VI. modul: Összetett digitális egységek

1. lecke: Multiplexerek-demultiplexerek, regiszterek, funkciós egységek

Cél:

A lecke célja, hogy a hallgató megismerje az összetett digitális egységek alapvető csoportjait, a csoportok egyes elemeinek felépítését, működését valamint a digitális hálózatokban történő alkalmazásainak módjait.

Követelmények:

Ön akkor sajátította el megfelelően a tananyagot, ha

- ismeri az összetett digitális egységek alapvető típusait, azok felépítését és működését, digitális architektúrákban betöltött szerepüket és alkalmazásuk lehetőségeit.

Időszükséglet:

A tananyag elsajátításához kb. 180 percre van szükség.

Kulcsfogalmak:

- MPX, DMPX,
- lebegő (3.) szint, átvivő kapu
- regiszter, FIFO, LIFO, RAM
- kettes komplemens
- univerzális összeadó-kivonó
- időzítő-vezérlő, FSM

Tevékenység: tanulmányozza a leckében bemutatott összetett digitális egységek felépítését, működési elvét. A tanulás során ismételje át a hivatkozásra kerülő, a korábbi leckékben már bemutatott egységek jellegzetességeit.

1. Összetett digitális egységek

Ebben a fejezetben az előzőekben már megismert egyszerű kapuszintű logikai hálózati elemek segítségével felépített összetett digitális egységek alapelemeit ismerhetjük meg. Ezek a magasabb szintű egységek – a digitális hálózatokban betöltött funkciójuk szerint – három alapcsoportban sorolhatók:

- *multiplexerek, demultiplexerek*
- *regiszterek*
- *funkciós egységek*

A multiplexerekkel már egy korábbi fejezetben megismerkedtünk: ezek a hálózati elemek a demultiplexerekkel együtt adatút szakaszok kijelölésére szolgálnak.

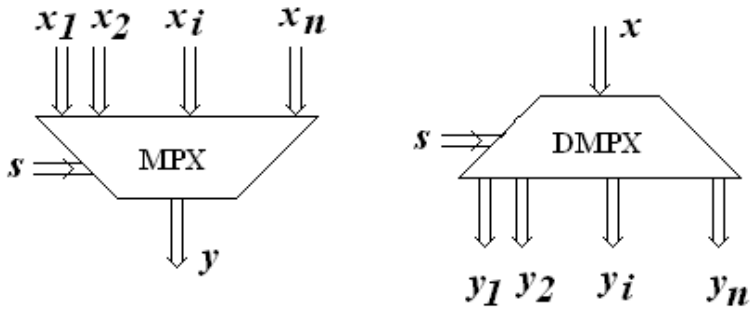
A regiszterek alapelemeivel, a tárolókkal szintén foglalkoztunk már: ezek adatokat tárolnak, és ezek elérését biztosítják.

A funkciós egységek alkalmazása egy digitális architektúrában adatok közötti műveletek elvégzését teszi lehetővé.

1.1 Multiplexerek, demultiplexerek

A multiplexerek és a demultiplexerek, mint adatút kijelölő egységek kerülnek felhasználásra a digitális hálózatokban. Felépítésük és funkciójuk alapján a kombinációs hálózatok csoportjába tartoznak, ezért már az első fejezetben pl. a multiplexerekkel részletesebben is foglalkoztunk, sőt a számláló bázisú szinkron sorrendi hálózatok tervezésekor, mint programozható logikai hálózati elemet is felhasználtuk egy speciális célarchitektúra építőelemeként.

Ismétlésként elevenítsük fel röviden a multiplexerek és demultiplexerek működésének és felépítésének lényegét: multiplexereknél(MPX) több forrás közül (x_1, x_2,.. x_n) kijelöljük azt, amelynek adata a kimenetre (y) kerül, míg a demultiplexerek(DMPX) esetén azt a kimenetet (y_1, y_2,.. y_n) címezzük meg, amelyen az egyetlen forrás (x) adatát meg kívánjuk jeleníteni. Az adatutak kijelölése vezérlőbemenetek (s) segítségével, címzéssel történik (1.1.a ábra):

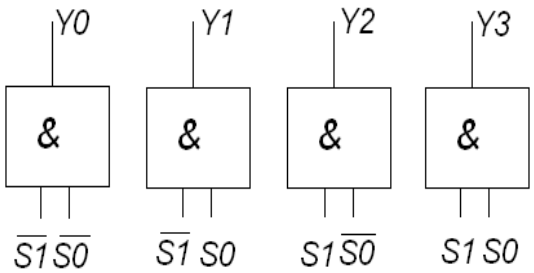


1.1.a ábra[1]

A multiplexerek és demultiplexerek szokásos szimbólum jelölése

Korábbi tanulmányaink során megismerhettük, hogyan lehet a multiplexert programozható logikai hálózati elemként alkalmazni: a mintermes kanonikus alakban megadott függvény mintermjeit a címző (vezérlő) bemenetekre adott címek képviselik, és a megcímzett adatbemenetre rá kell kapcsolnunk az adott mintermhez tartozó logikai értéket. Ezeknek a logikai konstansoknak a bemenetekre való kapcsolását tekinthetjük a multiplexerek programozásának.

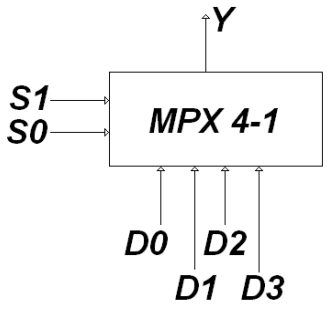
A demultiplexereket gyakran használják dekóderként: ha egy demultiplexer adatbemenetét állandó, logikai ’1’ szintre kapcsoljuk, akkor ez egyenértékű azzal, hogy a hálózatában szereplő ’ÉS’ kapuk bemenetei közül elhagyjuk az adatbemenetet. Az ilyen áramkör sajátossága, hogy a kiválasztó bemenetekre kapcsolt kombinációk csak egyetlen, a kiválasztó kódnak megfelelő kimeneten eredményeznek magas szintet. Az ilyen áramköröket nevezzük dekódereknek. Egy DMPX1-4 egység dekóderként történő felhasználására mutat egy realizációs példát a 1.1.b ábra:



1.1.b ábra[1]

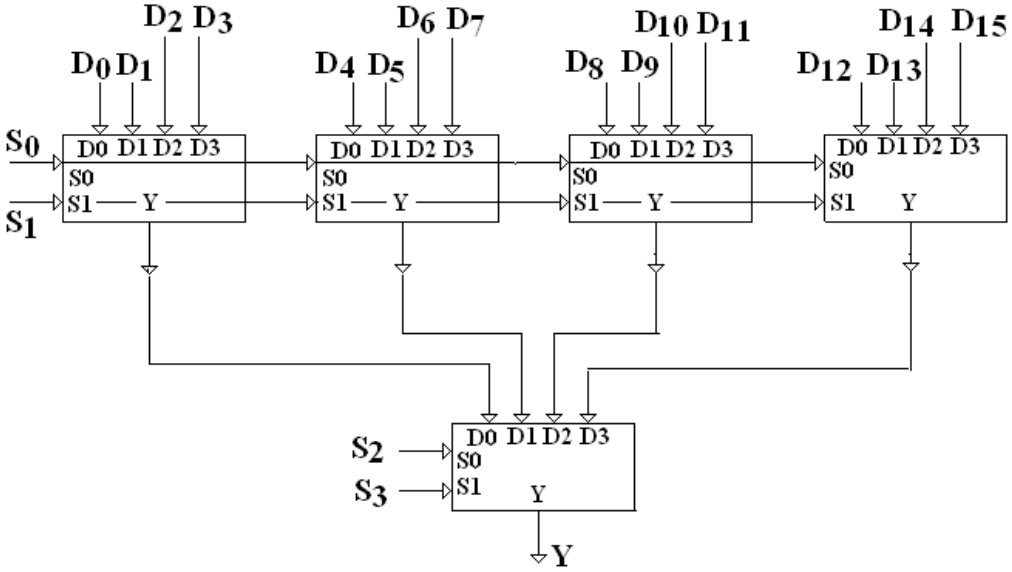
DMPX1-4 egységből kialakított kétbemenetű dekóder

Közepes integráltságú (MSI = Medium Scale Integration) áramkörökben gyakran használatos adatút választó eszköz az MPX4-1 multiplexer, mint alapegység (1.1.c ábra). Előfordul azonban, hogy esetenként a bemeneti adatunk négynél több bitből áll. Ebben az esetben célszerű a nagyszámban rendelkezésre álló alapegység felhasználásával elvégezni a bővítést. Erre mutat példát a 1.1.d ábra: itt MPX16-1 multiplexert alkottunk MPX4-1 egységekből. Látható, hogy nemcsak vízszintes irányban kellett bővíteni az egységet, hanem mélységében is: a felső sort „felfűző" címzőbemeneteken(s_0; s_1) megjelenő címvektor rész mind a négy egység bemenetei közül($D_0 - D_{15}$) ugyanazt a sorszámú bemenetet címzi meg, és a második sor egyetlen egységének címvektora(s_2; s_3) ebből a négy adatból választ ki egyet.



1.1.c ábra[1]

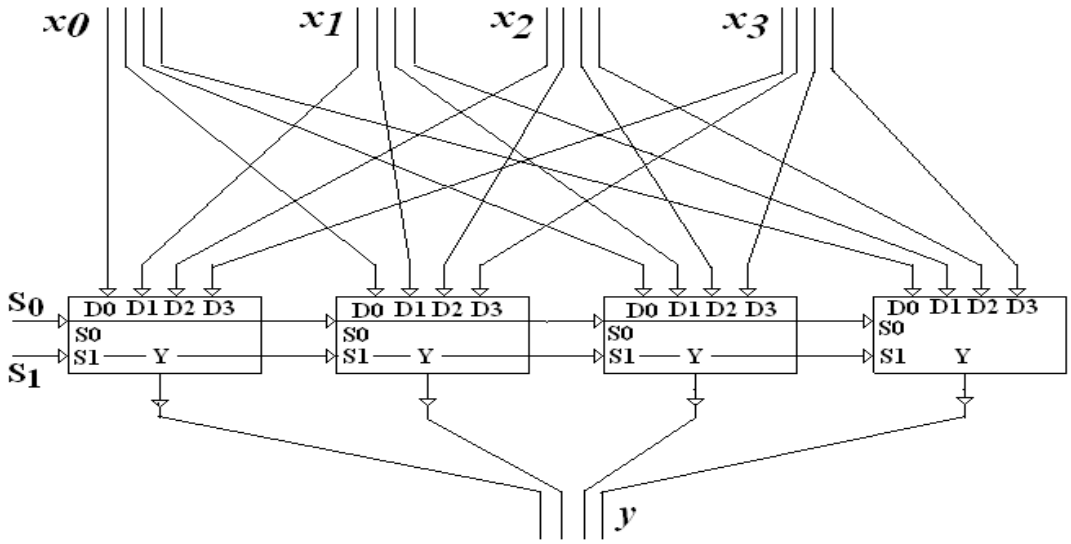
MPX4-1 egység



1.1.d ábra[1]

Bővítés a bemenetek számának növelésére MPX4-1 egységek felhasználásával

Amennyiben az a feladat, hogy 4 különböző adatbemenet($x_0 \dots x_3$) közül egynek az értékét kell egységesen továbbítani, akkor a sínek között választás céljából kialakított bővítésre mutat példát a 1.1.e ábra:



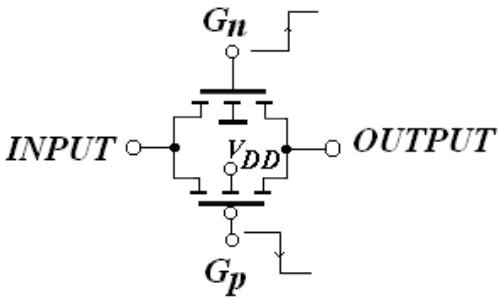
1.1.e ábra[1]

Négy, 4-bites sínből egyet kiválasztó multiplexer egység MPX4-1 alapegységekből

Az átvivő kapu (transmission gate) alkalmazása: a „lebegő szint" megvalósítása digitális hálózatokban

Eddigi tanulmányaink során egy hálózati struktúra adott pontján mindig kétféle logikai szintet határoztunk meg, illetve értelmeztünk: a magas ('1') és alacsony ('0') szinteket. Egy digitális rendszerben azonban sokszor szükség van az egyes hálózati részekben (elsősorban adatút átviteli pontokon) egy harmadik, ún. „lebegő" szint biztosítására. A harmadik logikai állapotot is lehetővé tevő kimenetek szerepe a modern logikai hálózatokban és a mikroprocesszoros rendszerekben meghatározó jelentőségű. Ha ugyanis garantálni tudjuk, hogy egyetlen közös pontra a kimenetével kapcsolódó több logikai elem közül *legfeljebb csak egy kimenete legyen „nem harmadik" állapotú*, akkor olyasmit tehetünk, amely *kétállapotú kimenettel rendelkező elemek esetén szigorúan tilos*: kimeneteket kapcsolhatunk össze.

Ezt a megvalósítást biztosítja a CMOS átvivő kapu (transmission-gate, transfer-gate), amely két MOSFET eszközből álló kapcsoló (1.1.f ábra.):



1.1.f ábra[1]

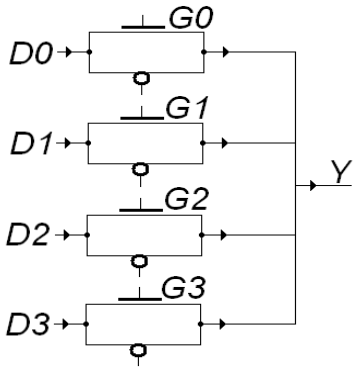
Az átvivőkapu (transmission gate)

A „transmission gate" (TG) egy *n*-MOSFET és egy *p*-MOSFET párhuzamos összekapcsolása. Ahhoz, hogy bekapcsoljuk, az *n*-MOSFET G_n elektródájára magas, a *p*-MOSFET G_p elektródájára alacsony szintet kell adni.

A tranzisztorok párban történő alkalmazásának a szükségességét a következők indokolják:

a logikai áramkörökben szokásos, úgynevezett 'egy tápfeszültséges' rendszerben a logikai alacsony szinthez a '0', vagy 'föld' (GND) potenciált, a logikai magas szinthez ('1') pedig a pozitív tápfeszültség értéket (V_{dd}) rendeljük. Megvizsgálhatjuk, hogy az *n*- illetve *p*-csatornás MOSFET hogyan viszik át egyik elektródájukról a másikra az egyes logikai szinteket. Azt fogjuk tapasztalni, hogy az *n*-MOSFET a logikai alacsony szintet jól átviszi, ezzel szemben a magas szintet csak „küszöbfeszültségnyi" hibával képes egyik oldaláról a másikra átjuttatni. (Küszöbfeszültségnek azt a feszültség értéket nevezzük, amelynek elérésénél(G) a tranzisztor „vezető" állapotba kerül, azaz „kinyit".) A *p*-csatornás MOSFET pedig éppen fordítva viselkedik, a magas szinttel boldogul hiba nélkül. Beláthatjuk, hogy a TG mind a magas, mind az alacsony szintet hibátlanul viszi át a bemenetről a kimenetre, hiszen egyik eszköz mindig kisegíti a másikat, és befejezi a folyamatot.

Egy átvivő kapukkal felépített MPX4-1 egységet láthatunk a 1.1.g ábrán:



1.1.g ábra[1]

MPX4-1 egység CMOS átvivő kapukkal

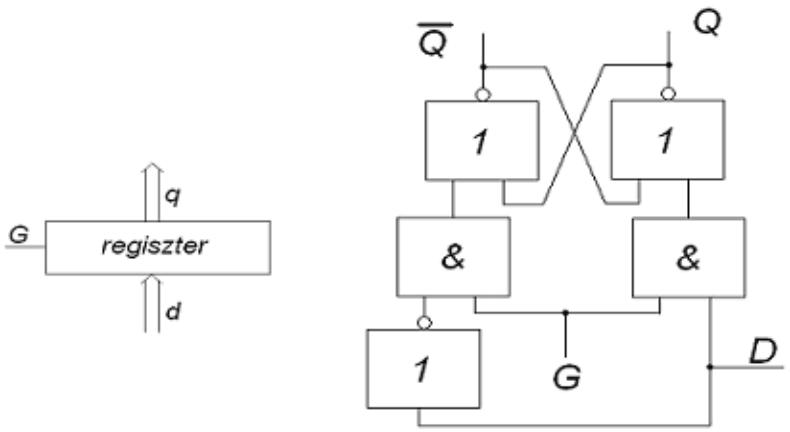
A vezérlőelektródákat bekapcsoláskor ellentétesen kell vezérelni, azaz a kis körökkel jelölt vezérlő bemenetek a G_0 - G_3 vezérlők negáltjai. Az átvivő kapukból felépített multiplexer jellegzetessége, hogy kimenete képes a logikai harmadik állapot felvételére, azaz ha egyik kapcsolót sem nyitjuk, a kimenet „lebeg”.

1.2 Regiszterek

A digitális hálózatokban a regiszterek adatokat tárolnak, és ezek elérését biztosítják. A különböző típusokat felépítésük és működésük alapján külön alfejezetekben ismertetjük.

1.2.1 Szintvezérelt statikus regiszter

A szintvezérelt statikus regiszter általános sémája, és DG tárolóból kialakított egybites cellájának kapuszintű realizációja 1.2.1.a ábrán látható:



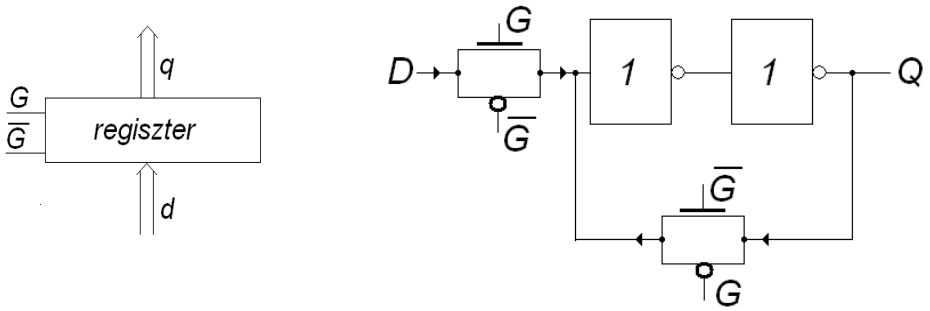
1.2.1.a ábra[1]

A szintvezérelt statikus regiszter általános szimbóluma, és DG tárolóból kialakított egybites cellájának kapuszintű realizációja

Erre a regiszterre egy bemeneti bitvektor (*d*) és egy logikai beírójel (*G*) csatlakozik. A regiszter a *G* beíró jel magas szintjére a *d* értékét a tárolóba írja. A regiszter átlátszó, azaz amíg a *G* jel magasan van, *d* változásai késleltetve megjelennek. *G* lefutása után az utolsó, még hatásos bemeneti érték marad a regiszterben.

1.2.2 Szintvezérelt statikus regiszter ponált és negált beíró jelekkel

A ponált és negált beíró jellel vezérelt CMOS regiszter szimbóluma és egybites cellájának kapuszintű struktúrája a 1.2.2.a ábrán látható:



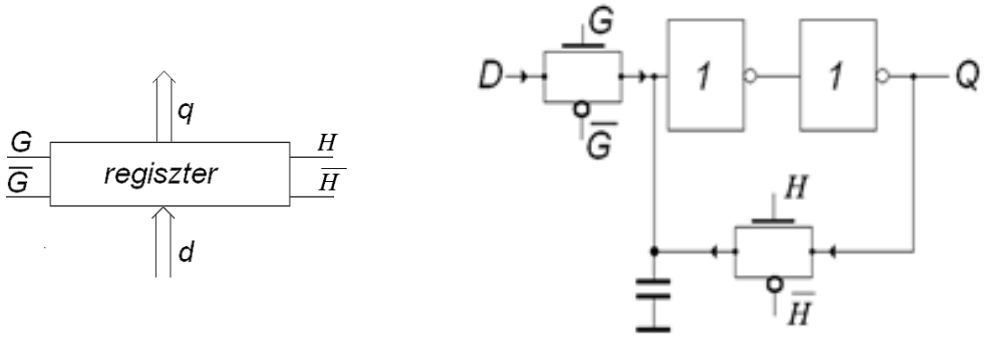
1.2.2.b ábra[1]

Ponált és negált beíró jellel vezérelt CMOS regiszter szimbóluma és egybites cellájának kapuszintű struktúrája

A CMOS technikában, különösen a VLSI (Very Large Scale Integration) áramkörökben előszeretettel alkalmazzák az átvivő kapus tárolókból álló statikus regisztert. Az egy bitnyi tároló ’(latch’) a két stabil állapotú (bistabil) invertergyűrű beírásának egy más módszerét alkalmazza, mint a NOR-bistabilok. A beírás (G=1) alatt a visszacsatolás meg van szakítva, hiszen a visszacsatoló átvivő kapu a G=0-nál van bekapcsolva. Ez a beírás után azonnal bekövetkezik, és megvalósul a tárolás. Ennek megfelelően a regiszter bemenetei között a G vezérlővezetéknek mind a ponált, mind a negált változata megjelenik.

1.2.3 Kvázistatikus regiszter

A kvázistatikus regiszter szimbólumát és egy-bitnyi NOR struktúráját a 1.2.3.a ábrán láthatjuk:



1.2.3.a ábra[1]

A kvázistatikus regiszter szimbóluma és egy bites cellájának struktúrája

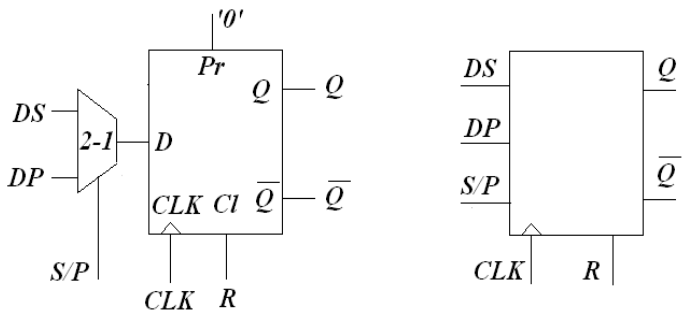
A kvázistatikus regiszter alapcellája (1-bites egysége) a CMOS inverter bemeneti kapacitásának átmeneti töltéstároló képességét használja ki. Itt a G beírójel két felfutása, azaz két beírás között egy tartó (*H, HOLD*) impulzus rendszeres jelentkezése szükséges. A H impulzusok között a bemeneti kapacitás tárolja az utoljára beírt szintet, a két inverter pedig regenerálja azt.

1.2.4 Élvezérelt regiszter

Az élvezérelt regiszterekben az átlátszóság a beírójel valamelyik éléhez kötődik. Ez lehet a beírójel felfutó éle, de lehet a lefutó él is. Az élvezérelt regiszterekből felépített digitális rendszer kevésbé érzékeny az órajelek időbeli elcsúszásából adódó aszinkronitásokra. Az élvezérlést a beíró jel bemenetre elhelyezett speciális szimbólum jelzi. Elfogadott, hogy a felfutó élre való átlátszóságot a beíró-jel ponált formája jelöli.

1.2.5 Soros elérésű tárolók

A soros elérésű memóriák alapeleme az élvezérlésű, ún. kétfázisú D-MS tároló. Ebből a tárolóból egy MPX2-1 multiplexer alkalmazásával olyan egységet kapunk, amelynek két adatbemenete közül (*DS, DP*) közül az *S/P* vezérlőjel szintjének egyike választ. A tároló *PRESET(Pr)* bemeneteit konstans logikai alacsony szintre kötjük, a *CLEAR(Cl)* bemeneteket ezzel szemben a kezdeti '0' állapot beállítására használni fogjuk. A 1.2.5.a ábrán látható egységet soros memóriák építőelemeként fogjuk felhasználni.

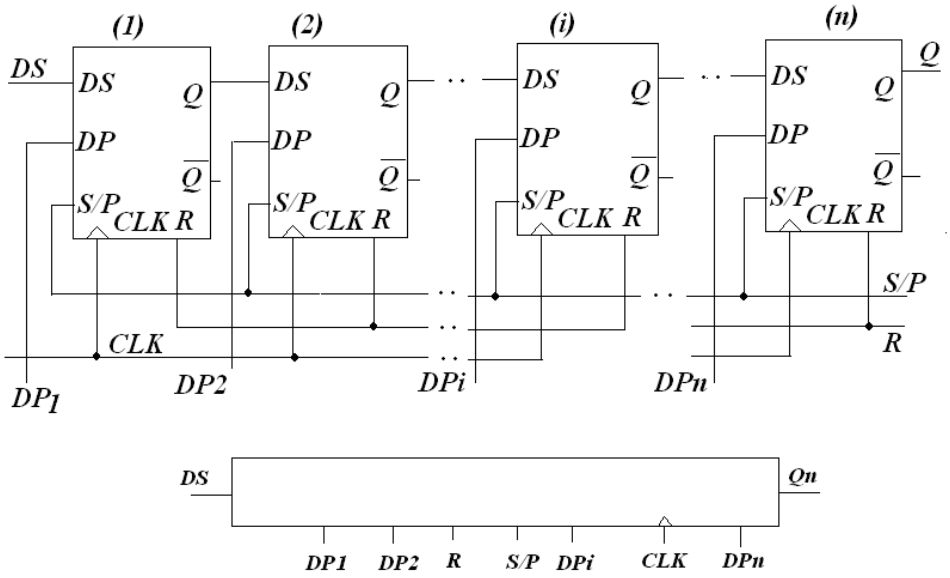


1.2.5.a ábra[1]

Az egybites soros memóriaelem MPX2-1 multiplexerrel(bal), és a sematikus szimbóluma(jobb)

1.2.5.1 Párhuzamosan is betölthető soros memóriák

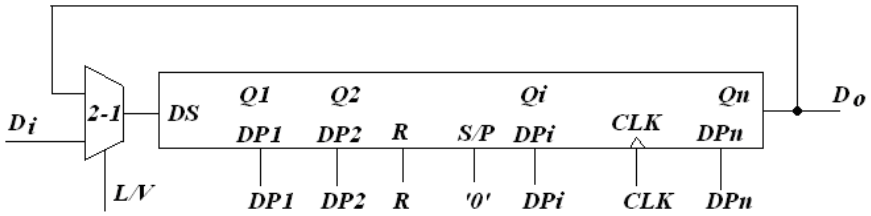
A 1.2.5.1.a ábrán egy nyitott, soros elérésű *n*-bites memóriablokk sémája látható. Az *S/P* vezérlőbemenet állapotától függően az órajelciklusra vagy balról-jobbra léptetés, vagy párhuzamos betöltés történik. Az egység az *R* jellel alapállapotba hozható.



1.2.5.1.a ábra[1]

Nyitott, soros elérésű 'n'-bites regiszter(felső kép), és általános sémája(alsó kép)

A 1.2.5.1.b ábra ennek az egységnek az a változata, amikor a memória tartalmát körbe forgatva bármely beírt adat elérhető a kimeneten, de egy adat elvesztése árán új adatot is betölthetünk a D_i bemenetről az *L/V* vezérlőbemenet segítségével. Ezt a megoldást alkalmazzák pl. az ún. gyűrűs számlálók kialakításánál. Egy *n* modulusú gyűrűs számlálónál az eredeti információ az *n*-dik lépés (ciklus) után kerül vissza a regiszter megfelelő helyiértékeire. A regisztereknek ezt a csoportját szokták felhasználni pl. soros működésű aritmetikai egység átmeneti tárolójaként, ha az egyik tényezőt - műveletvégzés után - változatlanul kell megtartani.

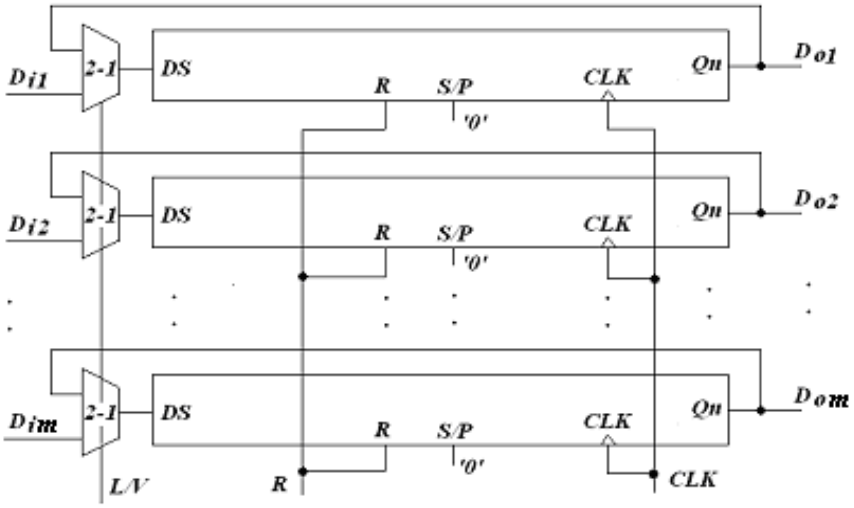


1.2.5.1.b ábra[1]

Párhuzamosan betölthető, sorosan rátölthető soros memória.

1.2.5.2 Szószervezésű soros memóriák

A megismert építőelemből 1-nél nagyobb, például *m* szószélességű soros memóriát építünk: elhagyjuk a párhuzamos beírás lehetőségét, azaz az *m* számú gyűrű *S/P* bemeneteit soros üzemmódra állítjuk be. A memória *L/V* vezérlővezetékével beállíthatjuk, hogy a memóriában lévő adatokat forgatjuk, vagy új adatot szúrunk be a régiek közé (1.2.5.2.a ábra):

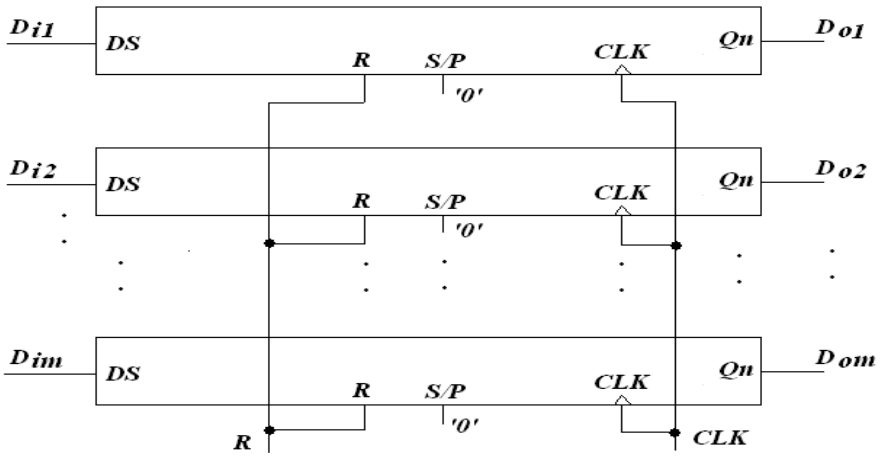


1.2.5.2.a ábra[1]

Egy m-bit szószélességű soros memória-egység

1.2.5.3 FIFO memóriák

A FIFO olyan soros memória, amelyből az az adat olvasható ki először, amelyet elsőként töltöttünk be (First In First Out). A fent bemutatott 1-bit szélességű párhuzamosan betölthető soros memória egységekből elvesszük a párhuzamos betöltés lehetőségét és *m* számú ilyen egységből *m*-bites szószélességű FIFO-t csinálunk, ahogyan azt a 1.2.5.3.a ábra is mutatja:

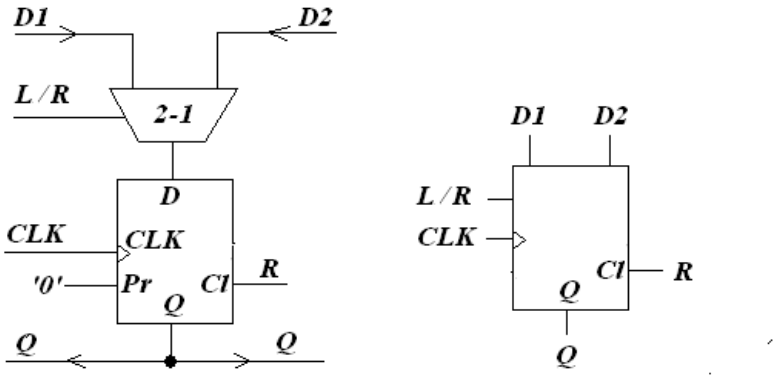


1.2.5.3.a ábra[1]

Egy m-bit szószélességű FIFO memória

1.2.5.4 LIFO memóriák

A LIFO memóriák alapelemeként szolgáló, MPX2-1 multiplexerrel kiegészített D-MS flip-flop sémája és szimbóluma a 1.2.5.4.a ábrán látható:

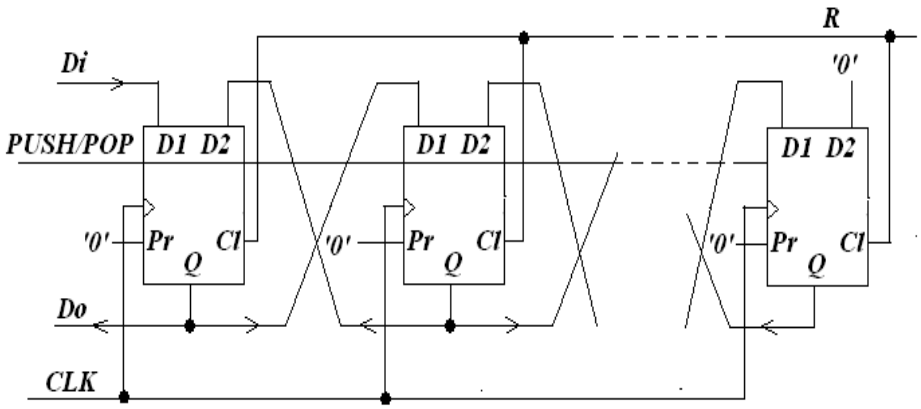


1.2.5.4.a ábra[1]

LIFO memória alapelem

A D-MS flip-flop kimenetén az *L/R* vezérlőjeltől függően vagy a baloldali D_1, vagy a jobboldali D_2 bemenet szintje jelenik meg. Ez a LIFO tárolók alapeleme. A LIFO olyan soros memória, amelyből az az adat olvasható ki először, amelyet utoljára töltöttünk be (Last In First Out). Két vezérlőbemenete van: betöltésre a 'betölt' (PUSH), kiolvasásra a 'kiugrat' (POP) vezérlőbemeneteket használjuk, természetesen egymás kizárásával. A LIFO egyetlen adatcsatlakozása tehát bemenet és kimenet szerepét is betölti.

A LIFO elemekből alkotott 1-bit szélességű LIFO memória a 1.2.5.4.b. ábrán látható. Ebből könnyen alkothatunk *m* szószélességű LIFO memóriát.

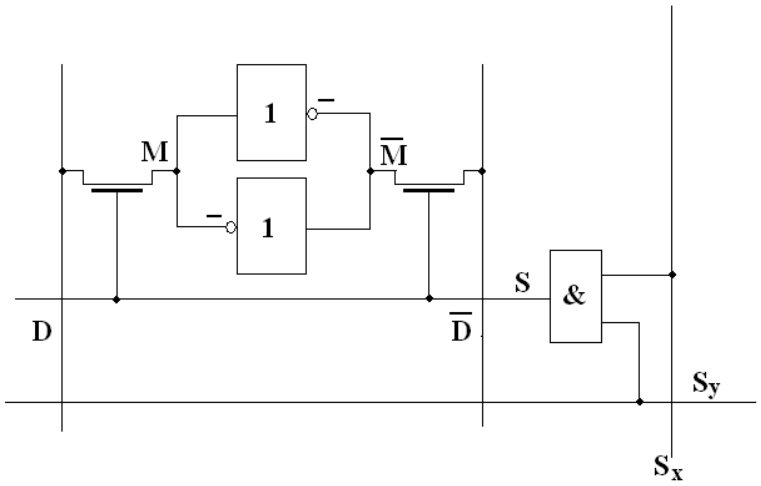


1.2.5.4.b ábra[1]

LIFO memória sor sémája

1.2.6 Párhuzamos hozzáférésű memóriák

Egy memóriablokk esetében a párhuzamos hozzáférés azt jelenti, hogy a memória minden egyes bitjéhez, vagy minden egyes szavához annak helyétől független elérési idővel férünk hozzá akár átírás(WRITE), akár kiolvasás(READ) szándékával. Ezeket az írható-olvasható memóriákat szokás RAM (Random Acccess Memory) egységeknek hívni. A RAM memóriák cellákból állnak. A RAM cellákban nemcsak a korábban említett harmadik állapot lehetőségét használjuk ki, de az azonos logikai szintek erőssége közötti különbségek lehetőségét is. Egy ilyen, 1-bites alapcellát láthatunk a 1.2.6.a ábrán:



1.2.6.a ábra[1]

1-bites RAM alapcella felépítése

A cella működését a következőképpen értelmezhetjük:

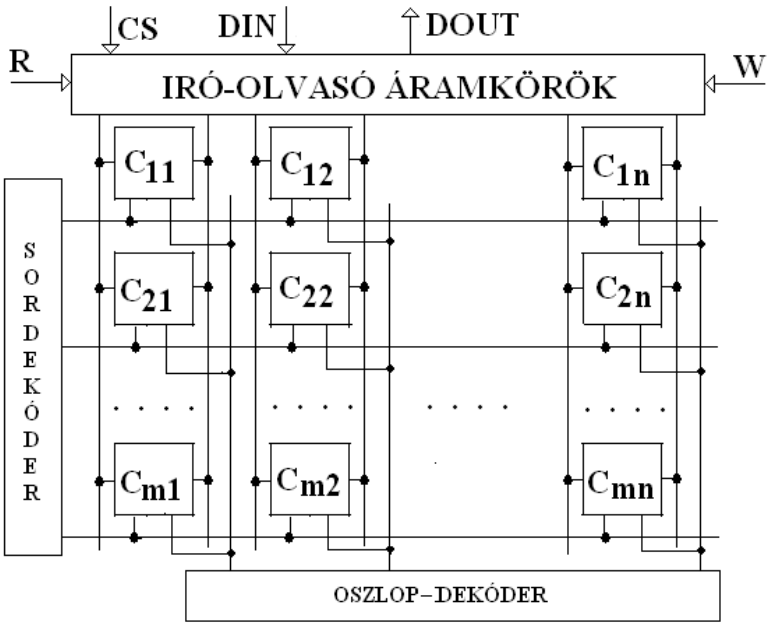
a realizációs rajzon az egymást ölelő inverterek bistabil cellát alkotnak. Ha a két elektronvezetésű MOSFET kapcsoló zárva van, azaz az *S* bemenet alacsony szintű, akkor az inveterek őrzik az utoljára beírt állapotot (az *S* vezérlőjelet egy S_x oszlopdekóder-bit és egy S_y sordekóder-bit kiválasztó jel 'ÉS' kapcsolatával állítjuk elő). Ez a *tárolás* állapota.

Ha *S* magas szintű lesz, akkor két eset van:

1. *olvasás(READ)*: ha a *D* és a $\overline{D}$ vonalakat kívülről lebegtetjük, akkor az *S* felemelkedésekor a *D* vonalon az *M*, a $\overline{D}$ vonalon az $\overline{M}$ jelenik meg. Ez tekinthető a tárolt adat kiolvasásának.

2. *<u>írás(WRITE):</u>* ha a D és a $\overline{D}$ vonalakat kívülről ellentétesen meghajtjuk, akkor a logikai szintek egymáshoz viszonyított erőssége határozza meg a lezajló folyamatot. Az inverterek kimenetét '–' jellel jelöltük meg, ezzel kifejezve, hogy azok 'gyengébbek', mint a D és a $\overline{D}$ értékeket az M és az $\overline{M}$ pontra kényszerítő MOSFET kapcsolók. Ha D=1 és M=0, illetve a $\overline{D}$=0 és $\overline{M}$=1, akkor a memóriacella átbillen a másik, az előzővel ellentétes állapotba. Ez az írás folyamata. (Az erősebb illetve gyengébb logikai meghajtóképességet az invetereket és a kapcsolókat alkotó MOSFET eszközök megfelelő méretezésével lehet elérni.)

A 7.2.6.b ábrán egy $m_x n$ RAM alapcellából álló bit-szervezésű RAM blokk látható:



1.2.6.b ábra[1]

Bit-szervezésű, $m_x n$ bites RAM blokk

A kapacitásától függetlenül csak egy adatbemenete(*DIN*) és egy adatkimenete(*DOUT*) van, címzése pedig sor- és oszlopdekóderekkel történik. A blokk tartalmaz még egy kapcsolódó *'író-olvasó áramkörök'* egységet is, melyen keresztül zajlik a cellákkal történő kommunikáció. Ennek lényege, hogy a megcímzett bit cellájának az állapota olvasáskor (*R*) a „DOUT" bit-kimenetre kerül, míg íráskor (*W*) a „DIN" bitvektorra kapcsolt érték a memóriának a kijelölt cellájába kerül. Mindkét művelet végrehajtásának a feltétele az is, hogy a használni kívánt blokk kijelölésre kerüljön, azaz a *CS* (Chip Select) vezérlő bemenet magas szintű legyen.

1.2.7 Állapotregiszterek, számlálók

A számlálókat - egy korábbi fejezetben(4.6) már bemutatottak szerint - főként időzítő-vezérlő áramkörökben rögzített funkciójú időzítő egységként szokták alkalmazni, így kapcsolódnak az adatfolyamatokat vezérlő egységek regisztereihez.

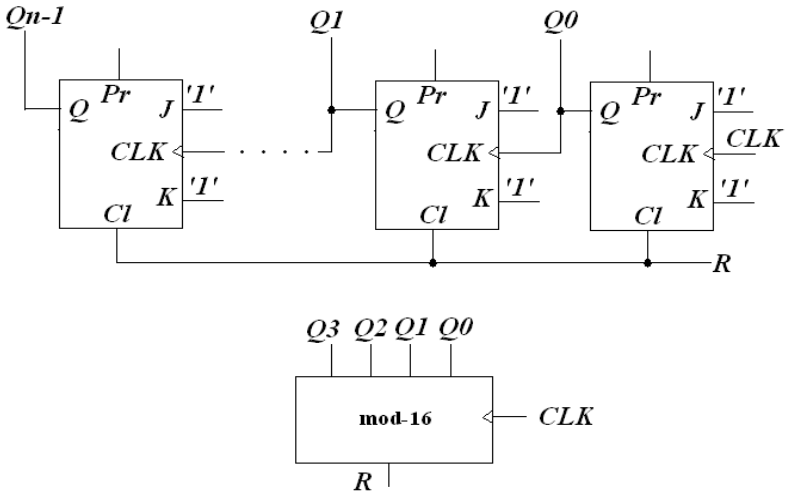
1.2.7.1 Szinkron számlálók

Korábbi tanulmányainkból (II/3) már tudjuk, hogy a 'Pr' és 'Cl" segédbemenetekkel rendelkező, élvezérelt JK -MS tároló (egy 'E' =enable engedélyező bemenettel kiegészítve) képezi a szinkron számlálók alapegységét. Ezeknek a számlálóknak a sajátosságaival, a különböző kialakítási és felhasználási lehetőségeivel –konkrét példákon és feladatmegoldásokon keresztül – már megismerkedtünk és foglalkoztunk, így ebben a fejezetben történő tárgyalásától – a továbbiakban – eltekintünk.

1.2.7.2 Aszinkron számlálók

Az aszinkron számlálók alapját –a szinkron számlálókhoz hasonlóan – a JK-MS tárolók alkotják, és ebben az esetben az ún. „kettes-osztó" funkciót használjuk ki az aszinkron számláncok létrehozásánál. Ismétlésként rögzítsük: a „kettes-osztó" funkció úgy valósul meg, hogy amennyiben a 'mester' tároló beírása az órajel felfutó, a 'szolga' beírása a lefutó élre történik, az órajel frekvenciája megfeleződik, azaz az így kapcsolt JK flip-flop az órajel frekvenciát 2-vel osztja (II/3.1.1.1). Aszinkron számláncokban az órajel logikai szerepet kap, ezért olyan tárolót kell választani, amelyben a 'Cl' bemenet közvetlenül és azonnal hat a kimenetre (az órajel közreműködése nélkül), azaz aszinkron módon működik.

Aszinkron *mod-m* (m = 2n) számlálót kapunk, ha *n* számú kettes osztót a 1.2.7.2. ábrán látható módon kapcsolunk össze:



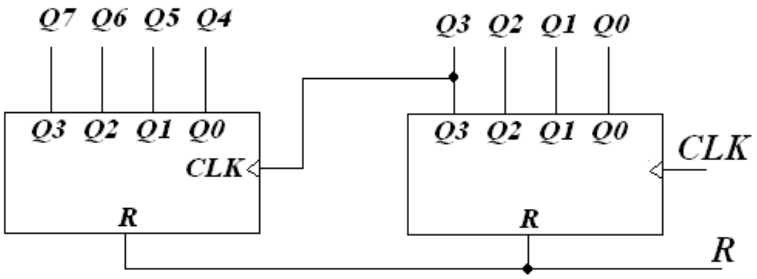
1.2.7.2.a. ábra[1]

Aszinkron számlánc kettes osztókból, és a szimbólum n = 4 esetén

A számlálási (inkrementálási) sorrend ($Q_{n-1}...Q_0$) *R* (RESET)vezérlés után:

$Q_{n-1}...Q_1Q_0$	*eredmény*
000…00	’0’
000…01	’1’
000…10	’2’
… … …	..
100…00	’2^{n-1}’
… … …	..
111…11	’2^n-1’ (=m-1)
000…00	’0’

Adott modulusú számlálókból (hasonlóan a szinkron számlálókhoz) kaszkádosítással nagyobb modulusú számlálót is ki lehet alakítani. Erre mutat példát a 1.2.7.2.b ábra. Itt mod-16-os aszinkron számlálókból állítottunk elő mod-256-os számláló egységet.



1.2.7.2.b ábra[1]

Mod-256 számláló 4-bites aszinkron számláncok kaszkádosításával

1.3 Funkciós egységek

A funkciós egységek egy vagy két, binárisan ábrázolt adattal valamilyen műveletet végeznek el. Működésük megértéséhez elsőként néhány fontos alapelvet rögzítenünk kell.

A funkciós egységek operandusai és eredményei bináris kódban ábrázolt számok. A kettő hatványaival súlyozott bináris kódolás elvét, az egész számok bináris alakjának felírását, a bináris számok decimálisakká való alakításának szabályait matematikai előtanulmányainkból már ismerjük. Szükséges ismeret még a negatív számok (egészek és törtek) ábrázolásának az a módja is, amely könnyűvé teszi az előjeles bináris számokkal való aritmetikai műveleteket. Ezért ebben a bevezető részben megismerjük a komplemens kódok fogalmát és a velük való számolás fő szabályait.

Minden hatvány-kitevős súlyokkal ábrázolt pozitív számból - függetlenül a számrendszer *r* alapjától - képezhető egy inverz szám. Az inverz szám minden egyes számjegye helyébe azt a számjegyet írjuk, amelyet az eredeti számjegyhez hozzáadva a rendszer maximális számjegyét kapjuk. Például 3-jegyű decimális számok esetén: ha N = 642, akkor N_i = 357, hiszen a maximális értékű számjegy 9. Belátható, hogy minden pozitív számra igaz, hogy

$$N + N_i = r^n - 1,$$

Itt *n* a számjegyek száma, azaz az előbbi példa esetében *n* = 3.

Az 3-jegyű pozitív decimális szám és inverzének összege 999, azaz 10^3-1. Ebből következik, hogy az *N* pozitív szám negatív párja a következőképpen írható fel:

$$- N = - r^n + N_i + 1$$

Tehát a -642 felírható, mint (-1000 + 357 +1).

Ha ezeket a kettes hatványai szerint súlyozott bináris számokra alkalmazzuk, akkor az inverz előállítása azt jelenti, hogy minden egyes számjegyet, mint logikai értéket (0 vagy 1) negálni kell, majd ebből megkapjuk a szám negatív párját, ha a legkisebb helyi-értéken 1-et hozzáadunk az inverzhez.

Példa :

legyen egy bináris tört a 0. 1 0 1, ahol a bináris pontot a legnagyobb helyiértékű pozíció után képzeljük. Ilyenkor a szám decimális tört alakban 1.(1/2) + 0.(1/4) + 1.(1/8) = + 5/8 .

Ennek a számnak az inverze 1.0 1 0, komplemense pedig

$$\begin{array}{r} 1.0\ 1\ 0 \\ +\ 0.0\ 0\ 1 \\ \hline 1.0\ 1\ 1 \end{array}$$

Ez tehát a -5/8 tört bináris, komplemens kódja.

A legfontosabb tulajdonság, hogy a +5/8 és a -5/8 összege bináris összeadással bináris zérust ad, azaz az összevonás műveleti eredményeit az algebra szabályainak megfelelően kapjuk meg:

$$\begin{array}{r} 0.1\ 0\ 1 \\ +\ 1.0\ 1\ 1 \\ \hline 1\ 0.0\ 0\ 0 \end{array}$$

A (2^1) súlyú pozícióban keletkezett túlcsordulást nem vesszük figyelembe. Ha a negatív számokat abszolút értékük komplemensével jelenítjük meg, akkor a következő aritmetikai szabályok adódnak :

- a számok összeadása az ábrázolásnak megfelelő eredményt szolgáltat
- egy szám kivonása azonos eredményt ad a komplemensének hozzáadásával.

Ebből az következik, hogy az aritmetikai egységünk a kivonást is összeadással végezheti el.

A fentiekben felvázolt matematikai alapok áttekintése után ismerjük meg részletesen a digitális hálózatokban leggyakrabban alkalmazott funkciós egységek felépítését és működését.

1.3.1 Komparátorok

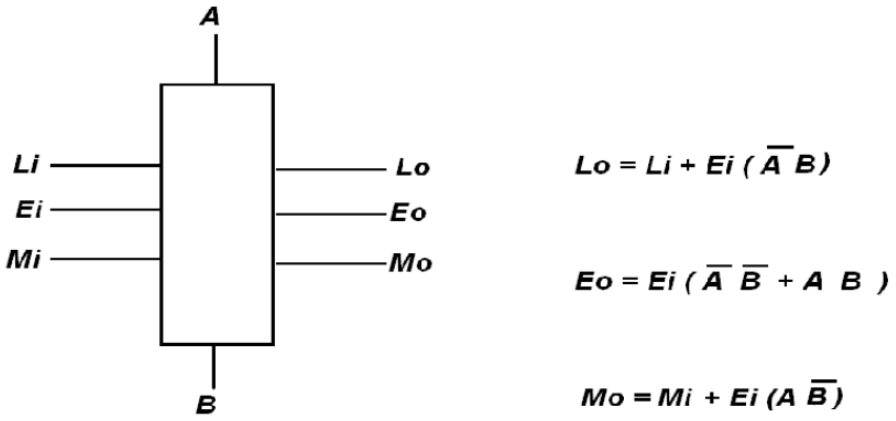
A komparátorok binárisan ábrázolt adatok összehasonlítását végzik el. (A következőkben csak pozitív bináris számok összehasonlításával foglalkozunk, de megjegyezzük, hogy léteznek a komplemens kódra kiterjesztett komparátor változatok is.)

Az összehasonlításnak általában háromféle eredménye lehet:

- az egyik adat nagyobb a másiknál (***Mo***),
- azonos értékű a másikkal (***Eo***),
- kisebb a másiknál (***Lo***).

A logikai áramkörcsaládok legtöbbjénél a komparátorok mindhárom lehetőséget egy-egy kimenettel reprezentálják, de a rugalmas felhasználáshoz az kell, hogy az adott méretű adatok összehasonlítását végző egységek összekapcsolhatók legyenek az operandus-méretek növelésének céljából. A bővítést szolgálja az összehasonlítás elve is, nevezetesen, hogy a komparátor az MSB-től az LSB felé haladva mindaddig nem dönt, amíg valamelyik bit-pozícióban eltérést nem talál. Az eltérés döntést eredményez, és feleslegessé teszi a további alacsonyabb helyi értékű bitek összehasonlítását.

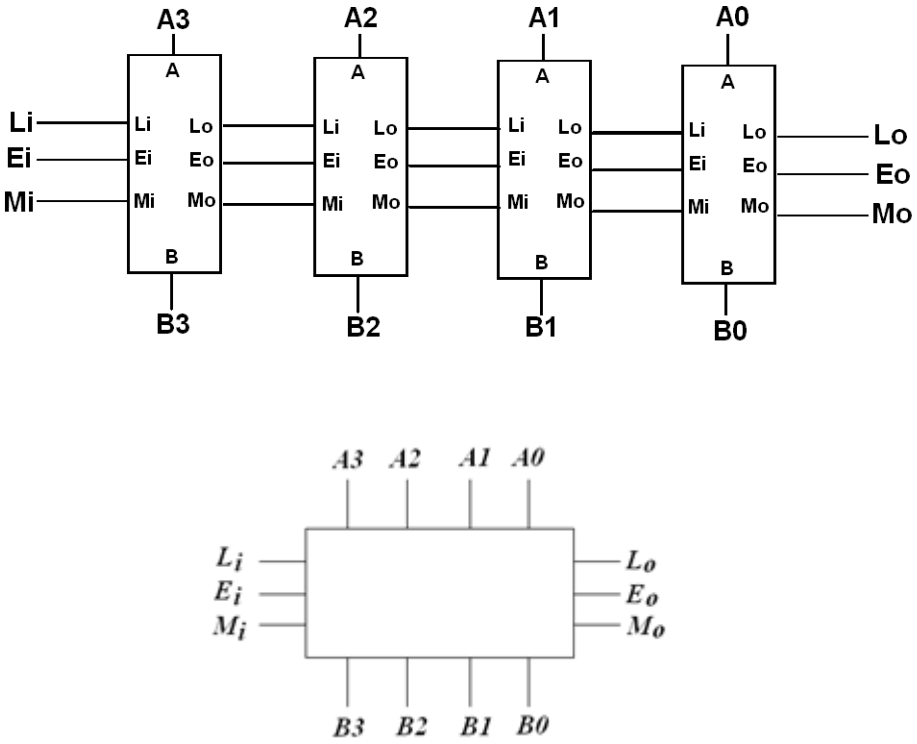
A fenti „univerzális egység” szükségességének elvét követve egészítsük ki az 1-bites alapegységet három bemenettel (***Mi, Ei, Li***), melyekkel biztosítjuk, hogy a már feljebb (MSB→) meghozott ***Mo*** = 1 vagy ***Lo*** = 1 döntés egyenesen kijusson a megfelelő kimenetre. Ezzel egy tetszőlegesen bővíthető (kaszkádosítható) komparátor egységet kapunk, melynek szimbóluma a 1.3.1.a ábrán látható:



1.3.1.a ábra[1]

Egybites univerzális komparátor sematikus ábrája

Egy 1-bites egységekből felépített 4-bites komparátor összeállítását és sematikus vázlatát láthatjuk a 1.3.1.b ábrán:

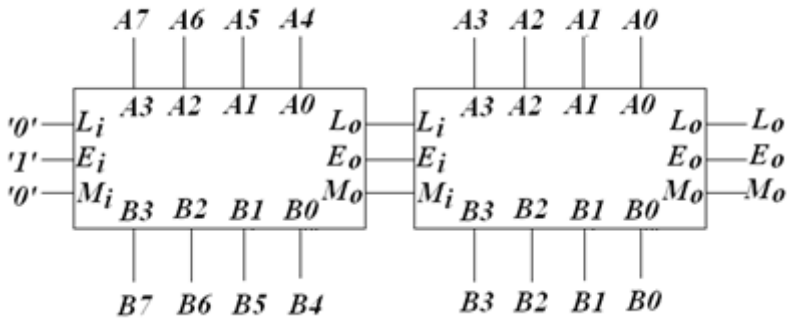


1.3.1.b ábra[1]

4-bites komparátor összeállítása 1-bites egységekből(fent), és sematikus vázlata(lent)

A 1.3.1.c. ábra mutatja, hogy pl. hogyan kapcsoljunk össze két 4-bites egységet, hogy egy 8-bitest kapjunk.

Megjegyzés: az MSB „felől”, azaz a baloldalról „érkező” adatértékek helyére – a helyes működés érdekében - az egység ***Mi, Ei, Li*** *bemeneteire a megfelelő logikai konstans értékeket kell kapcsolni.*



1.3.1.c ábra[1]

8-bites komparátor kialakítása két 4-bites egység kaszkádosításából

1.3.2 Összeadók

A digitális hálózatok aritmetikai logikai egységeinek leggyakrabban használt alapeleme az 1-bites összeadó. Az egység működésének leírása az igazság táblázata alapján egyszerűen definiálható: adjuk meg a táblában a bináris összeadás műveletének eredményét egy adott helyiértéken, ahová áthozhatunk értéket a megelőző, kisebb helyiértékről, ($\boldsymbol{C_i}$), hozzáadjuk az adott helyiértéken megjelenő operandus bitek (***A, B***) összegét (***S***), és képezzük a nagyobb helyiértékre való átvitel értékét is ($\boldsymbol{C_o}$)(1.3.2.a ábra). Ezzel egy olyan univerzális egységet alkottunk, melyet teljes összeadónak hívunk, és ennek segítségével tetszőleges bitszámú operandusokon elvégezhető a művelet (1.3.2.b ábra).

A	B	C_i	S	C_o
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

1.3.2.a ábra

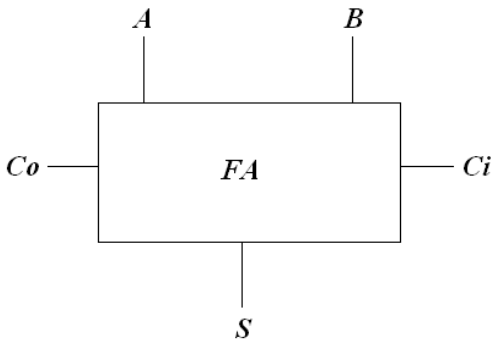
A teljes összeadó igazságtáblája

Az eredmény (***S***) és az átvitel (***C_o***) függvénye az összevonások alapján a következő algebrai alakban adható meg:

$$S = (A \oplus B) \oplus C$$

$$C_o = A\,B + B\,C_i + A\,C_i \quad = \overline{\overline{(AB)}\ \overline{(B\,C_i)}\ \overline{(A\,C_i)}}$$

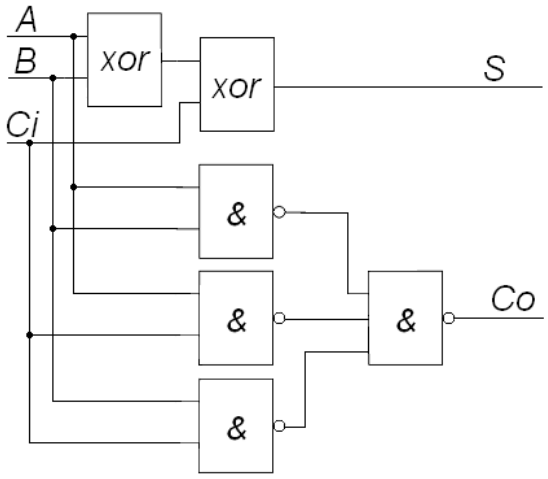
A teljes összeadó szimbóluma a 1.3.2.b ábrán látható:



1.3.2.b ábra[1]

A teljes 1-bites összeadó szimbóluma

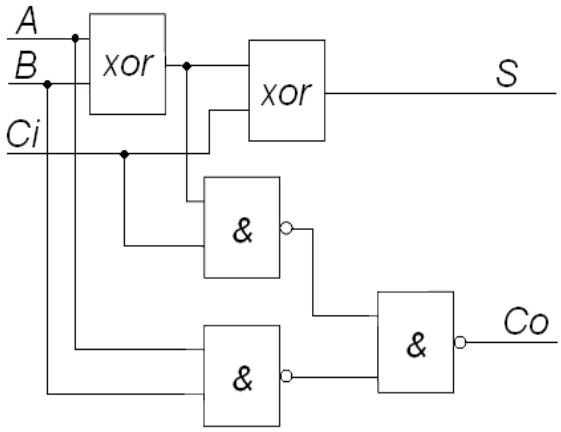
A táblából megadott függvényalak szerinti realizációt a 1.3.2.c ábrán áthatjuk:



1.3.2.c ábra[1]

A teljes összeadó egy lehetséges NAND-NAND realizációja (1)

Ez az áramköri megoldás viszonylag gyors műveleti eredményt szolgáltat, de több kaput tartalmaz, mint a 1.3.2.d ábrán látható realizáció:



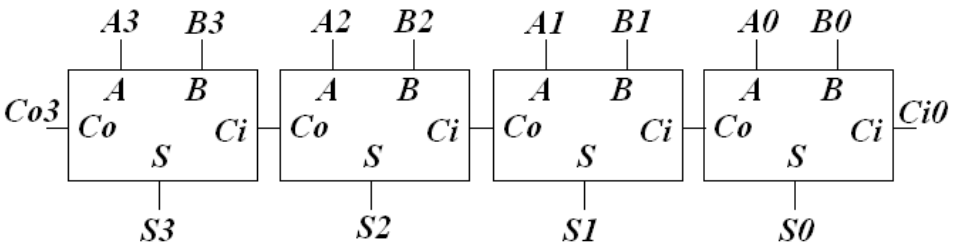
1.3.2.d ábra[1]

A teljes összeadó egy lehetséges NAND-NAND realizációja (2)

Ez a második (*2*) megoldás kevesebb kaput tartalmaz, mint az első (*1*), de lassabb a teljes eredmény ($\boldsymbol{S}$, $\boldsymbol{C_o}$) megjelenítése szempontjából, hiszen az átvitel ($\boldsymbol{C_o}$) három kapun keresztül alakul ki, mert az átalakított függvényalakból ez következik:

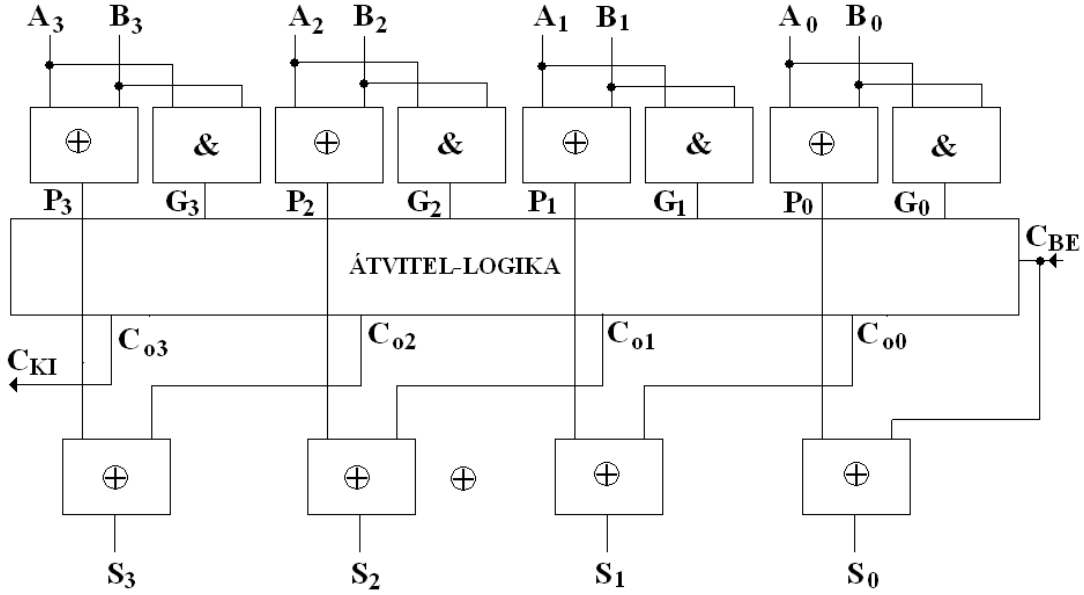
$$C_o = (A \oplus B)\ C_i + A\,B \ = \ \overline{\overline{(A \oplus B)\ C_i}\ \overline{(AB)}}$$

A teljes összeadók kaszkádosításával soros átvitelképzésű bit-vektor összeadót kapunk. (1.3.2.e ábra). A soros átvitelképzés óriási hátránya, hogy a legnagyobb helyiértéken az eredmény az összes fokozaton által késleltetve jelenik meg. Minél szélesebb az összeadó, annál nagyobb lesz a műveleti idő. A műveleti idő csökkentését szolgálják a párhuzamos átvitelképzésű, illetve a vegyes, párhuzamos-soros átvitel-képzésű összeadók.



1.3.2.e ábra[1]

Soros átvitelképzésű 4-bites összeadó egység, 1-bites alapegységek kaszkádosításával



1.3.2.f ábra[1]

Egy 4-bites, párhuzamos átvitelképzésű összeadó felépítése

A párhuzamos átvitelképzésű összeadó esetében minden helyiértéken - így a *k.*-ik helyiértéken is - képezzük a következő logikai jeleket:

$$P_k = A_k \oplus B_k$$
$$G_k = A_k \, B_k$$
$$S_k = (A_k \oplus B_k) \oplus C_{ik} = P_k \oplus C_{o(k-1)}$$
$$C_{ok} = (A_k \oplus B_k) C_{ik} + A_k \, B_k = P_k C_{o(k-1)} + G_k$$

Elvégezve a legkisebb helyiértéktől kezdve a C_{ok} értékek kiszámítását, és minden indexnél behelyettesítve a kisebb helyiértékek bemeneteit kapjuk azokat a logikai kifejezéseket, amelyek alapján összeállítható a 1.3.2.f ábrán bemutatott 4 bites párhuzamos átvitelképzésű összeadó egység.

1.3.3 Kivonók

Az alfejezet elején, a matematikai alapok áttekintése során már említésre került, hogy a kivonás műveletét a bináris számok kódolásának megfelelő megválasztásával összeadásra lehet visszavezetni. Azt a bináris kódot, amelynek alkalmazásakor az összeadás fenti módokon való elvégzése a kivonás műveletét is kiszolgálja, a már tárgyalt *komplemens kód*.

Emlékeztetőül néhány fontos tulajdonsága a kettes-komplemens kódú számábrázolásnak a következő:

legyen

A : a szám, súlyozott bináris kóddal

KK(A) : a szám kettes komplemense, adott szabály szerint előállítva,

egy kettes komplemens kódú szám (-1)-szerese pedig a szám kettes komplemense, vagyis

A + **KK**(A) = 0!

A kettes komplemens kód összetétele:

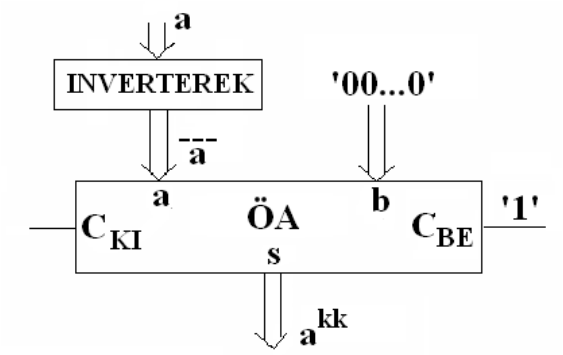
- *MSB* : előjel
- *(MSB-1) – LSB* : a számérték.

Ennek megfelelően

- ha a szám pozitív, akkor előjele ’0’, a számérték pedig a szám binárisan súlyozott abszolút értéke

- ha a szám negatív, akkor előjele ’1’, és az abszolút érték a kettes komplemens előállításával határozható meg

A 7.3.3.a ábra szerinti egység az *a* kettes-komplemens kódban ábrázolt szám kettes-komplemensét állítja elő: azaz, minden kivonandót rajta átengedve, összeadással hajthatjuk végre a kivonást:

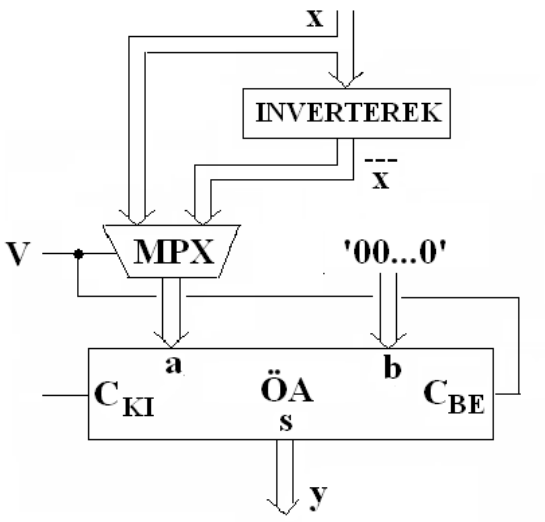


1.3.3.a ábra[1]

A kettes-komplemens képző egység sematikus ábrája, teljes összeadóból kialakítva

Ha ezt az egységet az összeadó *a* adatbemenetén a 1.3.3.b ábra szerinti kialakításban egy MPX2-1 multiplexerrel kiegészítjük, akkor egy ún. *vezérelhető kettes-komplemens képző egységet* kapunk. A működés lényege, hogy az egységünk

- ***V*** = 1 esetén az ***x*** kettes komplemensét állítja elő
- ***V*** = 0 esetén ***y*** = ***x***.

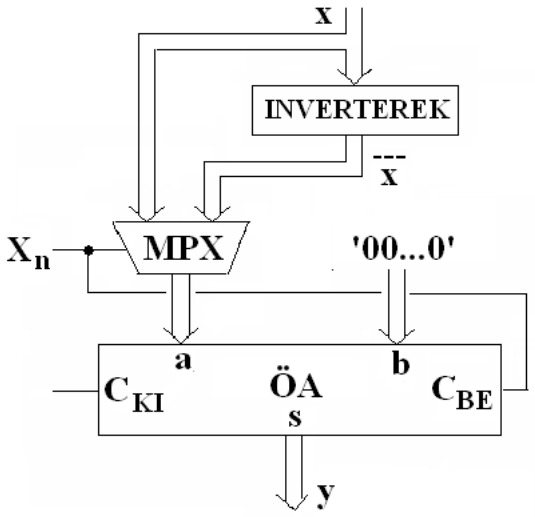


1.3.3.b ábra[1]

Vezérelhető kettes-komplemens képző egység.

Ezzel a kiegészítéssel lényegében egy *abszolútérték képző egységet hoztunk létre* 1.3.3.c ábra), melynek használatával azt a funkciót biztosítjuk, hogy bármely kettes komplemens kódban érkező szám abszolút értéke kerül az egység kimenetére. Az ábrán X_n az MSB, egyben a szám előjele:

- ha értéke ’1’, akkor a szám kettes komplemense lesz az abszolút érték,
- ha értéke ’0’, akkor ***y*** = ***x.***



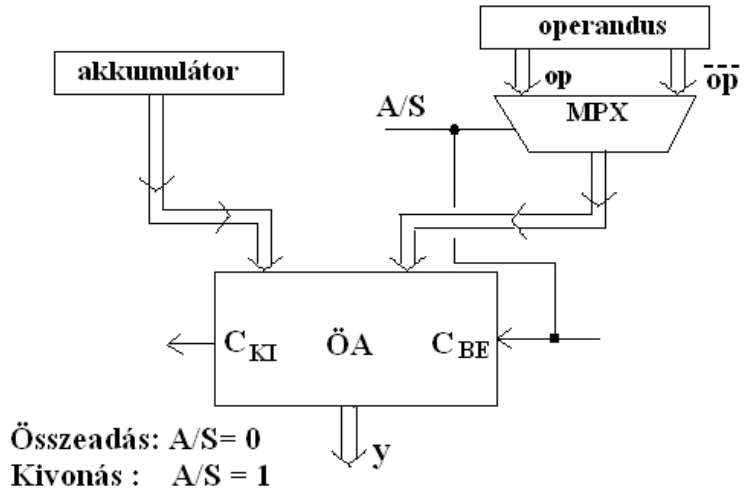
1.3.3.c ábra[1]

Abszolútérték képző egység

A 1.3.3.c ábrán látható egység képezi alapját a mikroprocesszorokban alkalmazott univerzális összeadó-kivonó egységnek(1.3.3.d ábra). A működés értelmezése a következő:

- ha összeadunk, ***A/S*** = ’0’, így az összeadó jobboldali bemenetére maga az operandus kerül az azt tároló regiszterből,

- ezzel szemben, ha ***A/S*** = ’1’, azaz kivonni kell, a multiplexer az operandus bitenkénti negáltját kapcsolja az összeadóra, sőt a C_{BE} bemenet is ’1’, azaz az operandus kettes-komplemense kerül összeadásra az akkumulátor tartalmával.



1.3.3.d ábra[1]

Mikroprocesszorokban alkalmazott univerzális „összeadó-kivonó” aritmetikai egység

1.3.4 Szorzók és osztók

A szorzás és osztás műveletének elvégzésére a digitális rendszerekben – a felhasznált alapelvek és kialakítások alapján - sokféle és különböző megoldási lehetőségek adódnak az alkalmazható funkcionális egységeket tekintve. Ezek lehetnek pl. fix- vagy lebegőpontos műveletvégzők, ezen belül soros vagy párhozamos átvitel logikával működő egységek stb.

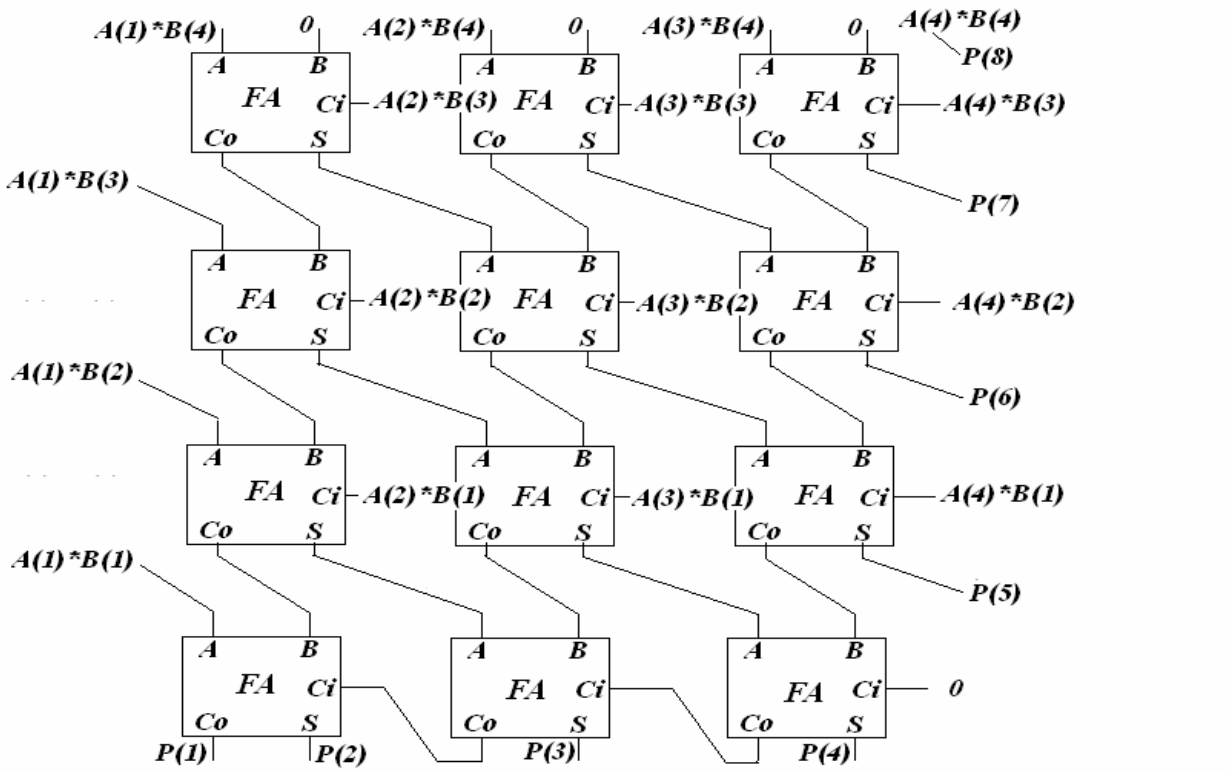
A teljesség igénye nélkül néhány példa ezek közül:

- szorzók:
  - előjeles vagy előjel nélküli szekvenciális szorzók
  - tömbszorzók (array-szorzók)
  - fa-szorzók (Booth-Wallace szorzók)
    …

- osztók:
  - előjel nélküli visszaállító osztó
  - előjeles nem visszaállító osztó
  - Radix 4 SRT osztó
  - funkcionális iterációs osztók (Newton-Raphson algoritmus)
    …

Ezeknek a funkcionális egységeknek a tételes és részletes bemutatása nem képezi ezen fejezet részét, de példaként egy tömbszorzó felépítésén keresztül betekintést nyerhetünk egy, a gyakorlatban gyakran alkalmazott logikai műveletvégző egység felépítésébe és működésébe.

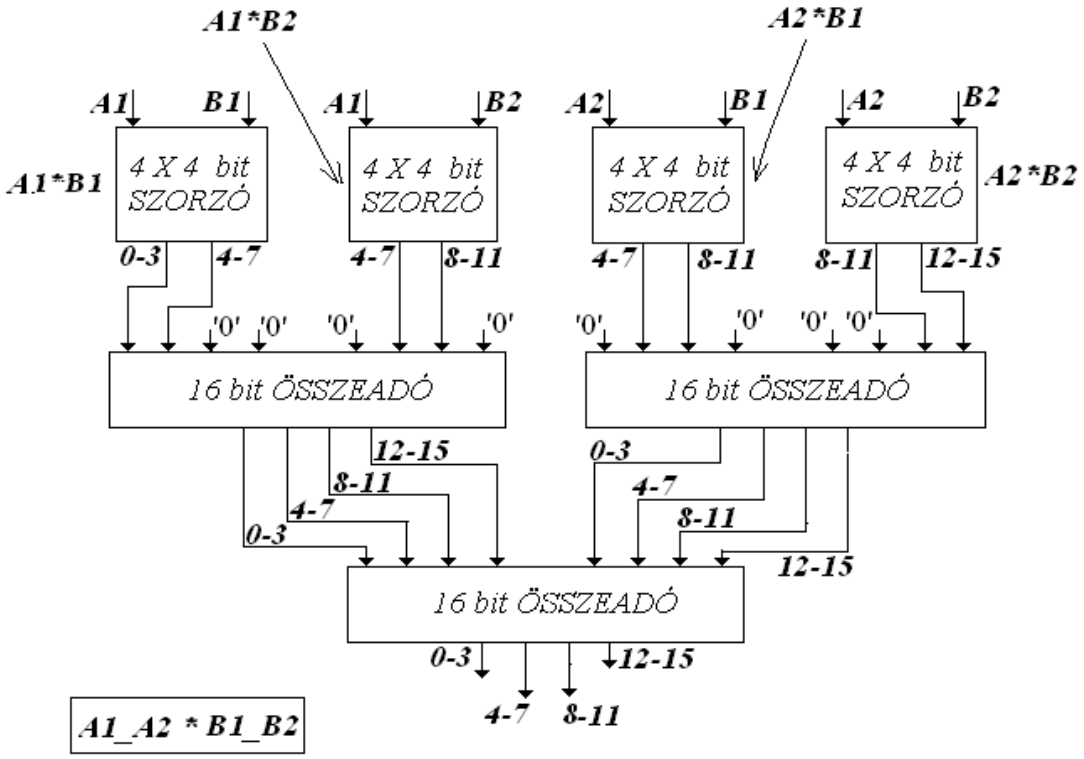
A 1.3.4.a ábrán egy 4x4 bites tömbszorzó (array-szorzó) kialakítását láthatjuk. A szorzókat, hasonlóan az összeadókhoz az jellemzi, hogy a bináris vektorok közötti bitenkénti 'szorzás-léptetés-összeadás' műveletsorát végzik el. A tömb speciálisan összekapcsolt teljes összeadókból áll. Az ***A(i)*B(j)*** jelölések 'ÉS' kapukat szimbolizálnak. Az ábrán két 4-bites bináris tört-vektor szorzását mutatjuk be, az indexek a bináris ponttól jobb felé, azaz a kisebb helyiértékek felé növekednek. Így a szorzat legnagyobb helyiértékén ***P(1)***, a legkisebb helyiértékén ***P(8)*** szorzat-bit szerepel. Hangsúlyozzuk, hogy az ábra szerinti tömb pozitív számok szorzására alkalmas, de hasonló tömbszorzó megoldásokat fejlesztettek ki kettes-komplemens kódban ábrázolt előjeles számok kettes-komplemens kódú eredményt szolgáltató szorzására is. Működés szempontjából lényeges tulajdonsága az egységnek, hogy az eredmény megjelenítési ideje, vagyis a blokk késleltetése a bitvektorok dimenzió számával közel lineárisan nő.



1.3.4.a ábra[1]

Egy 4 x 4–es tömbszorzó kialakítása teljes összeadókból valamint 'ÉS' kapukból

Nagyobb bitszámú egység kialakítására példa a 1.3.4.b ábrán látható, 4-bites tömbökből létrehozott 8x8 bites szorzó.

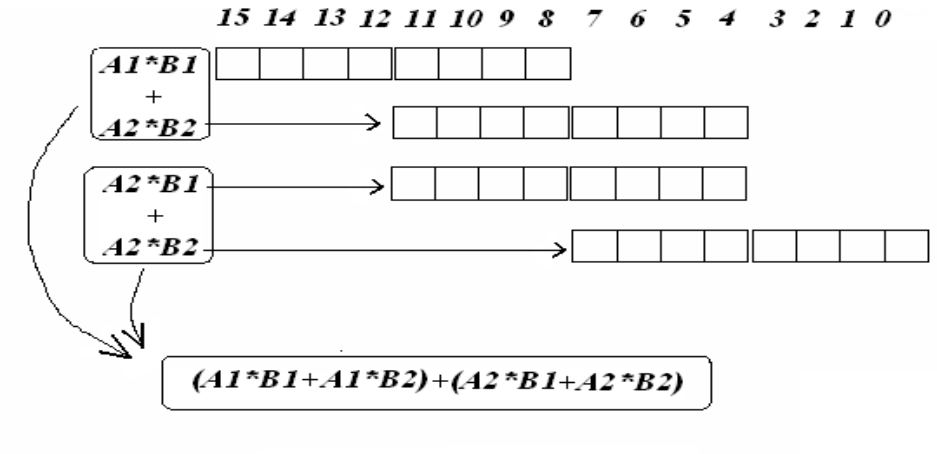


1.3.4.b ábra[1]

8 x 8 bites array-szorzó, 4 x 4-s komponensekből

Megjegyzés: A 7.3.4.b. ábra szerinti séma minden egyes összeköttetése egy 4-bites vektort jelöl. A helyes értelmezés: „A" szorzat MSB : 0, LSB: 15)

Ha az ***A1_A2*** két négybites vektor lánca az egyik operandus, a ***B1_B2*** - amely ugyancsak két négybites vektor lánca - a másik operandus, akkor a szorzatokat 4x4 bites szorzókkal, részletekben is elő lehet állítani. Ezután minden részletszorzatot a maga helyiértékének megfelelő helyen kell 16-bites összeadókra vezetni (1.3.4.c ábra).



1.3.4.c ábra[1]

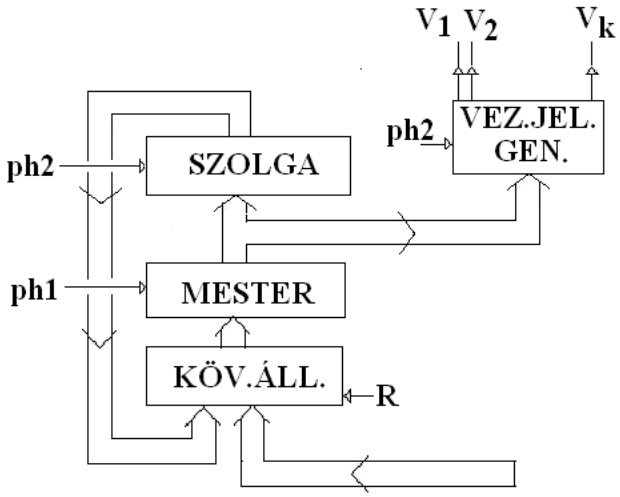
*Műveletvégzés a 8-bites array-szorzóban **A1_A2** és **B1_B2** operandusokkal*

1.3.5 Vezérlők

A 4.6.4 fejezetben már említést tettünk a digitális berendezések tervezésének egy fontos lépéséről, a dekompozícióról, melynek lényege, hogy elkülönítjük az adatutakat az azok kijelölését és időbeli mozgatását meghatározó időzítő vezérlő egységtől.

Mintapéldán keresztül (II/3.1.4.2) betekintést nyerhettünk egy szinkron számláló bázisú időzítő vezérlő egység tervezésébe és működtetésébe. Fontos gondolatként emeltük ki a hazárdmentes vezérlés biztosítását is.

A vezérlőknek egy másik csoportját alkotják az ún. FSM (Finite State Machine) típusú vezérlők. Strukturális kialakításukat tekintve egyik fő jellegzetességük, hogy az időzítő lényegében véges állapotszámú, MS tárolókból az adott célra felépített szinkron sorrendi hálózat. A kétfázisú (ph_1, ph_2) órajellel működtetett FSM vezérlő sematikus felépítése a 1.3.5.a ábrán látható:

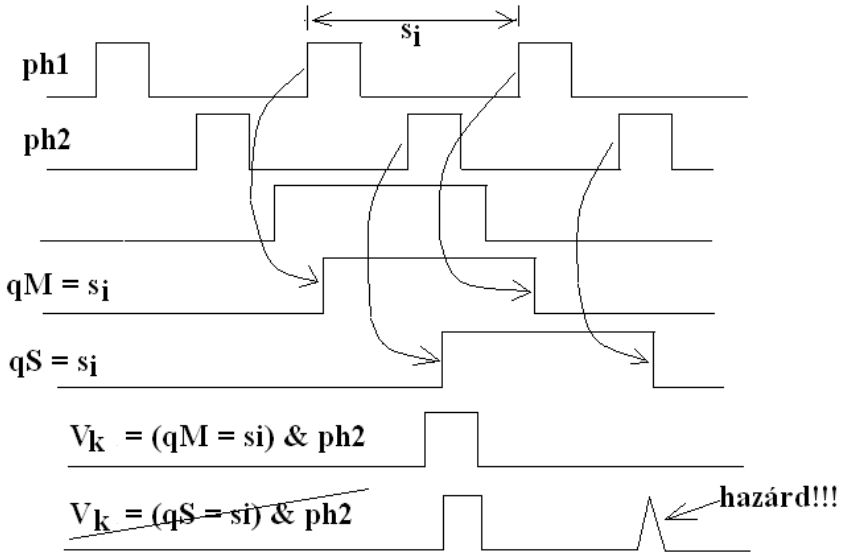


1.3.5.a ábra[1]

Kétfázisú órajellel(ph1,ph2) működő FSM típusú vezérlő

A rajzon jól felismerhető a klasszikus, kétfázisú szinkron sorrendi hálózat struktúra, ahogy a flip-flopok MESTER fokozatait is és a SZOLGA fokozatokat is összefogjuk egy-egy élvezérelt regiszterben. A működést reprezentáló idődiagramon (1.3.5.b ábra) az látható, hogyan kell kivitelezni egy ph_2 fázissal szinkronizált vezérlőjel generátorát: lényeges, hogy csak a MESTER fokozatról levett időzítő információ eredményez hazárdmentes megoldást. Fontos kitétel az is, hogy az állapotokhoz tartozó idő intervallumokat most a ph_1 felfutásai között kell definiálni.

Megjegyzés: a klasszikus MSI-szintű logikai áramkörök világában inkább a számláló típusú vezérlőt érdemes alkalmazni, hiszen a számláló készen van, és funkciói kihasználhatók, míg az LSI-VLSI világban leginkább a kétfázisú FSM-típusú vezérlő tervezése az ajánlott.



1.3.5.b ábra[1]

Kétfázisú órajellel(ph1,ph2) működtetett FSM típusú időzítő idő-diagramja

Az FSM vezérlőket gyakran alkalmazzák önálló makro-cellákban. Az önálló makro-cellák LSI-VLSI áramkörök építőelemei lehetnek. Sajátosságuk, hogy egy speciális feladatra tartalmazzák mind az adatstruktúrát, mind az időzítő vezérlőt, és beilleszthetők egy nagyobb vezérlési folyamatba. Fontos, hogy több ilyen önálló egység osztozkodhat az adatstruktúra elemein, ha nem akadályozza ezt erőforrás igények közötti ütközés.

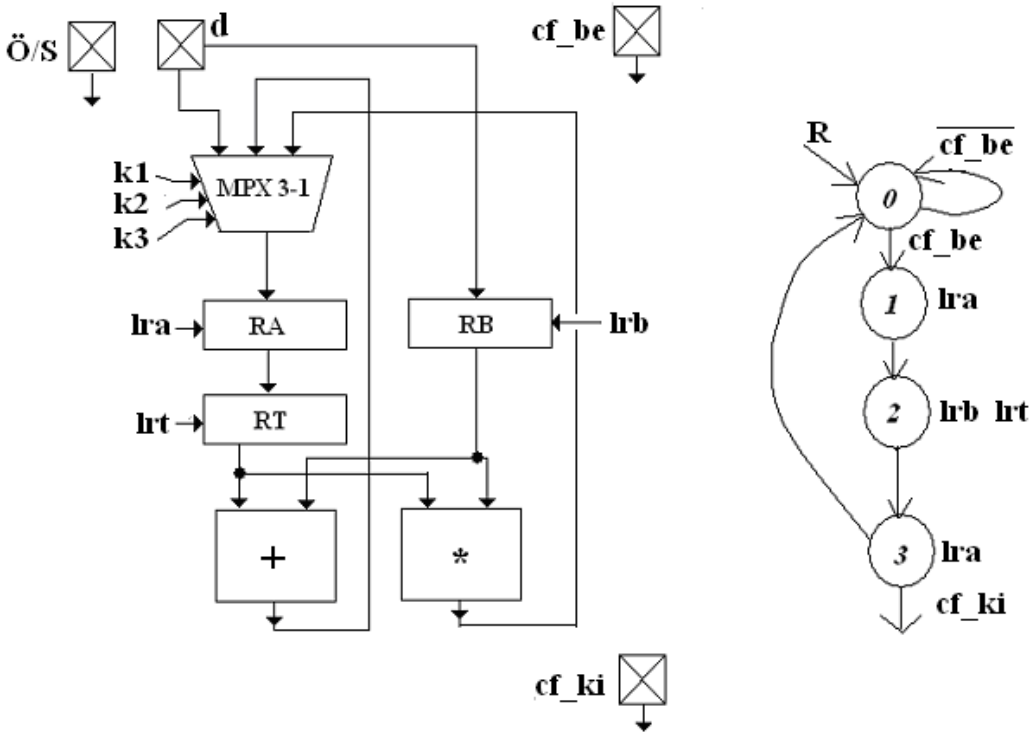
Példa:

*Valósítsuk meg a következő önálló (autonóm) makro-cellát: a megtervezendő egység várakozzon a vezérlési folyamatot átadó másik önálló egységtől vezérelt **'cf_be'** jel felfutására. Ha ez megérkezett, egymás után két adatot ugyanarról a **'d'** adatsín bemenetről írjon be két regiszterbe (**RA, RB**), majd az **'Ö/S'** bemenet szintjétől függően adja össze, vagy szorozza össze azokat, és az eredményt írja vissza az **'RA'** regiszterbe. Ha ezt elvégezte, küldje tovább a vezérlést egy harmadik önálló egységnek a **'cf_ki'** jel felemelésével, ő maga pedig várakozzék újabb indításra, de a **'cf_ki'** jelet ejtse le alapállapotába.*

Megoldás:

Ami az adatstruktúrát illeti, világos, hogy legalább két regiszterre és két funkciós egységre, egy összeadóra és egy szorzóra van szükségünk. Beláthatjuk azonban, hogy az

egyik operandusnak az eredménnyel való felülírása megköveteli egy átmeneti regiszter (***RT***) bevezetését is. Nyilvánvaló ugyanis, hogy a funkciós egységek kombinációs egységek, és bemenetükön nem írhatjuk felül azt az adatot, amely az eredmény stabilan tartásához és tárolásához szükséges. Ezért az ***RA*** tartalmát átmentjük egy átmeneti tárolóba, ezt kapcsoljuk a funkciós egységekre, és így az ***RA*** befogadhatja az eredményt. Azt is könnyen kitalálhatjuk, hogy az ***RA*** regiszterbe végül is három különböző forrásból kell tudni adatot betölteni. Ezek a '***d***' adatsín, az összeadó kimenete és a szorzó kimenete.



1.3.5.c ábra[1]

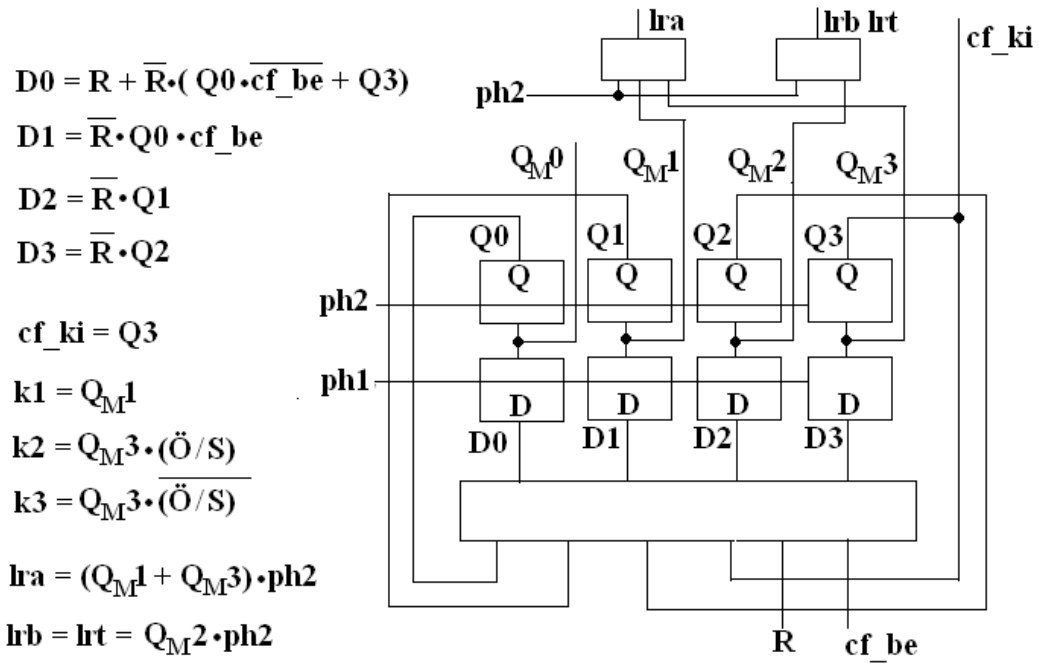
A mintapélda szerinti FSM típusú időzítő vezérlővel működő autonóm egység adatstruktúrája és időzítőjének állapotgráfja

Ezek után felvázolható a 1.3.5.c ábra bal oldalán látható adatstruktúra. Mindhárom regiszterből „kilógnak" a betöltőjelek, (***lra, lrb, lrt***) az MPX3-1 multiplexerből pedig a három kiválasztó bemenet, (***k1, k2, k3***). Természetesen a művelet-kiválasztó jel is szerepet kap a vezérlőben, csak úgy, mint az FSM-et elindító '***cf_be'*** külső vezérlő bemenet. Természetesen szolgáltatni kell a '***cf_ki'*** vezérlés továbbadó jelet is.

Az ábra jobb oldalán az FSM állapotgráffal megadott ütemezését láthatjuk. A '***0***' sorszámú állapot a kezdeti állapot, ahová az '***R'*** jel kényszeríti az időzítőt. Ebben az állapotban várakozni kell a vezérlési folyamat átadására, azaz a '***cf_be***' jel

felemelkedésére. Innen kezdve sorrendben követik egymást az *'1', '2'* és *'3'* sorszámú állapotok. Az *'1'*-es állapotban az *'lra-n'* beíró-impulzust kell kiváltani. Ez alatt a *'d'* sínre kell kapcsolódnia az ***RA*** regiszter bemeneteinek, azaz a multiplexer *'k1'* kiválasztó jelét kell magasra emelni. A *'2'*-es állapotban egyszerre töltjük a második adatot a *'d'*-ről az ***RB*** regiszterbe, illetve a ***RA***-ból az első adatot az átmeneti regiszterbe. Az *'lrb'* és *'lrt'* betöltő impulzusok tehát egyszerre jelentkeznek. Innen kezdve a két funkciós egység kimenetén megjelenik az összeg, illetve a szorzat. A *'3'*-as állapotban ismételten az *'lra'*-ra adunk impulzust, de ez alatt a *'k2'* vagy a *'k3'* kiválasztó bemenetek egyikének kell magasan lennie.

A megvalósított időzítő-vezérlő sémája a 1.3.5.d ábrán látható. A *'D'* bemenetekre csatlakozó fekete doboz tartalmát a baloldali logikai kifejezések mutatják. Az egyes *'D'* függvényalakokat elemezve látható, hogy '1'-es súlyú állapotkódolást alkalmaztunk.



1.3.5.d ábra[1]

A mintapéldában megfogalmazott időzítő vezérlő megvalósítása.

Felhasznált irodalom:

[1] Dr. Keresztes Péter: Digitális hálózatok (HEFOP elektronikus Jegyzet)