölést is megengedi, ahol direktcím 20h-7Fh, a bit kijelölés 0-7 lehet (pld: 20.0, 20.1,..., 2F.7) .

A 30h-7Fh terület a scratch pad, amelyet egyrészt szokványos RAM-ként lehet felhasználni, másrészt ennek a területnek a felső részére szokás a stacket elhelyezni. (A stack nem tehető a külső memóriába!) A stack a processzorok többségével szemben a magasabb címek felé hízik.

A 80h-FFh területen csak direkt címzéssel elérhetően a speciális funkciójú regiszterek (SFR) helyezkednek el (lásd később). Ugyanezen a címtartományon a 8032/52 µC-k esetén 128 byte-nyi RAM terület is van, mely csak indirekt címzéssel érhető el.

A belső RAM elérése

A belső RAM első 128 byte-ja direkt címzéssel és regiszter indirekt címzéssel is címezhető.

MOV A,cím
MOV cím,A

MOV A,@Ri
MOV @Ri,A

A speciális célú regiszterek és funkcióik

A belső memóriát követő 128 byte-os területen helyezkednek el az ún. speciális funkciójú regiszterek (**SFR**). Ezek egy része a processzoroknál megszokott funkciókat látja el, más részük pedig a belső perifériák parancs és adat regiszterei. *Az SFR területre csak direkt címzéssel lehet hivatkozni* (az utasítások mnemonikjában a direkt cím helyére a megfelelő SFR regiszter nevét kell írni). A *-al jelöltek bitenként is címezhetők.

***ACC** Az akkumlátor, az aritmetikai műveletek és a legtöbb logikai művelet célregisztere.

***B** Az osztás maradéka, ill. a szorzás eredményének felső byte-ja itt tárolódik (általános célú regiszterként is használható)

***PSW** A program státus szó, a bitek jelentése:

D7: **CY** Carry flag (aritmetikai túlcsordulás), a bit címzésű terület bitjei és a CY között adatmozgatás és logikai műveletek végezhetők.

D6: **AC** Auxiliary carry flag a byte alsó és felső négy bitje közötti aritmetikai túl- vagy alulcsordulást jelzi.

D5: **F0** Általános célú flag.

D4-3: **RS1,RS0** Regiszter bank kiválasztó bitek (00 a BANK0, 01 a BANK1 stb.).

D2: **OV** Az overflow flag, jelzi, hogy kettes komplemens aritmetikában az előjeles eredmény nem fér el 8 biten.

D1: Értéke közömbös.

D0: **P** A parity flag, jelzi, ha az akkumlátorban levő 1-esek száma páros.

SP A stack pointer. A stack a belső RAM-ban helyezkedik el és az inicializálási címtől növekszik.

DPTR Az adat pointer (16 bites), a külső memóriához fordulás (external RAM és ROM) az ebben tárolt címre lehetséges (ez alól a külső RAM első 256 byte-ja kivétel, mely az R0 és R1-el is címezhető). Külön is lehet hivatkozni a DPTR alsó és felső byte-jára (DPL, DPH).

***P1** Az 1-es port, párhuzamos be/kimenetként használható.

***P3** A 3. port, párhuzamos be/kimenetként, vagy az alternatív funkciókra használható (soros vonal, timer bemenetek).

***IP** Interrupt prioritás vezérlő regiszter (0: alacsonyabb prioritás), a bitek jelentése:

D7-5: Értéke közömbös.
D4: **PS** A soros vonal prioritás.
D3: **PT1** Az 1-es időzítő/számláló prioritás.
D2: **PX1** Az 1-es külső IT prioritás.
D1: **PT0** A 0-ás időzítő/számláló prioritás.
D0: **PX0** A 0-ás külső IT prioritás.

***IE** Az interrupt engedélyező regiszter, a bitek jelentése (1:engedélyez):

D7: **EA** A globális IT engedélyezés.
D6-5: Értéke közömbös.
D4: **ES** A soros vonali IT engedélyezés.
D3: **ET1** Az 1-es időzítő/számláló engedélyezés.
D2: **EX1** Az 1-es külső IT engedélyezés.
D1: **ET0** A 0-ás időzítő/számláló engedélyezés.
D0: **EX0** A 0-ás külső IT engedélyezés.

TMOD Az időzítő/számláló üzemmód regisztere.

***TCON** Az időzítő/számláló vezérlő regisztere.

TH0 A 0-ás időzítő/számláló felső 8 bitje.

TL0 A 0-ás időzítő/számláló alsó 8 bitje.

TH1 Az 1-es időzítő/számláló felső 8 bitje.

TL1 Az 1-es időzítő/számláló alsó 8 bitje.

***SCON** A soros port vezérlő regisztere.

SBUF Az elküldendő adat ill. a vett adat (egy csak írható és egy csak olvasható regiszter ugyanazon a címen).

PCON A power control regiszter

Mivel a belső perifériák működését és programozását terjedelmi korlátok miatt nem részletezzük, ezért azok regisztereinek bit kiosztását sem tüntettük fel az előbbi felsorolásban.

5.1.3. A mikrokontroller interrupt rendszere

A processzorban 5 vagy 6 (80C32/52) interrupt forrás van, a két külső IT, a két (három) timer és a soros vonal. A soros vonali adás és a vétel egy IT-hez van rendelve, ezért az IT okát szoftverből kell kideríteni.

Az interruptok egyenként engedélyezhetők vagy tilthatók az IE (interrupt engedélyező regiszter) segítségével. Az interrupt globálisan is engedélyezhető vagy tiltható ugyanezen regiszter EA bitjével.

Minden interrupt forrás tetszőlegesen rendelhető a két prioritási szint valamelyikéhez az IP (interrupt prioritás regiszter) megfelelő bitjének beállításával. Egy alacsonyabb prioritásra beállított interruptot megszakíthat egy magasabb prioritású, de nem szakíthat meg egy vele azonos prioritású interrupt.

Az IT flageket minden gépi ciklus végén mintavételezi a processzor. Egy interruptot akkor fogad el, ha:
- éppen nem fut egy magasabb prioritású IT rutin,
- az aktuális gépi ciklus a legutolsó ciklusa az utasításnak
- az utasítás nem RETI, vagy nem az IE vagy IP regiszterekre vonatkozik.

Az interrupt elfogadásakor automatikusan egy LCALL utasítás generálódik, az IT októl függő címre. Ezek a következők:

- külső 0-ás IT: 0003h
- 0-ás timer: 000Bh
- külső 1-es IT: 0013h
- 1-es timer: 001Bh
- soros vonal: 0023h
-2-es timer 002Bh

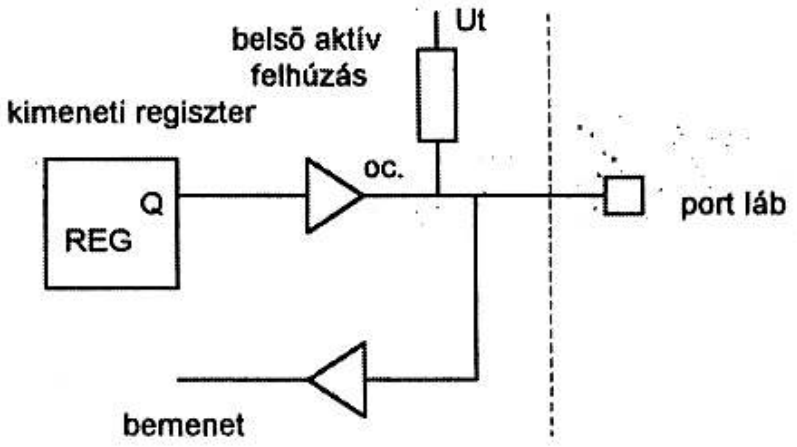
Az LCALL elmenti a következő utasítás címét a stackre, majd a megfelelő címre adja a vezérlést. A fenti címekre a megfelelő IT rutinra ugró utasítást lehet elhelyezni. Az IT elfogadása nem tiltja le automatikusan a globális IT engedélyezést, amint azt a legtöbb processzor teszi, ezért ha ez szükséges, az EI flag törésével tehetjük meg (ha az EI flag törlése közben jön IT, az már nem érvényesül), de ez esetben a RETI előtti újbóli engedélyezésről nem szabad elfeledkezni. Egy IT rutint RETI utasítással kell befejezni, innen tudja a processzor, hogy az aktuális IT véget ért (pld. elkezdődhet egy közben bejött azonos prioritású kiszolgálása).

5.1.4. A portok kezelése

A 8031/32-nek csak a P1 és P2 portja használható ki/bemenetként (a többit a külső memória illesztés lefoglalja). A 8051/52 esetén a többi port is használható, ha nem használjuk ki a külső memória illesztési lehetőséget ($\overline{EA}$=1). A portokra az SFR-ekre (speciális funkciójú regiszterek) vonatkozó utasításokkal lehet írni pld:

```
MOV P1,A
MOV P1,Rr
MOV P1,@Ri      (i:0,1)
CLR P1.0
```

A port bitek kialakításának elvét mutatja az 5.4. ábra.



5.4. ábra. Portok kialakítása

A bemenetként használt port bitekre 1-et kell kiírni. A P2 alternatív funkciói csak akkor működnek, ha a megfelelő port bitet 1-be programozzuk (RESET után ez az állapot áll elő). Ha az alternatív funkciót RESET után nem használjuk, vagy olyan állapotot biztosítunk, hogy az alternatív funkciójú kimenet magas szintet ad, akkor a megfelelő port biteket ki/bemenetként használhatjuk.

5.1.5. Kis fogyasztású állapotok

A mikrokontrollernek két kis fogyasztású üzemmódja van. Ezeket elemes, akkumlátoros táplálást igénylő esetekben célszerű alkalmazni. A PCON regiszter megfelelő bitjeivel állíthatók be.

Power down állapot

Leállítja az oszcillátort, az áramfelvétel lecsökkentése érdekében. A belső RAM és az általános célú regiszterek megőrzik tartalmukat, de minden órajelet igénylő funkció leáll (timerek, soros vonal, CPU). Az ALE és $\overline{\text{PSEN}}$ jelek alacsony logikai szintre állnak be. Csak a RESET hozza ki ebből az állapotból. Ezt az állapotot a mikrokontroller "kikapcsolására" lehet használni abban az esetben, ha bizonyos információkat szeretnénk megőrizni.

Idle állapot

A CPU órajele leáll, de az időzítők és a soros vonal tovább működik, a belső regiszterek megtartják értéküket, vagy a beállított üzemmód szerint változnak. Az ALE és a $\overline{\text{PSEN}}$ jelek magas szintre állnak. Ez az állapot csak egy IT vagy a RESET hatására szűnik meg (IDL törlődik). IT esetén egy újabb IDL üzemmódot csak a RETI végrehajtását követően lehet beállítani!

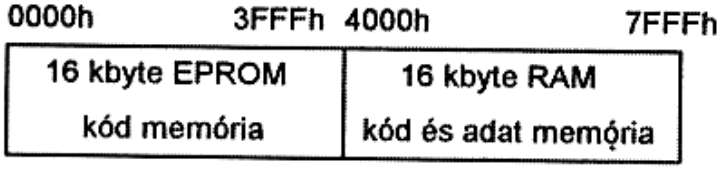
Ezt pl. olyan esetben célszerű alkalmazni, ha a készülék főprogramja nem csinál semmit, minden IT szinten fut, s az interruptok nem jönnek túl sűrűn.

5.1.6. Egyszerű külső memóriát és soros vonalat tartalmazó rendszer kialakítása

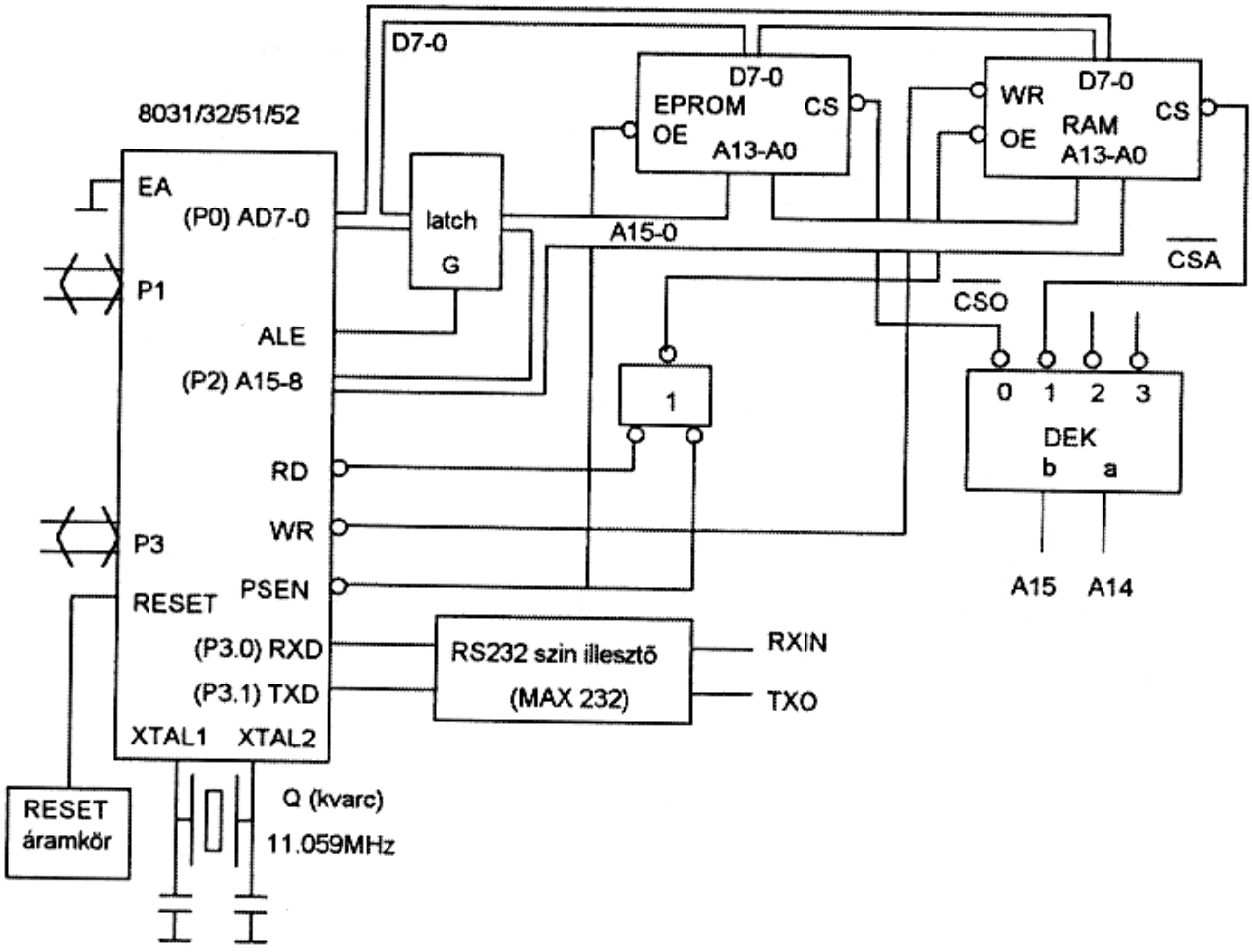
Program fejlesztési célokra olyan memória konfiguráció is kialakítható, amelyben a külső RAM-ba programot lehet tölteni és onnan futtatni. Ehhez a RAM-ot program memóriaként is elérhetővé kell tenni. Ha a RAM olvasást a RD helyett a PSEN+RD

engedélyezi, akkor ez teljesül. Ekkor azonban a ROM és az ily módon használt RAM területe nem fedheti át egymást.

A program letöltését pl. az aszinkron soros vonalon keresztül végezhetjük el, egy a μC-n futó ún. monitor program segítségével. Egy ilyen rendszer blokkvázlatát mutatja az 5.6. ábra. Itt egy 16 kbyte-os EPROM-ot és egy 16 kbyte-os RAM-ot illesztettünk a mikrokontrollerhez, ahogy azt az 5.5. ábra. memória térképe mutatja.



5.5. ábra. Memória térkép



5.6. ábra. Egyszerű fejlesztő kit blokkvázlata

Az aszinkron soros vonali be és kimenetre az ún. RS232 szabványban előírt feszültség szintek teljesítéséhez egy szint illesztő áramkört kell kapcsolni. Így a készülék pl. a PC soros vonalához kapcsolható.

5.1.7. Az MCS 51-el kompatibilis újabb mikrokontrollerek

Az MCS 51-es család széleskörű elterjedése miatt, némileg eltérő, de részben vagy egészben kompatibilis processzorokat is elkezdtek gyártani. Ezek közül a legsikeresebbek közé tartozik az ATMEL AT89xx családja.

A család összes tagja FLASH program memóriával rendelkezik. Ez egy EEPROM jellegű, elektromosan törölhető és programozható memória. Az 5.1. táblázatban látható egy rövid összefoglaló a család elemeinek legfontosabb paramétereiről.

A család lehetőséget ad, hogy az MCS51-nél megszokott programozási környezetet használjuk az újabb fejlesztésekhez, mivel szoftver kompatibilis. Ugyanakkor a legtöbb feladathoz megtalálható egy ár és tudás szempontjából viszonylag optimális típus.

Külön kiemelésre érdemes az AT89S8252 típus. Ennek egy speciális szinkron soros programozó bemenete is van, és nem igényel különleges feszültség szinteket az ezen keresztül történő programozáshoz. Így akár egy olyan rendszer is létrehozható, melyben egy másik mikrokontroller, mely soros vonalon (pl. modemről) kapja az adatokat, ezen az interface-en keresztül átprogramoz egy AT89S8252-t. A chipben 2 kbyte EEPROM is van, mely programból átírható, így a tápfeszültség megszűnése után is megőrizhetők adatok. A külső memória kezelésének megkönnyítésére itt két DPTR (adat pointer) is rendelkezésre áll.

	AT89C 1051	AT89C 2051	AT89C 51/LV51	AT89C 52LV52	AT89C55	AT89S825 2
FLASH	1k	2k	4k	8k	20k	8k
RAM (byte)	64	128	128	256	256	256
timer	1	2	2	3	3	4
SIO	-	+	+	+	+	+
port bitek	15	32	32	32	32	32
órajel frekv.	24MHz	24MHz	24MHz	24MHz	33MHz	33MHz

5.1. táblázat. Összefoglaló az AT89xx családról

FÜGGELÉK

1. Az ABEL-HDL nyelv (ABEL V4.30)

1.1. Az ABEL HDL nyelv elemei

Szintaktikai előírások

Az ABEL-HDL a következő alapvető szintaktikai előírásokkal dolgozik:

- Egy programsor legfeljebb 150 karakter hosszúságú lehet;
- A sorokat <LF> (soremelés), <FF> (lapdobás) vagy <VT> (vertikális tabulátor) karakterek zárhatják le. (Megfelelő természetesen az általánosan használt <CR><LF> kombináció is, pl. az <Enter> gomb az IBM PC-ken.)
- Azonosítókat, kulcsszavakat és számokat 'space' (szóköz) karakterekkel kell elválasztani. Nem kötelező a szóköz használata olyan helyeken, ahol az egyes szintaktikai egységek között vessző vagy egy operátor áll, ill. zárójelek előtt és után.
- Azonosítók, kulcsszavak és számok belsejében szóköz nem lehet;
- A nyelv kulcsszavai kis- és nagybetűs formában (akár keverve is) egyaránt szerepelhetnek;
- A felhasználói azonosítók (pl. jelnevek) tetszőlegesen tartalmazhatnak kis és nagybetűket, de a fordító ez esetben érzékeny a kis- és nagybetűkre (case sensitive).
- Deklarációkat, egyenleteket egy ; jellel kell lezárni, (a module, end, és title utasításokat nem)
- A " jel a kommentár kezdetét jelzi, a kommentárt egy másik " jel vagy a sorvég zárja

Azonosítók

A használható (érvényes) karakterek:

A - Z (nagybetűk), a - z (kisbetűk), (csak ASCII betűk, ékezetes betűk NEM)
0 - 9 (számok)
! @ # $ % ^ & * ( ) - = + [ ] { } ; : ' "
` ~ \ | , < > . / ? _ (aláhúzás)

Érvényes azonosítók:
- nem nyelvi kulcsszó
- betűvel vagy aláhúzással kezdődik
- max. 31 karakter hosszú

A program a kis- és nagybetűket megkülönbözteti az azonosítókban.

Nyelvi kulcsszavak

Kis- és nagybetűvel is írható (case insensitive)

CASE	GOTO	STATE
DECLARATIONS	IF	STATE_DIAGRAM
DEVICE	IN (obs)	TEST_VECTORS
ELSE	ISTYPE	THEN
ENABLE(obs)	LIBRARY	TITLE
END	MACRO	TRACE
ENDCASE	MODULE	TRUTH_TABLE
ENDWITH	NODE	WHEN
FUSES	OPTIONS	WITH
EQUATIONS	PIN	FLAG(obs)
PROPERTY		

Nyelvi kulcsszó nem lehet azonosító!

Számrendszerek

Az ABEL 2, 8, 10 és 16 alapú számrendszereket kezel, az alapérték általában a decimális. (A default a @RADIX deklarációval megváltoztatható.) A számrendszerek jelölését a példák mutatják:

75	decimális 75
^h75	hexa 75 (decimális értéke 117)
^b101	bináris 101 (decimális értéke 5)
^o17	oktális 17 (decimális értéke 15)
^h0f	hexa 0F (decimális értéke 15)

A karaktereknek is tulajdoníthatunk számértéket, pl. 'a'=97.

Logikai értékek

A true (logikai 1) és false (logikai 0) logikai értékeket a program számként ábrázolja, a true=1, a false=0.

Speciális konstansok

Érzékeny a betűnagyságra (case sensitive)

.C.	órajel impulzus low-high-low
.K.	órajel impulzus high-low-high
.U.	órajel felfutó él low-high transition
.D.	órajel lefutó él high-low transition
.F.	lebegő be- vagy kimenet
.P.	regiszter előtöltés (preload)
.SVn.	n=2..9 a bemenet meghajtható 2..9 V-al
.X.	don't care
.Z.	be- vagy kimenet nagy impedanciás

Stringek

A string két aposztróf (') közé zárt ASCII karakterek sorozata, space-t is tartalmazhat. Ha aposztróf szükséges a stringbe, akkor backslash (\) karaktert kell elé írni, ha backslash szükséges, azt kétszer kell leírni.

Pl. 'It\'s easy' = It's easy
'He\\she' = He\she

Logikai operátorok

!	! A	NOT	
&	A & B	AND	
#	A # B	OR	
$	A $ B	XOR	(antivalencia)
!$	A !$ B	XNOR	(ekvivalencia)

Aritmetikai operátorok

-	- A	kettes komplemens (negálás)
-	A - B	kivonás
+	A + B	összeadás
*	A * B	szorzás
/	A / B	előjel nélküli egész osztás
%	A % B	osztási maradék
<<	A << B	A-t balra shifteli B bittel
>>	A >> B	A-t jobbra shifteli B bittel (Mindkét shiftelésnél a szélről 0 töltődik)

Relációs operátorok

A relációs kifejezések eredménye logikai true vagy false

==	A == B	egyenlő
!=	A != B	nem egyenlő
<	A < B	kisebb
<=	A <= B	kisebb egyenlő
>	A > B	nagyobb
>=	A >= B	nagyobb egyenlő

A relációs kifejezéseket ajánlatos zárójelbe tenni, hogy a logikai operátorok magasabb prioritása ne zavarjon.

A relációs műveletek előjel nélküliek. Ezért pl. a (-1>2) true (1) értéket ad, mivel a -1 ábrázolása ..1111, és ez nagyobb mint ..0010 .

Értékadó operátorok

=	kombinációs hálózat esetén
:=	következő órajel után érvényes (regiszteres) érték (szekvenciális hálózat esetén)

Műveletek prioritása

1: - (kettes komplemens negálás), ! (NOT)

2: & (AND), << , >> (Shift left, right), *, /, % (szorzás, osztás, osztási maradék)

3: +, - (összeadás, kivonás), #, $, !$ (OR, XOR, XNOR)

4: ==, != (egyenlő, nem egyenlő), <, <=, >, >=

Halmazok (jelcsoportok)

A jelcsoportot alkotó jelekre és konstansokra közös azonosítóval lehet hivatkozni. A halmaz elemei [és] zárójelek közé kerülnek, egymástól vesszővel elválasztva, vagy csak a két szélső sorszámú elem egymástól a ".." range operátorral elválasztva. Példa a halmaz deklarációra:

Addr = [A15,A14,A13] vagy Addr = [A15..A13]

A szorzás (*), osztás (/), osztás-maradék (%) és shift (<<, >>) operátorok kivételével a többi operátor a halmazokra is értelmezve van.

Halmaz változóknak megfelelő hosszúságú halmazok vagy bitsorozatok (számok) adhatók értékül. Számok esetén azok bináris reprezentációját veszi, és ilyenkor a halmaz legnagyobb helyiértékű bitje a baloldali bit a deklarációban.

Blokkok

A blokk az ASCII szöveg egy szekciója, melyet a { és } zárójelek határolják. A blokkokat makrókban és direktívákban használják. A blokkokban használhatók a { és } jelek, ha backslash előzi meg. A blokkon belül lehetnek további blokkok.

Paraméterek és látszólagos paraméterek

A látszólagos paraméternek (dummy argument) nincsen értéke, csak a makrókban, module-ban vagy a direktivákban jelöli, hogy melyik paraméterrel végez műveletet. A makrodeklaráció törzsében a látszólagos paraméterek jele elé egy kérdőjelet (?) kell írni, és a kérdőjeles paramétert szóközökkel kell határolni.

Pl. OR_EM MACRO (a,b,c)
{ ?a # ?b # ?c }; (Ez a makrodeklaráció)
A definiált makro hívása valós paraméterekkel:
D = OR_EM (x,y,z&1)
Értékadás makro nélkül, ill. a makro kifejtése:
D = x # y # z&1

1.2. Funkcióleírás ABEL-HDL nyelven

Az ABEL-HDL forrás file felépítése

Az ABEL-HDL forrás file egy vagy több modult tartalmazhat. Minden modul egy komplett logikai (funkcionális) leírást tartalmaz.

Egy modul öt szekcióból áll, mely szekciók a következők:

- Fejléc
- Deklarációk
- Logikai leírás
- Teszt vektorok
- Vége

A fejléc szekcióban a következő elemek vannak:

MODULE (modul azonosítója és kezdete)

szintakszis:

MODULE modulnév
OPTIONS (program futását vezérlő opcionális elem)

TITLE (opc., kiírandó fejléc a dokumentációs és jedec file-okban)

szintakszisa:

TITLE 'string'

A fejlécben az utasításoknak a fenti sorrendben kell következniük.

A deklarációs szekció következő elemekből állhat

DECLARATIONS	kulcsszó
DEVICE	(eszközdeklaráció)
PIN	(kivezetések bekötése)
NODE	(belső csomópontok)
ISTYPE	(pin és node attribútum specifikáció, opc.)
konstansok	
MACRO	(makró(k))
LIBRARY REFERENCES	(könyvtári hivatkozások)

(Egy objektum első előfordulásához képest az összes rá vonatkozó deklarációnak predefinitnek kell lennie.)

Logikai leírás szekció, mely a feladat funkcionális, strukturális leírása (Argumentumaikban a korábban deklarált elemekre hivatkozhatunk.)

EQUATIONS	(Boolean kifejezések)
TRUTH_TABLE	(igazság táblák)
STATE_DIAGRAM	(idődiagramok)
FUSES	("biztositék"-deklarációk)
XOR_FACTORS	(XOR kapukat tartalmazó eszközök esetén)

Teszt vektor szekció

TEST_VECTOR	(teszt vektorok)
TRACE	(opc., szimuláció display vezérlése)

Vége szekció, mely a modul végét jelző END utasításból áll.

szintakszisa:

END modulnév

A deklarációs szekció

Eszközdeklaráció

Megadható, hogy milyen típusú PLD eszközzel történjen a feladat realizációja. Mivel az ABEL rendszer eszközfüggetlen feladat leírást is támogat, az eszközdeklaráció opcionális. Ez utóbbi esetben a PLD eszköz kiválasztása a tervezés végén történik, az ún. fitter programmal.

Az eszközdeklaráció szintakszisa:

eszközazonosító DEVICE eszköz_típusjel

Példa: ic2 DEVICE p16r6

Jelek deklarációja

A jelnevek szintakszisa: [!]jelazonosító[.ext]
A jelazonosító opcionális kiterjesztése, a .ext rész, az ún. dot extension az áramkör egyértelműbb leírását segíti elő.

A jelek deklarációja a PIN és NODE kulcsszavakkal történik. A PLD eszköz kivezetéseinek jeleit a PIN, a belső csomópontok jeleit a NODE kulcsszóval kell deklarálni.
A szintakszis:

jelnév [,jelnév...] PIN [pin# [,pin#...]] [ISTYPE 'attr'];
jelnév [,jelnév...] NODE [node# [,node#...]] [ISTYPE 'attr'];

ahol pin# ill. node# a kivezetés ill. csomópont sorszáma, 'attr' pedig a kivezetés ill. csomópont típusának deklarációja.

Dot extensions

Architektura-független dot extensions

.clk	élvezérelt flip-flop órajel bemenete
.oe	három állapotú kimenet engedélyezése
.pin	visszacsatolás a kivezetésről (pin-ről)
.fb	visszacsatolás a regiszterről

Architektura-specifikus dot extensions

.d	D flip-flop adatbemenete
.j	JK flip-flop J bemenete
.k	JK flip-flop K bemenete
.s	RS flip-flop S bemenete
.r	RS flip-flop R bemenete
.t	T (toggle) flip-flop T bemenete
.q	visszacsatolás flip-flop Q kimenetéről
.pr	regiszter preset bemenete (szinkron vagy aszinkron)

.re	regiszter reset bemenete (szinkron vagy aszinkron)
.ap	aszinkron preset bemenet
.ar	aszinkron reset bemenet
.sp	szinkron preset bemenet
.sr	szinkron reset bemenet
.le	latch engedélyező (enable) bemenete, aktív low
.lh	latch engedélyező (enable) bemenete, aktív high
.ce	órajel engedélyező bemenet
.fc	flip-flop mode control

Jel-attributumok

Szintakszis: jelnév [,jelnév] ISTYPE 'attr'

Ha a jelek a deklarált típusú PLD IC adott sorszámú kivezetéseihez vagy belső pontjaihoz vannak rendelve a PIN vagy NODE deklarációval, akkor ez a jelek típusát is egyértelműen meghatározza, és a típusmeghatározó attribútumok megadására nincs szükség.

Az általános attribútumok

'com'	"kombinációs" vagy közönséges jel, nem regiszter kimenet,
'reg'	regiszteres kimenet D típusú flip-floppal, a normalizálásnál figyelembe véve a cél-eszköz esetleges inverterét,
'neg'	a jelzett bemenet vagy kimenet negált kialakítású, a reduce-fixed opció esetén erre optimalizál,
'pos'	a jelzett bemenet vagy kimenet nincs invertálva, a reduce-fixed opció esetén erre optimalizál.

Architektúrát meghatározó (architecture dependent) attribútumok

'reg_d' regiszteres kimenet D típusú flip-floppal, és a kimenet polaritását meg kell határozni a 'buffer' vagy 'invert' attribútumokkal.

'reg_g' regiszteres kimenet D típusú flip-floppal, és a kimenet polaritását meg kell határozni a 'buffer' vagy 'invert' attribútumokkal. A logikai egyenletekben és igazságtáblákban .d és .ce kiterjesztésű jeleket kell használni.

'reg_jk' regiszteres kimenet JK típusú flip-floppal, és a kimenet polaritását meg kell határozni a 'buffer' vagy 'invert' attribútumokkal. A logikai egyenletekben és igazságtáblákban .j és .k kiterjesztésű jeleket kell használni.

'reg_sr' regiszteres kimenet SR típusú flip-floppal, és a kimenet polaritását meg kell határozni a 'buffer' vagy 'invert' attribútumokkal. A logikai egyenletekben és igazságtáblákban .s és .r kiterjesztésű jeleket kell használni.

'reg_t' regiszteres kimenet T típusú flip-floppal, és a kimenet polaritását meg kell határozni a 'buffer' vagy 'invert' attribútumokkal. A logikai egyenletekben és igazságtáblákban .t kiterjesztésű jeleket kell használni.
'buffer' a megfelelő flip-flop és a kivezetés (pin) közt nincs inverter
'invert' a megfelelő flip-flop és a kivezetés (pin) közt inverter van
'xor' a jelet egy XOR kapu állítja elő, mely két bemenetét egy-egy olyan logikai hálózat vezérli, mely a szokásos módon szorzattagok összegét képzi.

Logikai leírás szekció

Kombinációs hálózatok leírása

A kombinációs hálózatok viselkedése logikai egyenletekkel vagy igazságtáblával írható le. A **logikai egyenleteket** az EQUATIONS kulcsszó vezeti be, az egyenletek szintakszisa az alábbi:

```
[WHEN condition THEN] equation;
[ELSE equation];
```

Példák:

```
EQUATIONS
    A = B & !C # D ;
    regaddress = base + 3 ;
    !F = (B == C) ;
    WHEN B THEN A = B ;  ELSE A = C;
```

Az **igazságtáblák** szintakszisa:

```
TRUTH_TABLE  (bemenőjelek   -> kimenőjelek)
        bemenő értékek -> kimenő értékek;
        bemenő értékek -> kimenő értékek;
```

Példa:

```
TRUTH_TABLE     (BCD_code -> [a, b, c, d, e])
                    0     -> [1, 1, 1, 1, 1];
                    1     -> [0, 1, 1, 0, 0];
                    2     -> [1, 1, 0, 1, 1];
```

A példában a BCD_code és [a,b,c,d,e] egy set (halmaz).

Szekvenciális hálózatok leírása

A szekvenciális hálózatok viselkedése logikai egyenletekkel, igazságtáblával vagy állapotgráffal írható le.

A logikai egyenletek szintakszisa hasonló a kombinációs hálózat esetéhez, de az értékadó operátor formája := . Az egyenlet baloldali változója az órajel után veszi fel a := értékadó operátor utáni kifejezés értékét.

Példák:
```
EQUATIONS
        Q := D & !RESET ;
        count := count + 1 ;
```

Az igazságtábla (működési tábla) szintakszisa hasonló a kombinációs hálózat esetéhez, de itt a bemenő jelek és a belső regiszterek jelei közé a :> operátor kerül.
```
TRUTH_TABLE          (bemenőjelek  :>  reg.jelek   [ -> kimenőjelek])
                     bemenő értékek :> reg.értékek [ -> kimenő értékek] ;
                     bemenő értékek :> reg.értékek [ -> kimenő értékek] ;
```

Az állapotgráf leírásának szintakszisa

```
STATE_DIAGRAM  reg.jelek  [ -> kimenőjelek]
        STATE state_azonosító : [equation]  transition_utasitás ;
        [STATE state_azonosító : [equation] transition_utasitás ;]
```

Az állapot átmeneti (transition) utasítások az
IF-THEN-ELSE,
CASE és
GOTO utasítások, a szokásos jelentésükkel. Ezeket az utasításokat opcionálisan követheti a WITH-ENDWITH átmenet-utasítás.

Példák:
```
STATE_DIAGRAM  [Qa, Qb]
        STATE S0 :
                        IF Enable THEN S1
                        ELSE S0 ;
        STATE S1 : z = 0 ;
                        CASE  (Enable = = 0) : S1 ;
                                (Enable = = 1) : S2 ;
                        ENDCASE
        STATE S2 : z = 1 ;
                        GOTO S0;
```

A WITH-ENDWITH utasítás az IF-THEN-ELSE, CASE vagy GOTO átmenet utasítás után használható, kimenet viselkedésének átmenet segítségével történő leírására.

Szintakszisa:

transition_utasitás következő_állapot WITH equation
[equation] ...
ENDWITH ;

Pl.: IF a==1 THEN S1 WITH x := 1 ;
y := 0 ;
ENDWITH ;

Nem teljesen specifikált bemenet ill. kimenet

A nem teljesen specifikált megadás esetén van szerepe a @DCSET direktívának. A direktíva hiányában az ABEL program a nem specifikált esetekhez tartozó függvényértéket 0-nak veszi, ha a kimenet nincs invertálva, és ha a kimenet invertálva van, akkor 1 értéket rendel a nem specifikált esetekhez. (Az ABEL régebbi verziói mindig így működtek.)

Ha a @DCSET direktíva szerepel a leírásban, akkor utána a nem specifikált esetekben a függvényértéknél valódi don't care értékekkel számol a program.

Az @ONSET direktíva megszünteti a don't care értékek használatát.

Teszt vektorok

A teszt vektorok a tervezett logikai hálózat helyes működését írják le. Szerepük a terv szimulációval történő ellenőrzésénél és a beprogramozott PLD IC ellenőrzésénél van.

Az ABEL szimulációs programja a kimenetek kiszámított állapotát összeveti a teszt vektorokban megadott állapotokkal, és eltérés esetén hibát jelez.

A teszt vektorok a programozó készüléket vezérlő file-ba (.jed) is átkerülnek, és a programozó készülék a "beégetés" után ezek ráadásával ellenőrzi a beprogramozott PLD IC működését.

A teszt vektorok megadása a TEST_VECTORS kulcsszóval, a következő szintakszisnak megfelelően történik.

TEST_VECTORS ['megjegyzés string']
(bemenőjelek -> kimenőjelek)
[bemeneti értékek -> kimeneti értékek ;]
[bemeneti értékek -> kimeneti értékek ;]

Példa:

```
TEST_VECTORS  ([A,B] -> [C,D])
                [0,0] -> [1,1] ;
                [0,1] ->  2  ;
                [1,0] -> [0,1] ;
                [1,1] -> [0,0] ;
```

A teszt vektorok több részletben, eltérő "megjegyzés"-sel is megadhatók. Kimeneti értékként a don't care (.X.) is megadható.

A szimulációs program által kiírandó/kijelzendő jelek a teszt vektor fejlécétől eltérőek is lehetnek, a kijelölésük a TRACE utasítással történhet.

TRACE (bemenő jelek -> kimenő jelek);

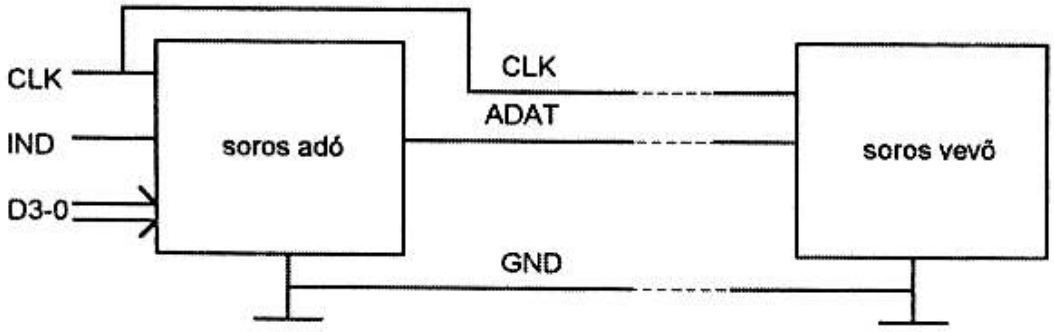
Példa:

```
TEST_VECTORS    ([A,B] -> [C,D])
                  [0,0] -> [1,1] ;
                  [0,1] ->  2  ;

TRACE           ([A,B] -> [C, D]);
                  [1,0] ->  [0,1] ;
                  [1,1] ->  [0,0] ;
```

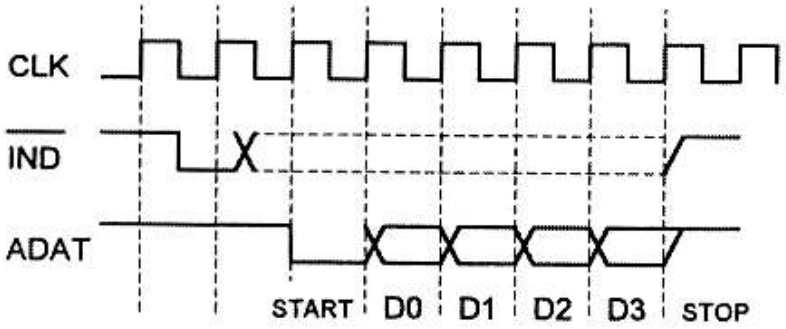
2. Mintapélda ABEL nyelvre, szinkron soros adó tervezése

Az alábbiakban egy 4 bites szinkron soros adó ABEL nyelvű megadását mutatjuk be. A szinkron soros adatátvitel célja, hogy kevés vezetéken és gyorsan juttassuk el az információt egyik helyről a másikra. A feladatban megvalósított adó esetén a földvezetéken kívül (GND, 0V feszültségű pont) 2 vezeték szükséges az adatátvitelhez, egy órajel és egy adatvezeték (1. ábra) .



1. ábra Szinkron soros adó

Az átvitel a 2. ábra idődiagramja szerint történik:



2. ábra Soros adó idődiagramja

Az adatvezeték alapállapotban magas szinten van. Az adó az indító jelet (IND) követően 1 órajelnyi időnként a kimenetre adja az egyes biteket. Az első bit a START bit, ami kötelezően 0 szint. Ez jelzi a vevőnek, hogy adatátvitel kezdődik, ugyanis a teljes adatátvitel befejezése után a kimenet azonnal 1 szintre áll. A következő 4 bit az adatbitek, D0-D3 sorrendben. Az adó 4 adat bemenetén az adatot nem szabad változtatni az átvitel ideje alatt. Az adás a stop bittel fejeződik be, ami legalább 1 órajelnyi ideig 1 értékű. Újabb indítás csak újabb IND impulzus hatására következik be. Az IND impulzus legalább egy órajel hosszú, különben nem garantált az érzékelése.

A feladatot egy GAL 16V8-al oldjuk meg. A működéseket állapotgráffal írtuk le. Az állapotgráf leírásában azt adjuk meg, hogy egy-egy állapotból melyek lesznek a következő állapotok a megadott feltételek teljesülése esetén és mi a kimenet. A konkrét feladatban a kimeneti értékadások regiszterekre vonatkoznak, ezért ezek csak *a következő órajel hatására hajtódnak végre.* A működést végig lehet követni a

tesztvektorok segítségével. A tesztvektorokkal a valódi működést próbáljuk szimulálni, leírjuk, hogy adott bemenet (gerjesztés) hatására milyen kimenetet várunk.

Az állapotgráf S0 állapotában a rendszer az IND jelre vár. Ha ez megjött, akkor minden egyes újabb órajelre egy következő állapotba lép (számolja hogy hányadik bit kiadásánál tart), miközben a megfelelő sorrendben a kimeneti regiszterbe másolja a kiadandó biteket. A végén az IND gomb elengedésére vár (gyors órajel esetén egyébként egy nyomógomb megnyomás alatt többször is megismételné az adat sorossá alakítását).

A szinkron soros adó ABEL nyelvű leírása

```
module  sou
title  'szinkron soros adó'

        sou  device 'p16v8'; "GAL16V8

"kimenetek:
        Q0  pin 19 ISTYPE 'reg';      "soros kimenet
        ST2 pin 14 ISTYPE 'reg';      "vezerlő állapot bitek
        ST1 pin 13 ISTYPE 'reg';
        ST0 pin 12 ISTYPE 'reg';
"Az 1-es láb sorrendi hálózat esetén kötelezően órajel bemenet


"bemenetek:
        CLOCK       pin 1;          "dedikált órajel bemenet
        !IND        pin 2;          "!: alacsony aktív
        D0,D1,D2,D3 pin 3,4,5,6;    "adat bemenetek
        OE          pin 11          "kimenet engedélyezés


" Speciális konstansok
        C, x, Z  = .C. , .X., .Z.;  "clock, don't care, 3-state


" állapot definíciók
        S  = [ST2,ST1,ST0];
        S0 = [   0,   0,   0];          "bekapcsolási kezdő állapot
        S1 = [   0,   0,   1];
        S2 = [   0,   1,   0];
        S3 = [   0,   1,   1];
        S4 = [   1,   0,   0];
        S5 = [   1,   0,   1];
        S6 = [   1,   1,   0];
        S7 = [   1,   1,   1];
```

Irodalom jegyzék

[1] Selényi E.-Benesóczky Z.: Digitális technika példatár, Műegyetemi kiadó, J5-5005, 1991

[2] John F Wakerly: Digital Design, Prentice Hall, Englewood Cliff, New Jersey, 1990

[3] TTL Advanced Low-Power Schottky, Advanced Schottky Data Book, 1991

[4] Pak K.-Samiha Mouard: Digital Design Using Programmable Gate Arrays, Prentice Hall, ISBN 0-13-319021-8

[5] XILINX Programmable Logic Data Book 1994

[6] ACTEL FPGA Data Book and Design Guide (1994)

[7] Lattice Data Book (1994)

[8] ATMEL 8051 Architecture Data Sheets (www.atmel.com/atmel/products)

[9] S.D.Brown-R.J.Francis-J.RoseZ- G.Vranesic: Field Programmable Gate Arrays

[10] Gerdai Gábor: Programozható logikai áramkörök

Ez az egyetemi tankönyv elsősorban a BME villamosmérnöki és Informatikai Karon oktatott Digitális technika tantárgy II. félévi anyagának elsajátításához kíván segítséget nyújtani, azonban ajánljuk mindenkinek, aki érdeklődik a digitális technika ill. az egyszerűbb mikroprocesszoros rendszerek tervezése iránt.

A könyv felépítésében az előadás anyagát követjük, sok helyen tervezési mintapéldákkal illusztráljuk az elmondottakat.

Az első fejezetben a kombinációs és sorrendi funkcionális elemeket ismertetjük.

A második fejezet a programozható logikákkal foglalkozik, a szakirányú digitális képzésre is gondolva, az átlagosnál nagyobb mélységben.

A harmadik fejezetben a funkcionális elemekkel való strukturált tervezést (adatstruktúra-vezérlő felbontás, mikroprogramozott vezérlő terezés) taglaljuk.

A negyedik fejezet az egyszerűbb mikroprocesszorok belső felépítését, működését és a mikroprocesszoros rendszer kialakítását (memória és periféria illesztés) mutatja be.

Az ötödik rész a mikrokontrollerekkel foglalkozik, elsősorban a széles körben elterjedt MCS8051 családra koncentrálva.

ISBN 963 420 756 3
55033
www.kiado.bme.hu