Mi az a GAL?

Komló Bertalan okl. villamosmérnök
Rétallér István híradásipari technikus (HA5BWH)

Ki nem bosszankodott már amiatt, hogy valahová egy inverter kellett volna és nem maradt egy szabad kapu sehol a „környéken", amiből kialakítható lett volna. Vagy milyen jó lenne néha egy olyan IC, amiben nem csupa NAND kapu lenne, hanem mondjuk felesben NAND és NOR. A PAL és most a GAL valóra váltják ezt a hihetetlen álmot. IC-t tervezhet a felhasználó, pont olyat, ami az adott helyre kell!

A Generic Array Logic – rövid nevén a GAL – egy darab gyurma, amit olyanra formálok, amilyenre akarok. Komolyabban szólva, egy szabadon alakítható szilíciumfelület, amiből kapuk és flip-flopok, illetve ezek hálózata alakítható ki, természetesen kötött darabszámban.

Ez túl szépen hangzik, ennek biztosan kemény ára van – szólal meg bennünk az élettapasztalat. De most feleslegesen.

Lássuk a tényeket:

– 1 db 16V8 akár 4-6 TTL IC-t kiválthat. Ára mindössze 150-160 Ft, ebben már az ÁFA is benne van.

– Nyákfelületben: 5 db 14 lábú IC helyett egy db 20 lábú (keskeny) IC kell csak.

– Sebességben: a GAL ki- és bemenetei között a késleltetési idő független a megvalósított hálózat bonyolultságától. A késleltetés a HC és LS TTL család sebességének megfelelő, de figyelembe véve, hogy a GAL egy bonyolult hálózatot egymaga meg tud valósítani, a diszkrét tokokból összerakott hálózathoz képest gyorsabb is lesz.

– Fogyasztása: CMOS technológiával készül.

További előnyök:

– 100-szor újra programozhatóak.

– Egy speciális bit kiégetésével (bár továbbra is tökéletesen működnek) többé nem kiolvashatóak.

– Törlés után ez a bit is törlődik, tehát újra teljes értékű, írható/olvasható lesz az eszköz.

– Törléséhez nem kell UV lámpa, a programozó készülékkel törölhető is.

Az általunk tárgyalandó 16V8 és 20V8 típusok mindegyike egyaránt 8 D flip-flop kialakítását teszi lehetővé. Hogy kapukból hány darab alakítható ki, azt nem lehet ilyen egyszerűen meghatározni. Pl. a 16V8 a következő kapukká alakítható:

– 9 db inverter

– 6 db kétbemenetű kapu (AND, NAND, OR, NOR, XOR, XNOR stb.) tetszőleges arányban keverve.

– 8 db D flip-flop közös órajel-bemenettel és reset lehetőséggel.

Ezek a példák annak illusztrálására is szolgálnak, mire nem szabad egy GAL-t pazarolni, ugyanis sokkal bonyolultabb hálózatok is realizálhatók vele. Az **1. ábra** egy konkrétan megvalósított példán keresztül ezt illusztrálja.

Ez egy logikai analizátor vezérlőegysége. Rövid, tüskeszerű impulzusokat is észre kell vegyen, ezek billentik az RS flip-flopokat, amiknek a kimenete a „D" flip-flopokon át, már a rendszer órajeléhez szinkronozottan jelenik meg. Számoljuk össze, hogy hány TTL IC-t vált ki itt egyetlen GAL:

– 1 db 74LS86 az egy darab XOR kapu miatt

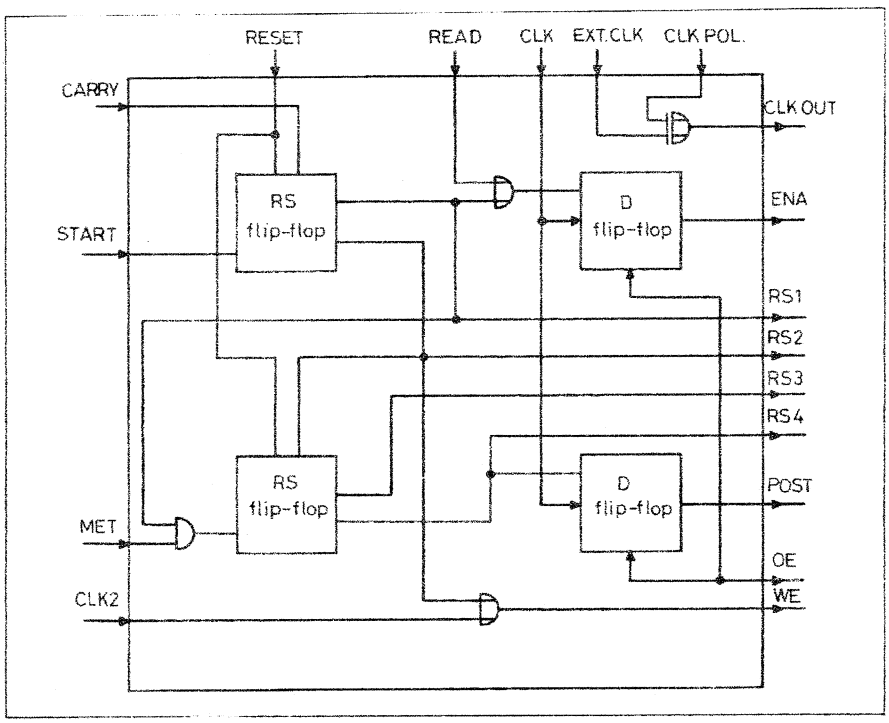
– 1 db 74LS32 a kettő darab OR kapu miatt

– 1 db 74LS08 az egy darab AND kapu miatt

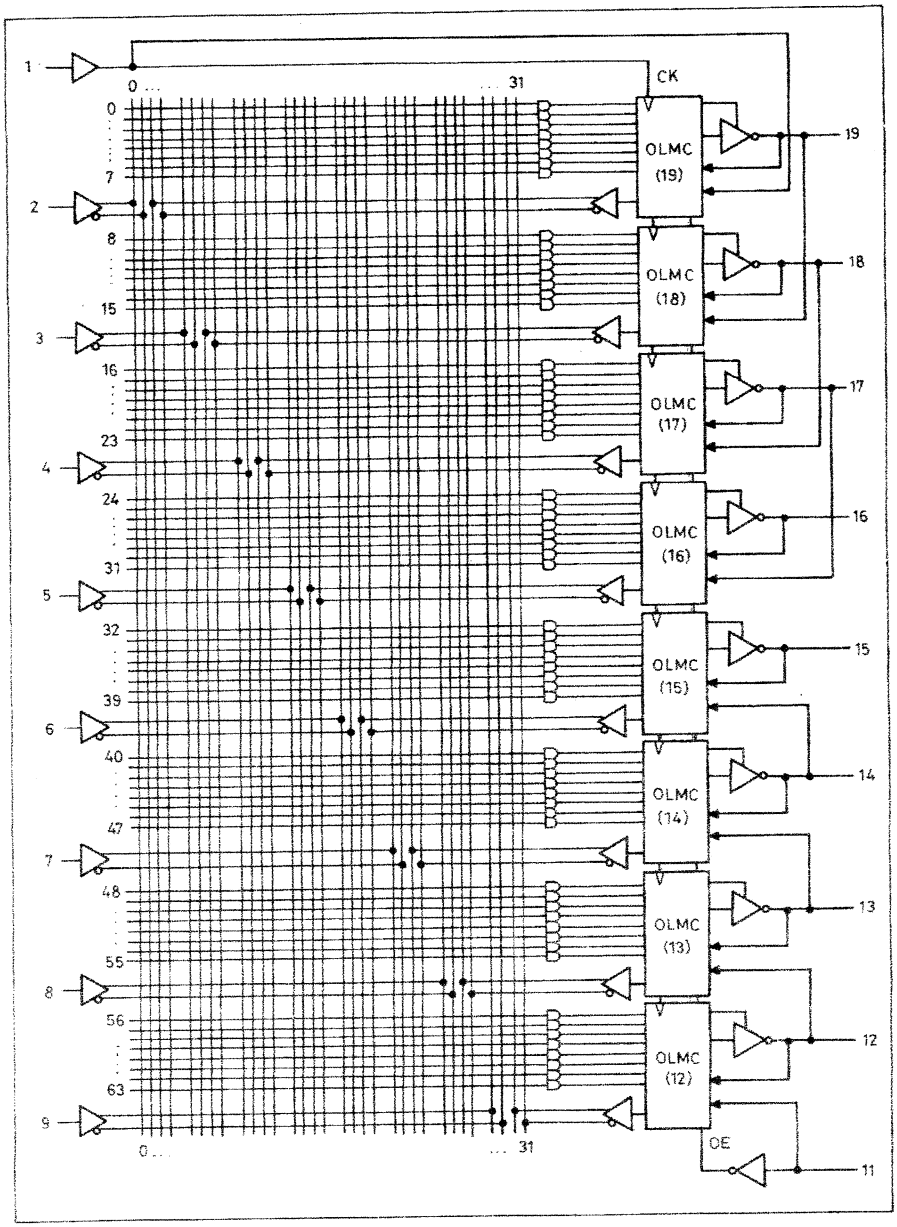
– 1 db 74LS74 a D flip-flopok kialakításához

– 1 db 74LS02 az RS flip-flopok kialakításához.

Ezzel még nem is számoltunk be mindent. Kihagytuk, hogy az ENA és POST kimenetek az OE lábon tristate állapotba vezérelhetők, hogy a felső RS flip-flop nem 2, hanem 3 bemenetű és egyik lábán szintfordításhoz egy inverter is kellene.



1. ábra



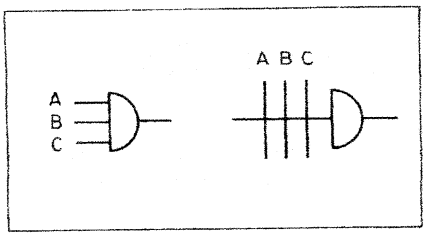
2. ábra

Még ilyen nagyvonalúan számolva is 5 db TTL IC-t vált ki egyetlen GAL!

Ahhoz, hogy egy bonyolult kapuhálózatot jó hatásfokkal tudjunk megvalósítani, vagy egy GAL fejlesztőrendszer szükséges – ebből több, neves program is kering kézről kézre – vagy a logikai egyenletek megfelelő szintű ismerete.

A GAL tartalmának összeállításához az ismert félvezetőgyártók közül az AMD PALASM nevű programja, vagy az ABEL a National Semiconductor nevével fémjelzett software egyaránt alkalmas. A logikai egyenletek nagy figyelmet kívánó egyszerűsítése nemcsak fárasztó, de hibalehetőségeket is hordozó procedúra, bár kétségtelen, hogy még ma is vannak akik „kézzel" tervezik a GAL tartalmát. Aki teheti mindenképpen valamelyik programmal tervezzen, mert a jobbak még szimulálni is tudják a megtervezett áramkört a felhasználó által tetszőlegesen megadott bemenőjelekre, akár a kimeneteken megjelenő hullámformát is megrajzolják.

Akár programmal, akár manuálisan tervezzünk, szükség van a GAL belső felépítésének ismeretére, hogy a benne rejlő lehetőségeket ki tudjuk használni. A **2. ábra** a 16V8 belsejét mutatja.



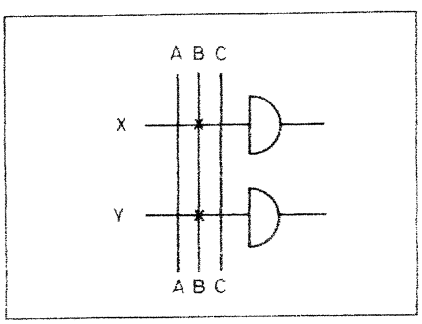
3. ábra

A mátrixszerűen elrendezett vezetékek tetszőleges találkozási pontjaiban összeköttetést lehet létesíteni, míg más metszéspontokat egymástól elszigetelten tarthatunk. Az ábrázolás azonban egyszerűsítéseket tartalmaz. Ezt a **3. ábra** mutatja.

A bal oldali rész a hagyományos ábrázolása egy AND kapunak, a jobb oldali a GAL-on belül alkalmazott jelölést mutatja be. Ezzel a jelölésmóddal egyszerűbbé és könnyebben érthetővé válik a belső struktúra rajza **(4. ábra).**

A 4. ábra nem azt jelenti, hogy az X és Y pontok össze lennének kötve, pusztán arról van szó, hogy a középső függőleges vezeték mindkét kapu „B" bemenetére csatlakozik. Ez a jelölésmód elsőre talán kissé idegen, de meg kell barátkoznunk a gondolattal, hiszen gyártótól függetlenül ez az adatlapokon alkalmazott szokásos ábrázolási mód.

A 2. ábra tüzetesebb vizsgálata azt mutatja, hogy itt AND kapuk bemeneteit tudjuk programozni, és a nyolc kapu kimenete bemegy valamibe. Ez a valami az OLMC, ami egy nyolc bemenetű OR kapuval egyetlen logikai függvénnyé fogja össze az AND kapuk elején található 64 programozható mátrixpontot. Az OLMC (Output Locic Macrocell – kimeneti logikai makrocella) először a GAL-oknál alkalmazott építőelem, aminek csak egy része a már említett nyolc bemenetű OR kapu. Ezenkívül multiplexe-



4. ábra

Mi az a GAL 2.

Komló Bertalan okl. villamosmérnök
Rétallér István híradásipari technikus (HA5BWH)

Lássuk, hogyan formálhatjuk a GAL struktúráját az OLMC-k segítségével. Az OLMC-k tipikus felépítése az 5. ábrán látható. Figyelmesen megnézve az ábrát láthatjuk, hogy az OLMC 1 db 8 bemenetű OR kapuból, 1 db 2 bemenetű XOR kapuból, 1 db D FLIP-FLOP-ból, 1 db kimeneti bufferből és 5 db multiplexerből áll. Az OLMC struktúrájának változtathatóságát a multiplexerek biztosítják. Nézzük meg, mi a szerepük az egyes multiplexereknek!

PTMUX (product term multiplexer) az OLMC-hez kapcsolódó 8 db product term (az ÉS mátrix elemei) egyikét vagy az OR kapura, vagy a tristate multiplexerre kapcsolja.

OMUX (kimeneti multiplexer) a kimeneteket kapcsolja regiszteres vagy kombinációs típusú kimenetté.

FMUX (feedback multiplexer) a visszacsatolást változtatja az adott kimenetről, illetve ez a multiplexer kapcsolja bemenetté az adott OLMC-t.

TSMUX (tristate multiplexer) a kimeneti buffer tristate vezérlőjeleit kapcsolja.

Az ötödik multiplexer a TSMUX egyik bemenetét vezérli a **SYN** bit függvényében.

Tartozunk az OLMC multiplexereit vezérlő bitek ismertetésével:

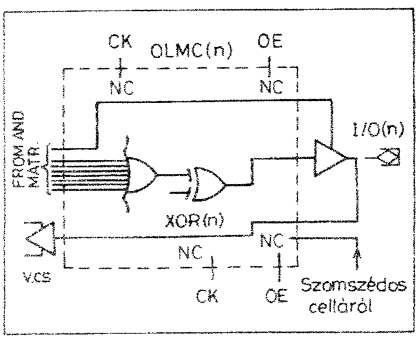
SYN (synchronous) ha ez a bit logikai L értékű az eszközben legalább egy regiszteres kimenet van.

ACO ha ez a bit logikai L értékű, nincs az eszközben tri-state kimenet.

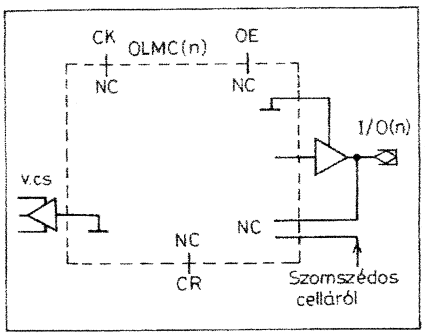
AC1(n) ez a bit jelzi, hogy az adott láb kimenet vagy bemenet (L = kimenet H = bemenet)

XOR(n) ez a bit dönti el, hogy az adott kimenet invertáló vagy nem invertáló típusú („H" = invertáló „L" = nem invertáló).

A SYN és az ACO egy-egy bit mely mind a nyolc OLMC-t vezérli egyidejűleg, az XOR(n) és AC1(n) bitekből minden OLMC-hez tartozik egy önálló bit. Az 5. ábrán látható



6. ábra



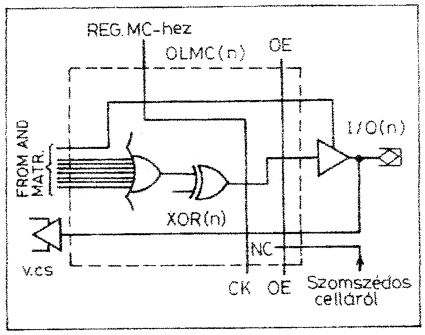
7. ábra

AC1(m) bit a szomszédos OLMC AC1 bitje. Azt hogy az adott OLMC-nek melyik a szomszédja, a 2. ábráról dönthetjük el, aszerint, hogy melyik kimenetről van csatolás az adott OLMC-re.

A vezérlő jelek ismeretében nézzük meg, milyen konfigurációk alakíthatók ki. Az **1. táblázatban** a két kitüntetett (16V8-nál a 12-es és a 19-es, 20V8 esetén a 15-ös és a 22-es lábakhoz tartozó) OLMC-k lehetséges konfigurációit foglaltuk össze.

A **2. táblázatban** az összes többi lábhoz tartozó OLMC-k lehetséges konfigurációit soroltuk fel.

A GAL programozásának jelentős részét az OLMC-k programozása teszi ki, ezért a jobb érthetőség kedvéért a következő ábrákon a különböző



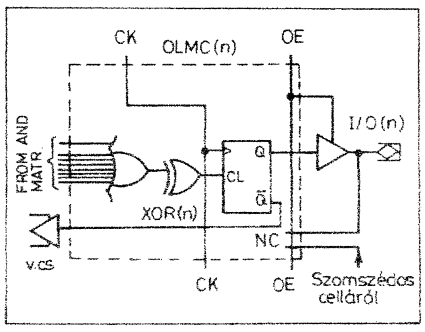
8. ábra

módon beprogramozott OLMC-k valóságos képét rajzoltuk fel.

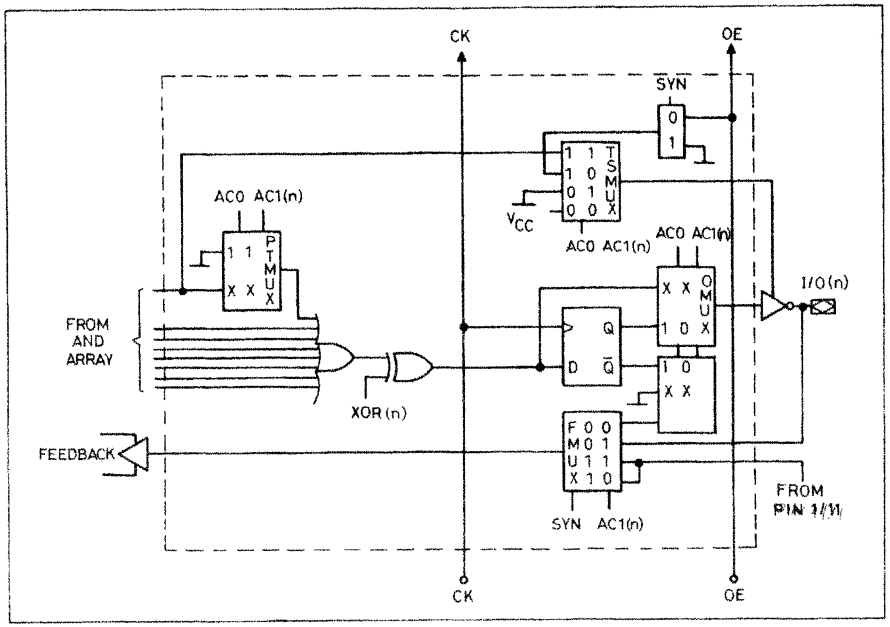
A **6. ábrán** egy kombinációs hálózatként megvalósított kimenetet láthatunk. Ez akkor áll elő, ha csak kombinációs kimeneteket hozunk létre az eszközben. Itt a kimeneti buffer tristate vezérlése az egyik PT (product term) felhasználásával történik.

A **7. ábrán** egy letiltott kimenet látható. Ilyet akkor hozunk létre, ha az adott lábat sem kimenetként, sem bemenetként nem használjuk.

A **8. ábra** nagyon hasonlít az 5. ábrához. A különbség csupán annyi, hogy ezt a kombinációs kimenetet egy olyan GAL-ban hoztuk létre, amelyben regiszteres kimenet is van. Ebben az esetben az 1. láb az órajel-bemenet, a 11-es (20V8-nál a 13-as)



9. ábra



5. ábra

reket, egy flip-flopot, buffereket és kapukat tartalmaz, amik részben a szomszédos makrocellától függetlenül programozhatók, más jellemzők tekintetében egy csokorrá fogja össze a nyolc kimenetet. Az OLMC felépítése az **5. ábrán** látható.

Mielőtt ennek taglalásába bonyolódnánk, néhány eddig mellőzött részlet tisztázása szükséges.

A 16V8 név a PAL világában szokásos jelölésmódot követi, jelentése a következő: az IC max. 16 bemenetűvé, illetve 8 kimenetűvé alakítható. A két feltétel csak addig látszik egymásnak ellentmondani, amíg fel nem hívjuk rá a figyelmet, hogy ez nem egyidejűleg értendő. Mivel a 20 lábból kettőt elhasználtunk a tápfeszültség csatlakoztatására, a megmaradó 18 a következőképpen osztható meg:

16 bemenet	–	2 kimenet
15 bemenet	–	3 kimenet
11 bemenet	–	7 kimenet
10 bemenet	–	8 kimenet

Ebből rögtön kitűnik, hogy kell lenni olyan lábaknak, amelyeket tetszés szerint akár bemenetként, akár kimenetként használhatunk. Az alábbi táblázat a 16V8 lábainak felhasználhatóságáról nyújt összefoglalót.

Kivezetés	Funkció
1.	bemenet vagy clock
2. ... 9.	csak bemenet
10.	GND
11.	bemenet, kimenet vagy tristate vezérlő
12.	csak kimenet
13.	bemenet vagy kimenet
19.	csak kimenet

Az OLMC-k a táblázatnak megfelelően lehetővé teszik, hogy a kivezetést akár kimenetként, akár bemenetként működtethessük. Ezen túl meghatározzák, hogy a kimenetként használt kivezetés szimpla kimenet legyen vagy netán egy flip-flop is legyen oda beépítve. Ekkor az 1. kivezetés feltétlenül az órajelbemenet kell legyen.

Az egyes OLMC-k némileg eltérhetnek egymástól, hogy pontosan a táblázatban feltüntetett funkciók megvalósítását tegyék lehetővé.

A továbbiakban megvizsgáljuk, hogyan tudja a GAL kihasználni az OLMC-k nyújtotta lehetőségeket.

(Folytatjuk)

Mi az a GAL? 3.

Komló Bertalan okl. villamosmérnök
Rétallér István híradásipari technikus (HA5BWH)

A terjedelem szabta lehetőségek nem engedik, hogy a GAL felépítésével tovább foglalkozzunk, ehhez inkább a kiválasztott típus adatlapja nyújt segítséget. Az előző részekben leírtak már elegendő támpontot adnak ahhoz, hogy eligazodjunk az adatlapon.

Mint már utaltunk rá, nagy jártasság és még nagyobb figyelem szükséges ahhoz, hogyha az ezekbe az eszközökbe kerülő logikai hálózatot papíron ceruzával akarjuk megtervezni. Nagyon ajánlatos valamilyen tervezőrendszer használata, mert a tévesztésekből eredő hibákat könnyű az utólagos ellenőrzéskor újra elkövetni.

Tekintettel arra, hogy – bár több rendszer forog közkézen – nem mindenki jut hozzá GAL tervezőprogramhoz, ismertetjük a GAL programozók által használt JEDEC formátumot. Ez a formátum lehetővé teszi, hogy egy egyszerű szövegszerkesztővel hozzunk létre a programozó számára alkalmas fájlt, lévén az egy ASCII karakterekkel felépített információtömb, ami megmondja, mely biztosítékokat kell „kiégetni" és melyeket érintetlenül hagyni. A tervezőprogramok szintén JEDEC formátumú fájlban hozzák a végeredményt. A példában bemutatott fájlt a később hivatkozott GAL programozó állította elő egy National Semiconductor gyártmányú, üres 16V8A típusú IC-ből. Lássuk az egyes részek értelmezését.

```
KoBRIs programmer, 18/07/1993 19:17
National Semiconductor's 16V8A converted
into JEDEC format

*B
*
QP20*
QF2194*
G0*
F0*
L0000 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
L0032 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
L0064 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
L0096 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
L1856 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
L1888 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
L1920 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
L1952 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
L1984 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
L2016 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
L2048 !!!!!!!!*
L2056 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
L2120 !!!!!!!!*
L2128 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
L2192 11*
C1F78*
*C0000
```

A fájlt egy szövegrész vezeti be, amelynek sem hosszára, sem tartalmára semmilyen megkötés nincsen. Akár több sorból is állhat vagy teljesen hiányozhat is, ettől a formátum még szabványosnak tekintendő.

E rész funkciója az, hogy a programok a fájl előállítására vonatkozó szöveges információkat valahol szabadon elhelyezhessék. A példa szerinti fájl a programozókészülék típusát, a kiolvasás idejét, a kiolvasott IC gyártóját és típusát tartalmazza, de ez csak minta, ettől teljesen eltérő adatok is lehetnek itt.

A továbbiakban egy [STX] karakter következik. (Emlékeztetőül: a karakter a Start.-Of-Text jel az ASCII táblázatban. Ügyeljünk arra, hogy ez már az első kötelezően létező elem, tehát olyan szövegszerkesztőt használjunk, amely képes elő is állítani a hexadecimális karaktereket – legalábbis a minimálisan szükséges STX [02H] és ETX [03] karaktereket!) A karaktert közvetlenül követi a fejrészszöveg, ami szintén kötetlen formátumú, akár több sorból is állhat. Ezt a mezőt – és mostantól fogva minden mezőt – kötelezően egy * (csillag) karakter (ASCII 2AH) zárja le. A mintában ez a mező egy üres sorból áll, amit soremelés zár le. Még ez a soremelés sem kötelező!

A következő négy mező opcionális. Teljes mértékben hiányozhatnak is, de ha vannak, akkor meg kell feleljenek az alábbiaknak:

QP – a Q utáni P arra utal (pin number), hogy az IC lábainak számát lehet megadni, decimális számként. Az egész string egybe írandó, egészen a lezáró csillagig, ezt nem szakíthatja meg sem betűköz, sem soremelés, sőt a Q előtt sem lehet semmi, ez kell legyen az első karakter.

QF – a Q utáni F arra utal (Fuse number), hogy az IC biztosítékainak számát lehet megadni, decimális számként. Ugyanazok a megkötések érvényesek, mint a QP mezőre.

G – a kiolvasást meggátoló security bit értéke. Ha 1-be állítjuk, engedélyezi a GAL programozónak, hogy automatikusan kiolvashatatlanná tegye az IC-t. Egyébként ugyanazok a megkötések érvényesek, mint a QP mezőre.

F – fuse default. Azt határozza meg, mi történjék az olyan biztosítékokkal, amelyekről esetleg külön nem történik intézkedés. (Ld. a következő mezőt.) Az F mezőre is ugyanazok a megkötések érvényesek, mint a QP mezőre.

Az **L** mező kötelezően létező eleme a táblázatnak. Ennyiben szigorú az előírás, azonban ez a mező kevésbé kötött. Az első karakter ugyan az L kell legyen, amivel egybeírva egy decimális szám következik, majd egy elválasztó karakter, de pl. az, hogy a szám hány digites, nincs megkötve, az 1 ugyanúgy jogos, mint a 001 vagy 00000000001. Az L utáni szám a biztosíték sorszámát jelzi. (Ld. júliusi szám, 2. ábra. A biztosíték sorszáma úgy értendő, hogy a 0,0 pozíció alatti bit az egyes, az az alatti a kettes stb. egészen a 63.-ig. A számozás újra fentről folytatódik, tehát a 0,0 ponttól jobbra van a 64. bit, ez alatt a 65., alul a 127. majd fentről indul a 128. A 31. oszlop alján a 2047. bit található.)

1. táblázat

SYN	ACO	AC1(n)	PTMUX	TSMUX	OMUX	FMUX
0	0	0	8 PT	aktív	komb.	nincs vcs.
0	0	1	X	TS	X	bemenet
0	1	0	8 PT	közös OE	reg.	reg. vcs.
0	1	1	7 PT	PTOE	komb.	komb. vcs./be
1	0	0	8 PT	aktív	komb.	1 és 11/13 be
1	0	1	X	TS	X	1 és 11/13 be
1	1	0	X	TS	X	1 és 11/13 be
1	1	1	7 PT	PTOE	komb.	1 és 11/13 be

A rövidítések magyarázata:

X	meghatározatlan
8 PT	a vagy kapura 8 product term kapcsolódik
7 PT	a vagy kapura 7 product term kapcsolódik
aktív	a kimenet mindig engedélyezve van
TS	a kimenet mindig tiltva van
közös OE	a kimenetet az OE (11-es/13-as láb) engedélyezi
PTOE	a kimenetet egy product term engedélyezi
komb.	kombinációs kimenet
reg.	regiszteres kimenet
vcs.	visszacsatolás
reg. vcs.	regiszteres visszacsatolás
komb. vcs./be	kombinációs visszacsatolás vagy bemenet
1 és 11/13 be	az 1-es és a 11-es (20V8-nál 13-as) lábak bemenetek

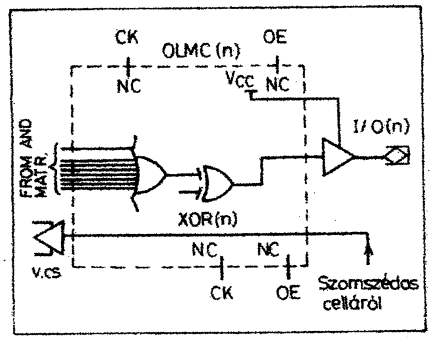
2. táblázat

SYN	ACO	AC1 n	AC1 m	PTMUX	TSMUX	OMUX	FMUX
0	0	0	0	8 PT	aktív	komb.	nincs vcs.
0	0	0	1	8 PT	aktív	komb.	szomszéd
0	0	1	0	X	TS	X	nincs vcs.
0	0	1	1	X	TS	X	szomszéd
0	1	0	0	8 PT	közös OE	reg.	reg. vcs.
0	1	0	1	8 PT	közös OE	reg.	reg. vcs.
0	1	1	0	7 PT	PT OE	komb.	komb. vcs./be
0	1	1	1	7 PT	PT OE	komb.	komb. vcs./be
1	0	0	0	8 PT	aktív	komb.	nincs vcs.
1	0	0	1	8 PT	aktív	komb.	szomszéd
1	0	1	0	X	TS	X	nincs vcs.
1	0	1	1	X	TS	X	szomszéd
1	1	0	0	X	TS	X	nincs vcs.
1	1	0	1	X	TS	X	nincs vcs.
1	1	1	0	7 PT	PT OE	komb.	komb. vcs./be
1	1	1	1	7 PT	PT OE	komb.	komb. vcs./be

A jelölések azonosak az 1. táblázat jelöléseivel, szomszéd alatt a „szomszédos" OLMC-hez tartozó láb értendő.

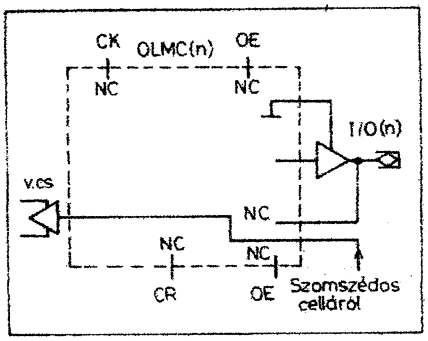
láb a kimeneti bufferek tristate vezérlését végzi. A kombinációs kimenet tristate vezérlése egy PT-ről történik.

A **9. ábrán** egy regiszteres kimenet látható. Ebben az esetben az 1-es láb a regiszterek órajelbemenete, a 11-es (20V8-nál a 13-as) láb a kimenetek tirstate vezérlését végzik.



10. ábra

A **10. ábrán** egy olyan kombinációs kimenet látható, amely csak kombinációs kimenetet tartalmazó eszközben lett megvalósítva. Az ilyen kimenet mindig aktív (a tristate buffer mindig engedélyezve van), a VAGY kapura nyolc PT kapcsolható. Az ilyen eszközben az 1-es és a 11-es (13-as) lábak bemenetek. Ugyanezt eredményezi a SYN=0, ACO=0 és AC1(n)=0 kombináció is.



11. ábra

A **11. ábrán** egy bemenetté kapcsolt OLMC vázlata látható. Figyelemre méltó, hogy az OLMC a szomszédos lábat kapcsolja bemenetté. Ennek az az oka, hogy az 1-es és a 11-es (13-as) lábakat a „szomszédos" OLMC-k kapcsolják normál bemenetté, illetve vezérlő bemenetté. Ez a struktúra alakul ki SYN=0, ACO=0 és AC1(n)=1 vezérlés esetén is.

Az OLMC-k vezérlésének ismertetése után rátérhetünk a GAL programozásának ismertetésére.

(Folytatjuk)

Mi az a GAL? 4.

Komló Bertalan okl. villamosmérnök
Rótallér István híradásipari technikus (HA5BWH)

Sorozatunk kényes ponthoz érkezett. Most már ideje lenne bizonyítanunk, hogy ez a sok mesebeszéd jó is valamire.

Lássuk, hogyan lehetne létrehozni az alábbi kapurendszert egy 16V8-ban **(1. ábra)**! Fordítsuk meg a kérdést és nézzük meg, hogyan valósította meg ezt a feladatot egy GAL fejlesztőrendszer. Már rendelkezünk azokkal az ismeretekkel, hogy ezt az ellenőrzést kritikus szemmel elvégezhessük.

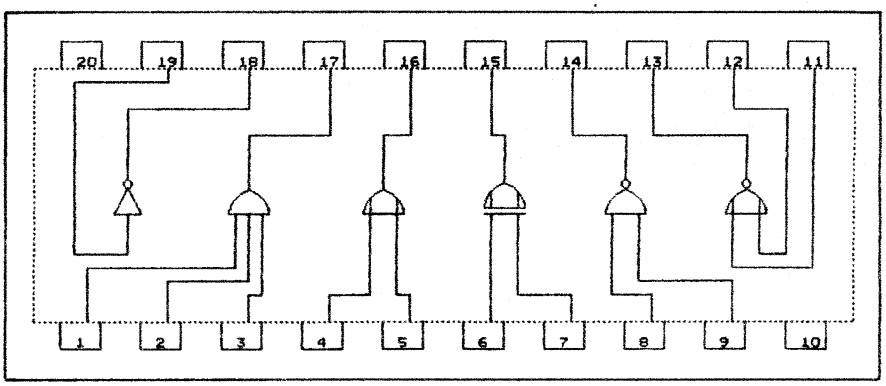
Előző számunkban ismertettük a JEDEC file felépítését, lássuk, hogyan is fest ez most a mi esetünkben **(lista)**.

A függőleges oszlopok számozása ott 0...31-ig terjed, ahogy a JEDEC file egy sorában is 32 bitet találunk. De mivel a file a biteket sorszámozza, a rajz pedig komplett bitsorokat, így minden bit-sorszámot osztani kell 32-vel, hogy megtudhassuk, hányadik sorban helyezkedik el. Jól követhető ez az előző számban közölt, üres 16V8-at bemutató JEDEC file segítségével. A bitek kezdőcímei 0, 32, 64, 96 stb., – amik a 32-vel osztás után a rajz szerinti 0, 1, 2, 3 stb. címsorokat adják közvetlenül. **(2. ábra)**

Egy pontosítást kell még megtennünk, hogy tovább mehessünk. A júliusi számban történt ugyan utalás arra, hogy a makrocellák kis mértékben eltérnek egymástól, de most már fontossá válik ezt pontosítani. Az akkor közölt OLMC csak a 2. és 19. lábra vonatkozik, a 13...18. lábakhoz rendelt cellák felépítése a **3. ábra** szerinti. Térjünk vissza a JEDEC filehoz. Az első bit címe esetünkben 256, ami a fentiek értelmében a 8. címsort írja le. (Vigyázat! A számsor 0-val indul, nem 1-gyel!) Ennek a 7. bitje 0, jelöljük be! Ha a függőleges oszlopot végignézzük, azt találjuk, hogy ide be van már keményen drótozva egy OLMC-ről jövő feedback ág, ami még ráadásul invertál is. De mi hajtja meg?

A file 2193. címén 0 van, ez az AC0 bit. A 18. lábhoz rendelt makrocellához tartozó AC1 bit 0 (címe: 2121), a szomszédos (a 19. lábhoz rendelt) cella AC1 bitje 1 (címe: 2120), így az FMUX a 19. lábat hozza a feedback buffer bemenetére. A buffernek viszont, mint az előbb felfedeztük, az invertáló kimenete programozott. Kezd derengeni a 18-19. lábak közötti inverter körvonala! És a sejtés valóra válik, kövessük csak végig a 18. lábhoz rendelt makrocella

```
A tokban levő signature: GATES
●
*
QP20*
QF2194*
G0*
F0*
L0256 11111110111111111111111111111111*

L0512 01010111111111111111111111111111*

L0768 11111111101110111111111111111111*

L1024 11111111111111111101101111111111*
L1056 11111111111111111011110111111111*

L1280 11111111111111111111111111101110111*

L1536 11111111111111111111111111101110*

L2048 11101011*
L2056 11100010100000100010101010100010110010100000000000000000000000000*
L2120 10000001*
L2128 11111111111111111111111111111111111111111111111111111111111111111*
L2192 10*
```



1. ábra

A sorszám utáni első számjegy az azon a címen található biztosítéknak az állapotát szabja meg. A 0 jelenti a programozandót, az 1 azt, hogy érintetlenül kell hagyni (szakadás). A következő számjegy a sorszám szerint következő biztosítékról intézkedik.

Az egyes számjegyek közé itt elválasztó karaktereket is iktathatunk, vagy akár egyetlen sorba leírhatjuk az egész biztosítékmezőt. Az elválasztó karakter akár space, akár soremelés is lehet. Teljesen legális formátumúak az alábbiak is. Figyeljük meg, hogy az L mező nem a 0 címen, hanem 256-nál kezdődik. A 0..255 címekre, a 288..511 címekre, illetve az 544-től felfelé eső címekre az F mező intézkedik. A jobb olvashatóság kedvéért a biztosítékokat tízesével csoportosítva írták le.

```
DEMO FILE
*BGAL16V8*
QP20*
QF2194*
G0*
F0*
L000256 1111111011 0111101110 1111111111 11*
L000512 1111111011 1011011110 1111111111 11*
C1571*
*C0000
Ugyanaz a tartalom más formában leírva:
DEMO FILE
*BGAL16V8*
QP20*
QF2194*
G0*
F0*
L000256 11111110
1101    1110
        1
     11
    0111111111111*
L000512 1111
                1
   11
    01110110111101111111111111*

C1571*
*C0000
```

Ez egy kissé szokatlan és célszerűtlen, de szabályosan felépített forma! A mező ugyanis a csillagig tart, nem tartalmaz mást, csak a megengedett 0 vagy 1 karaktereket és a megengedett kétféle elválasztó karaktert: a betűközt és soremelést – kétségtelenül egy kissé rendszertelen elrendezésben... A példa arra jó, hogy bemutassa, ezt a mezőt bátran rendezhetjük a kívánt formájúra, ha az elválasztó karaktereket helyesen használjuk. Külön kell szót ejtenünk az OLMC-k környezetében levő speciális biztosítékok címeiről. Ezeket az alábbi táblázatban foglaltuk össze.

Biztosítékcím		Funkciója
16V8	20V8	
0000–2047	0000–2559	biztosítékmátrix
2048–2055	2560–2567	XOR bitek
2056–2119	2568–2631	Signature bitek
2120–2127	2632–2639	AC1 bitek
2128–2191	2640–2703	OLMC OR kapura
2192	2704	SYN bit
2193	2705	AC0 bit

A következő mező kötelezően létező. Itt már újra érvényesek a QP mezőnél leírt szigorúbb megkötések.

C – a checksum mező. A fájl ellenőrző összegét adja, modulo 4 digites hexadecimális érték, ami úgy képződik, hogy bájtonként összeadjuk a biztosítékmező tartalmát:

	7. bit	6. bit	5. bit	4. bit	3. bit	2. bit	1. bit	0. bit
0. bájt bizt. cím	7	6	5	4	3	2	1	0
1. bájt bizt. cím	15	14	13	12	11	10	9	8
stb.								
stb.								
62. bájt bizt. cím	503	502	501	500	499	498	497	496
stb.								
stb.								
stb.								

A következő – szintén kötelező – tétel a már említett ETX karakter, aminek ASCII kódja 3. Vezérlőkaraktereket is kezelni képes szövegszerkesztők többnyire ^C-ként jelenítik meg. Közvetlenül utána egy 4 digites, az STX és ETX karakterek közötti összes karakterre vonatkozó ellenőrző összeg kell következzen. Ennek ellenére mi itt mégsem térünk ki a számítására, mert bár a mezőnek kell léteznie, a checksum számítása nem kötelező. Ez esetben 0000-t kell ide írjunk.

Vannak más, itt nem említett mezők is a JEDEC file kelléktárában, azonban azok opcionálisak és kevésbé fontosak számunkra. A GAL programozó nem is veszi figyelembe, csak az itt leírt mezőket kezeli, így részletezésükre nem térünk ki.

Ez úton mondunk köszönetet Fórián Sándor barátunknak, aki értékes tanácsaival járult hozzá a fejlesztéshez.

Ajánlott irodalom:

- ELEKTOR 1992/3 Von Dipl.-Ing. M. Nosswitz: GAL programmer
- ELEKTOR 1992/3 Von Daniel Gembris: GALs programmierung und einsatz
- SGS-THOMSON GAL16V8 és GAL20V8 adatlapok

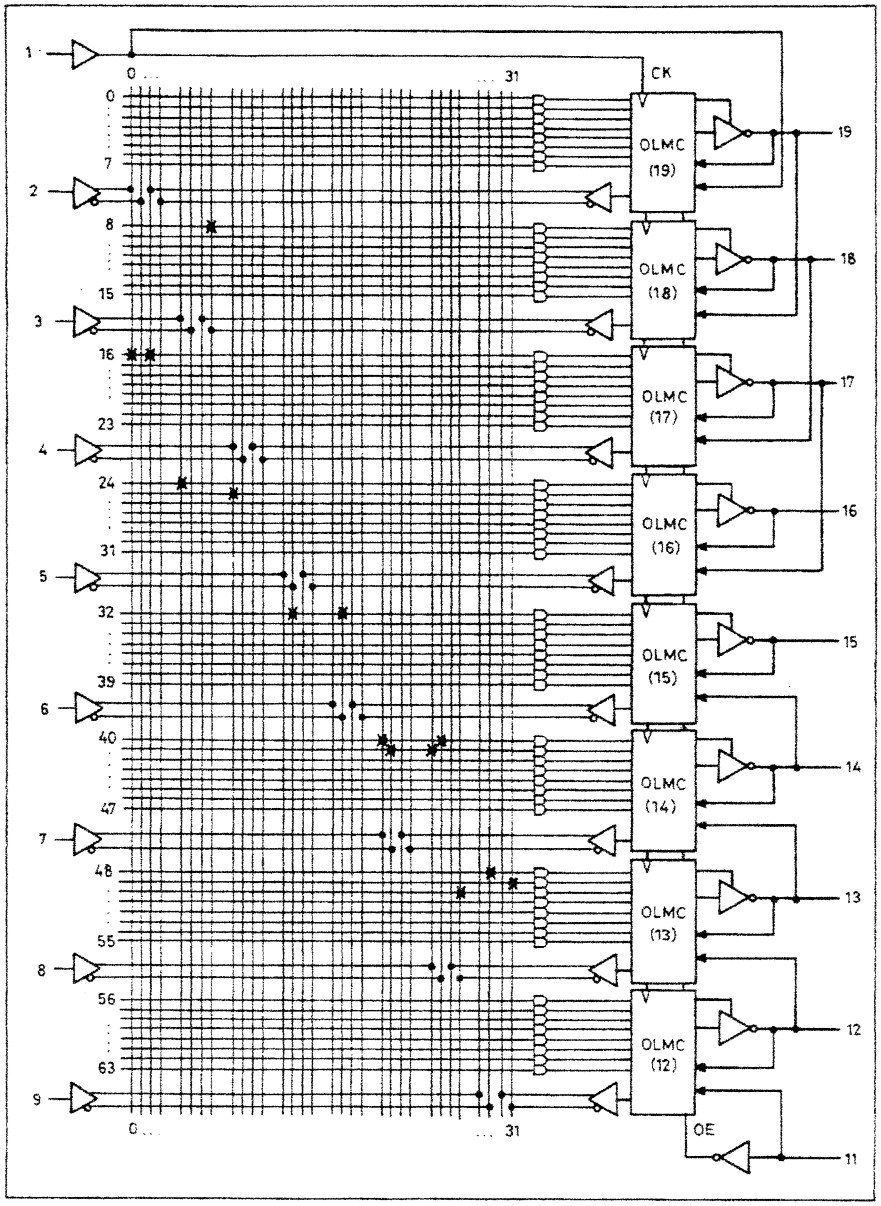
belsejében: A VAGY kapun, polaritást vezérlő XOR kapun át az OMUX jelű multiplexerre jut, innen már csak egy ugrás a „véletlenül" éppen engedélyezett tristate-buffer és már ott is vagyunk a 18. lábon, mint kimeneten.

Hasonló úton, a feedback buffer használatával jutunk el az 1. lábtól, pontosabban a buffer nem invertáló kimenetéről a 2. függőleges sorra. (Vigyázat! A számozás 0, 1, 2 stb.) A file az 512-vel kezdődő biztosítéksort írja le, ez az 512/32=16. vízszintes sor a rajzon. E sornak a 0. bitje is programozott, amiből azonnal látni, hogy az maga az IC 2. bemenete, meghajtón át. Sőt, a 4. bit is programozott, ami maga a 3. láb, bufferen keresztül. És talán még emlékezünk a júliusi számban taglalt, furcsa ÉS kapukra: e három pont ÉS kapcsolatot alkotva megy a 17. lábhoz tartozó OLMC-be, majd onnan ki, magára a 17. lábra. Megvan az ÉS kapu, is.

A következő, a VAGY kapu nagyon furcsa az első ránézésre. A 768. bitcím a 24. vízszintes sorra utal. Ennek 9. és 13. bitje programozott és ez egy ÉS kapcsolatot jelent, nem VAGY-ot! Mit csinált ez a tervezőrendszer? Ha már itt tartunk, nézzük meg! Bemenetként a 4. és 5. lábat használja, amint azt elvártuk, de invertálja őket! A kimenet aztán a 16. lábra érkezik, még ez is rendben volna, de ebben meg az a furcsa, hogy az XOR bit itt 0 szintű, holott mostanáig mindig 1 volt. (Ld. a 2053. bitet a JEDEC file-ban.) A kimenet is invertálni fog! Erre már beugrik, miről is van szó: csak annyi történt, hogy a program valamiért a szája ízének jobban megfelelő azonosság szerint tervezett:

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

Természetesen „egyszerűbb" lett volna, ha a 24. soron a 8. bitet, a 25. soron pedig a 12. bitet programozzuk. Ezzel a bemeneteken nem lenne invertálás és az OLMC más-más sorára érkezvén, a belső VAGY kapu foghatná össze a függvényt. Ekkor természetesen az XOR bit is maradna 1-ben. Próbáljunk csak így módosítani a file-t! Bár belül egy kicsit más lesz, az IC-t black boxként kezelve ugyanannak látszik a lábak felől. Számunkra a második megoldás a kézenfekvő, a program viszont a takarékosság jegyében gondolkodik: csak a 24. sort használta fel, míg a



2. ábra

„kézi" tervezéssel a 25. sort is elherdálnánk. Itt és most ugyan mindegy lenne, csak meg akartuk védeni a program szempontjait is, miért azt az utat választotta, amelyet bemutattunk: takarékosságból.

A kizáró VAGY kapuval kapcsolatban csak azt kell felidéznünk, hogy

$$Y=(A\cdot\overline{B})+(\overline{A}\cdot B).$$

Az invertálást a 6. és 7. lábakról jövet meg lehet oldani. Azt már nagy rutinnal vesszük észre, hogy a 32. sor egy ÉS kapcsolat az invertált 6. láb és a 7. láb között. Ugyanígy jön létre egy másik ÉS kapcsolat a 6. és az invertált 7. láb között, de az a 33. soron. Mindkét ÉS kapcsolatot az OLMC-ben levő VAGY kapu fogadja, ezzel teljesült is az egyenlet.

A NAND kapu megvalósítása triviális: a 40. soron megvalósuló ÉS kapcsolatot a kimeneten az XOR bittel kell csak meginvertálni. Ld. a 0 szintű 2053. bitet a JEDEC file-ban.

A NEM-VAGY kapu az

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

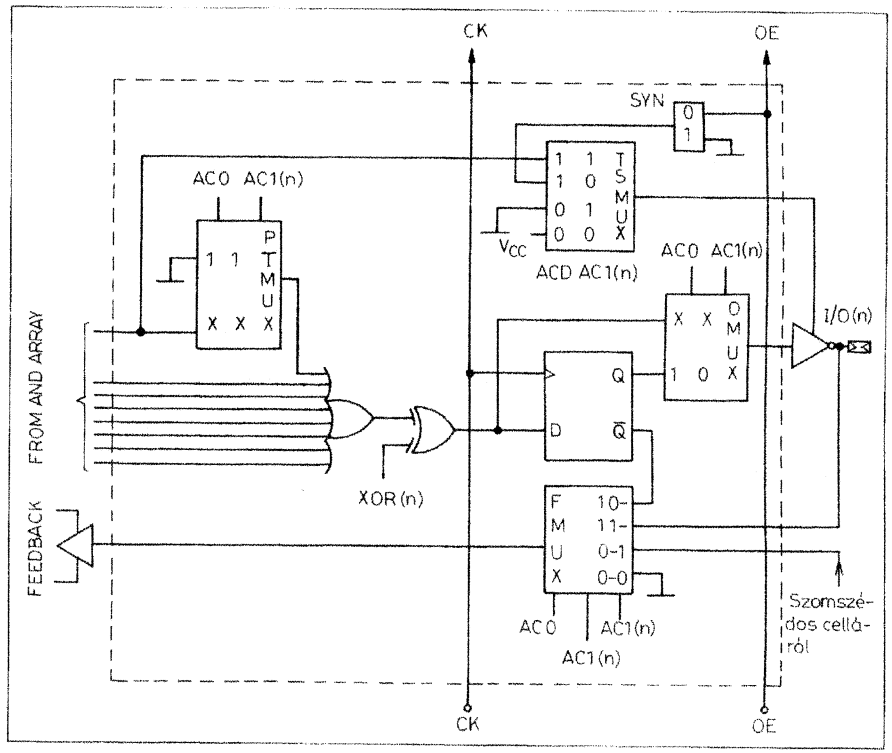
azonosságot veszi figyelembe. Vegyük észre, hogy ez ugyanaz a trükk, amelyet a 24. sor kapcsán már taglaltunk. A különbség csak annyi, hogy az XOR bit nem írja elő a kimenet invertálását.

Továbbra is úgy véljük, hogy a célszerű mód valamelyik GAL fejlesztőrendszer használata, de a fentiek ala-

pos elsajátításával nem boszorkányság JEDEC file előállítása szövegszerkesztővel sem. Az általunk gyártott, szeptemberben közölt GAL programozó nem ellenőrzi egyik checksum-ot sem, abból a megfontolásból, hogy ellenőrzi azt úgyis a host gép, így ezek kiszámolásával ne is vesződjünk.

Célszerű eljárás a JEDEC file „kézi" előállításakor az alábbi módszer:

Helyezzünk a foglalatba egy méretben az alkalmazásnak megfelelő tokot (a gyártó közömbös, de a méret legyen azonos). Töröljük ki, majd mentsük tetszőleges néven lemezre. Ha a program indításakor azonnal kiírjuk a puffert, mielőtt bármit betöltenénk vagy beolvasnánk egy IC-t, akkor még IC-t sem kell behelyeznünk, minden bit az üres állapotnak megfelelő 1 szintű még a program jóvoltából. Az így nyert file-t mentsük biztonságos helyre, majd előbb munkafile-t másolva, megfelelő „blankettához" jutunk, amit csak ki kell tölteni, nem kell a biztosítékok sorszámozásával vesződnünk. Mind 16V8-ra, mind 20V8-ra állítsuk elő ezt a „mesterfile"-t, aztán már csak türelem és



3. ábra

koncentrálás kérdése, míg megszületik az első saját tervezésű IC.

Sorozatunk ezzel véget ért, reméljük, sokak érdeklődését felkeltették a GAL által képviselt új világ lehetőségei.