Szines PRIMO adapter - működési leírás

1. Tervezési szempontok:

A PRIMO tervezésekor kifejezetten fekete/fehér konstrukciót igényeltek, ezért a későbbi szines változat létrehozása bizonyos trükkök alkalmazását igényelte. Mindenképpen olyan megoldást kellett választani, ami a már meglévő PRIMO-ban is alkalmazható, minél kevesebb ponton csatlakozni a PRIMO-ban már meglévő jelekhez. Így lehetne tömören megfoglamazni a feladat egyik leglényegesebb peremfeltételét. Ez eleve kizárta a képmemóriához való hozzáférést. Maradt helyette a sorosított video információ "megcsapolása". A PRIMO-ban átkötés révén 256x256 raszteres kép megjelenítésére is lehetőség van, így tulajdonképpen 8Kb-nyi információ áll rendelkezésünkre a fenti módszer alkalmazásával. A képméretet csökkenteni nem szabad, így a szinezésre kb. 2KB-os többlet információnk van (a 6KB-os feket/fehér információ mellett). A pontonkénti szinezés tehát kizárt (mivel a TV-nél a színinformáció maximális frekvenciája 1 MHz, a pontonkénti színezés elvetése nem jelent túlzott veszteséget), marad a színezhető képelem növelése. Jelöljük ennek méretét mxn -el; ezen a mezőn belül csak két különböző -előtér/háttér-szín alkalmazható a világosság -a továbbiakban "y" információ-függvényében. Legyen egy képelemhez rendelt színkód "W" bit hosszúságú, ekkor a következő egyenletet írhatjuk fel:

$\frac{2Kb}{W} = \frac{6Kb}{mxn}$ vagyis m x n =3W. Ennek a legkézenfekvőbb megoldása m=n=W=3 vagyis 3x3-as képelemhez 3-bites színkód tartozna, vagyis az "y" információt is felhasználva egyszerre 2^{3+1} =16 szín lehetne a képernyőn. Ezt a megoldást (bár deszkamodell szinten elkészült) végűl is elvetettük, a 3-bit

a 8-bites gépben való kezelésének nehézségei miatt. Az egyenlet helyett a felhasználók táborához fordultunk és így a következő képelem-méretek alakultak ki:

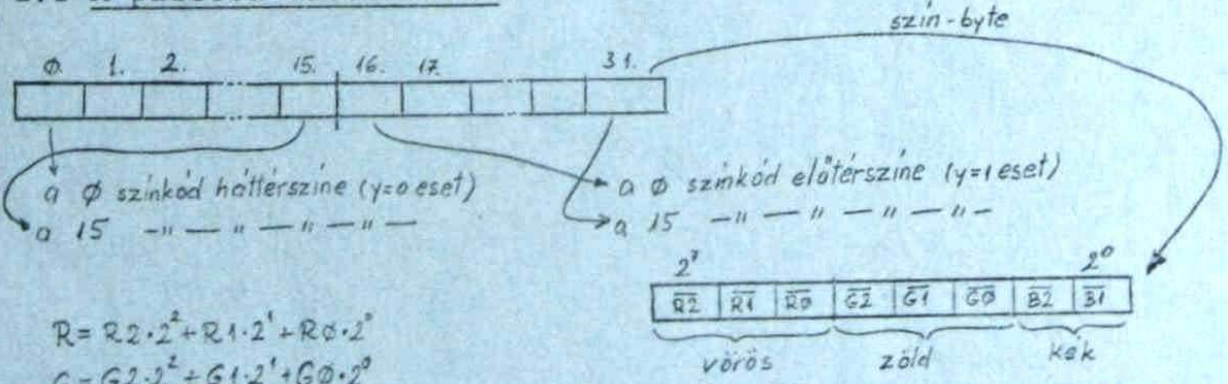
a, 4x4 gyors kezelhetőség (grafika stb.) miatt
b, 6x6 a karakterméretehez való alkalmazkodás miatt
c, 6x9 teledata/teletex jellegű alkalmazások esetén minimum 25x40 karakter megjelenítése is feltétel.

E képelem méret növekedést most már a színkód hosszának növelése is követhette, így (elsősorban a 8 bites struktúra miatt) W=4 -re módosult, azaz egyidőben 32 szín megjelenítésére van mód. A színkód és a konkrét színek összerendelését 32x8 bites memória végzi. Ennek töltésére a 8 KB-ból lefoglaltunk 3x32B-ot (azaz egyidőben 3 palettányi információ áll előkészítve). A fent említett 3 üzemmód beállítására, a három palettából az aktív(ak) kiválasztására illetve érvényességi idejük megszabására még 32B-ot lefoglaltunk. Így a 256 sor helyett csak 252 sornyi információ maradt a szín és az "y" információkra.

2. A képmemória kiosztása:

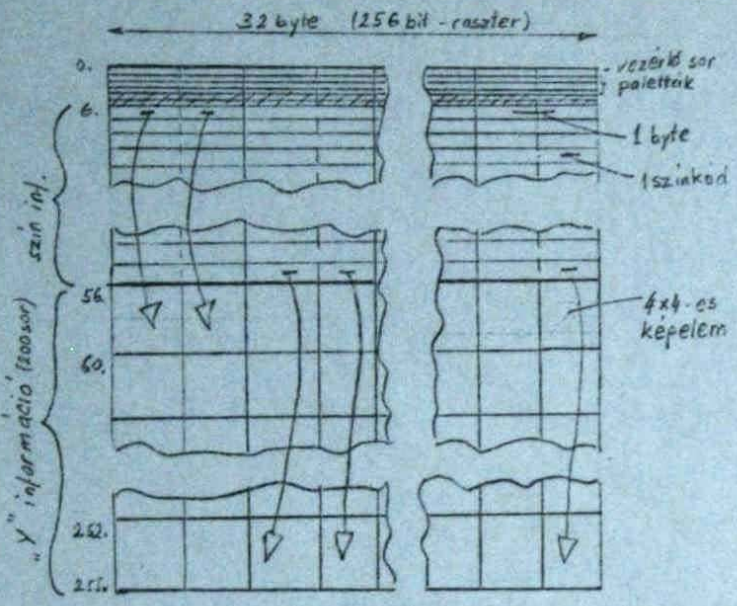
Ezt az 1. ábra elég részletesen szemlélteti, de a paletták értelmezésére és a színkód/y hozzárendelésre az egyes üzemmódoknál még érdemes kitérni.

2.1 A paletta értelmezése:



$R = R2 \cdot 2^2 + R1 \cdot 2^1 + R0 \cdot 2^0$
$G = G2 \cdot 2^2 + G1 \cdot 2^1 + G0 \cdot 2^0$
$B = B2 \cdot 2^2 + B1 \cdot 2^1$

2.2 A 4x4-es üzemmódhoz tartozó színkód/y összerendelés:

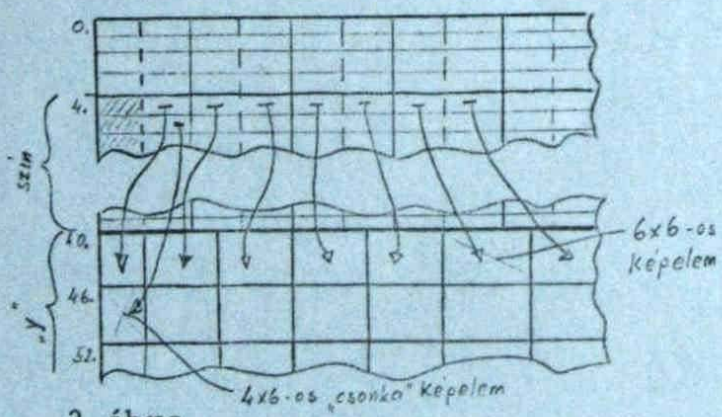


2 ábra

Mint a2-es ábrán látható, az összerendelés teljesen "normális". 4 "y" sorhoz 1 szín - sor tartozik, azaz 5 sorból áll egy szines képelem sor. Ebből adódik INT (252/5)=50 képelemsor lehetséges, a raszterméret pedig 200x256.

2.3 A 6x6-os üzemmódhoz tartozó színkód/y összerendelés:

Mivel 256 nem osztható 6-tal (256=42x6+4) az első képelem oszlop szélessége csak 4 raszter (A PRIMO karakterei az első 4 rasztert nem használják ki.). A 42 karakternyi képelem színét 21 byte adja meg, az első "csonka" képelemét egy fél-byte. A maradék 10,5 byte kihasználatlan. (Kihasználásuk okozta bonyolítás nem térülne meg.)



3 ábra

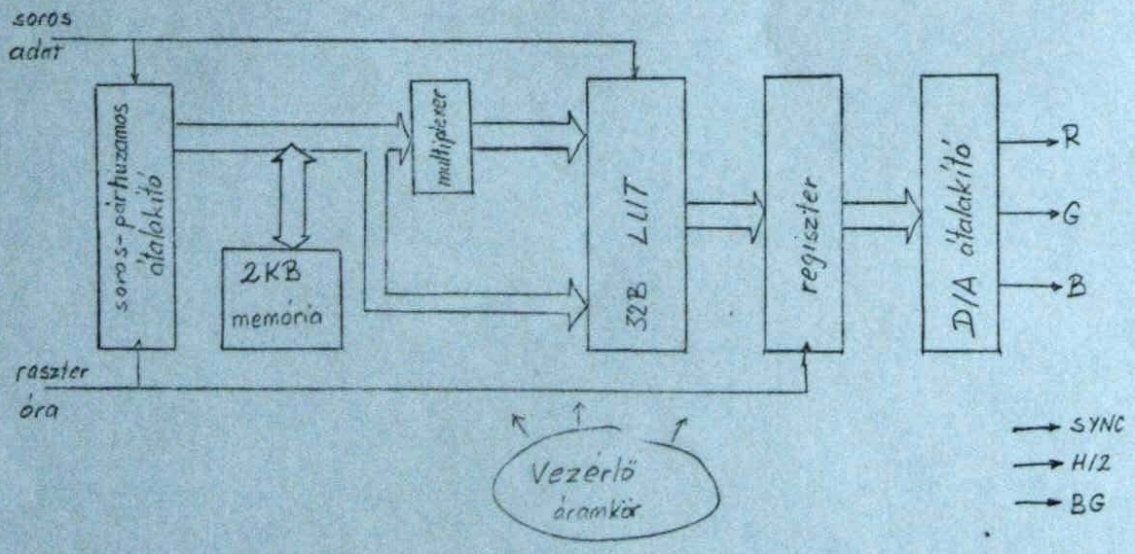
Így 6 y sorhoz 1 színsor tartozik, azaz 7 sor ír le egy képelem sort. A képelem sorok száma tehát 252/7=36. Az "y" sorok száma 6x36=216 a karaktersorok száma pedig 36/2=18.

2.4 A 6x9-es üzemmódhoz tartozó színkód/y összerendelés:

A soronkénti megfeleltetés megegyezik a 6x6-os esettel. A képelem sorok száma INT(252/(9+1))=25 az y sorok száma 9x25=225, a két maradék sor helye a 4x4-es esettel egyező.

3. A színes adapter működési elve:

Az y információkat megelőző információkat (maximális esetben - 4x4 - 56x32B=17.923(kisebb mint) 2Kb)-bytos formában beírjuk egy 2Kb-os RAM-ba, majd az "y" információk megjelenésekor a megfelelő színinformációkat kiolvassuk a megfelelő ütemben. A kiolvasott byte-okat egy multiplexer "tördeli színkódra és ez az "y" információval együtt a LUT címbemeneteire kerül. A kiolvasott adatokat egy raszterfrekvenciával járó regiszter tárolja majd egy *D/A* konverter szolgáltatja az R, G, B jeleket. Az üzemmódok értelmezését és a LUT feltöltését tulajdonképpen a RAM-ból képkioltás alatt végezzük, természetesen ennek az egy képidőnyi csúszásnak nincs semmi jelentősége.



4 ábra

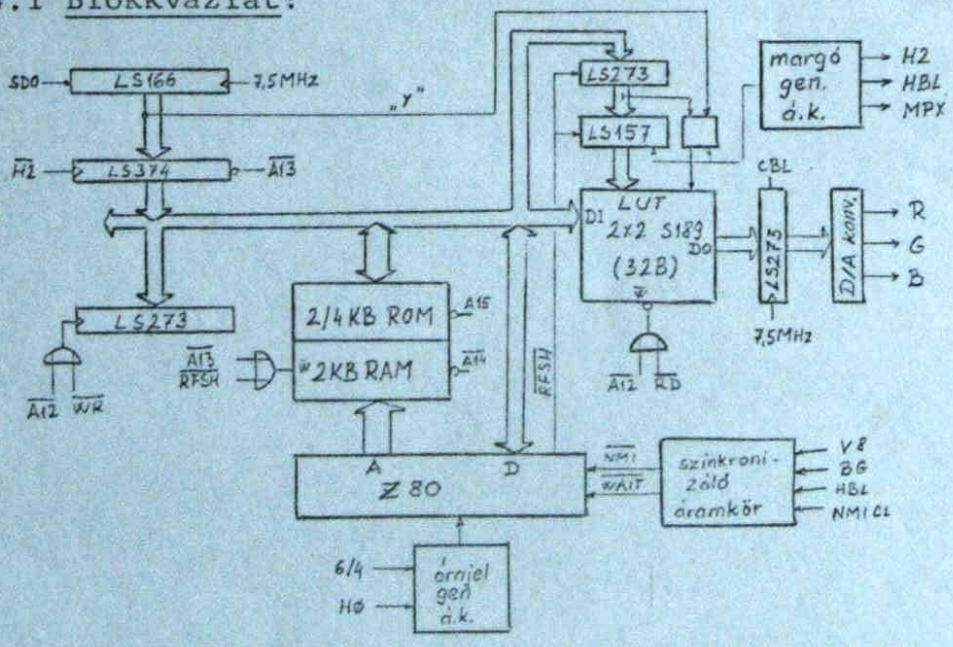
Közismert, hogy e struktúrát működtető vezérlő, címző stb. áramkör megvalósítható diszkrét hardware elemekkel (huzalozott logika) vagy mikroprogramozott vezérlővel. Ez utóbbi elég drága, a PRIMO eddigi hagyományaitól eltérő megoldás, az első bár a tervezése papíron elkészült rugalmasságával vannak bajok.

Volt viszont már egy előzetes kísérletünk, melyben a PRIMO kettős hozzáférésű memóriáját a számlálólánc helyett, egy második Z80-as processzorral olvastattuk ki a megjelenítéshez.

Ez a processzor generálta a szinkron stb. jeleket is. A tapasztalat azt igazolta, hogy bár lényeges anyagi megtakarítást e módszer nem hoz - ezért sem volt érdemes a gyártásba még bevezetni - meglehetősen rugalmas, sok hasznos lehetőséget biztosító megoldás. Egy ilyen rendszer programozása viszont igen kemény dió. Végül mégis ezt a megoldást választottuk.

4. A hardware megoldás részletes ismertetése:

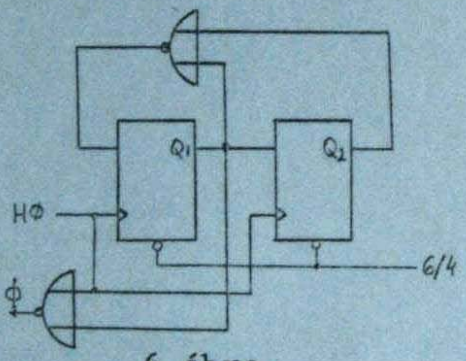
4.1 Blokkvázlat:



5. ábra

4.2 A processzor órajele:

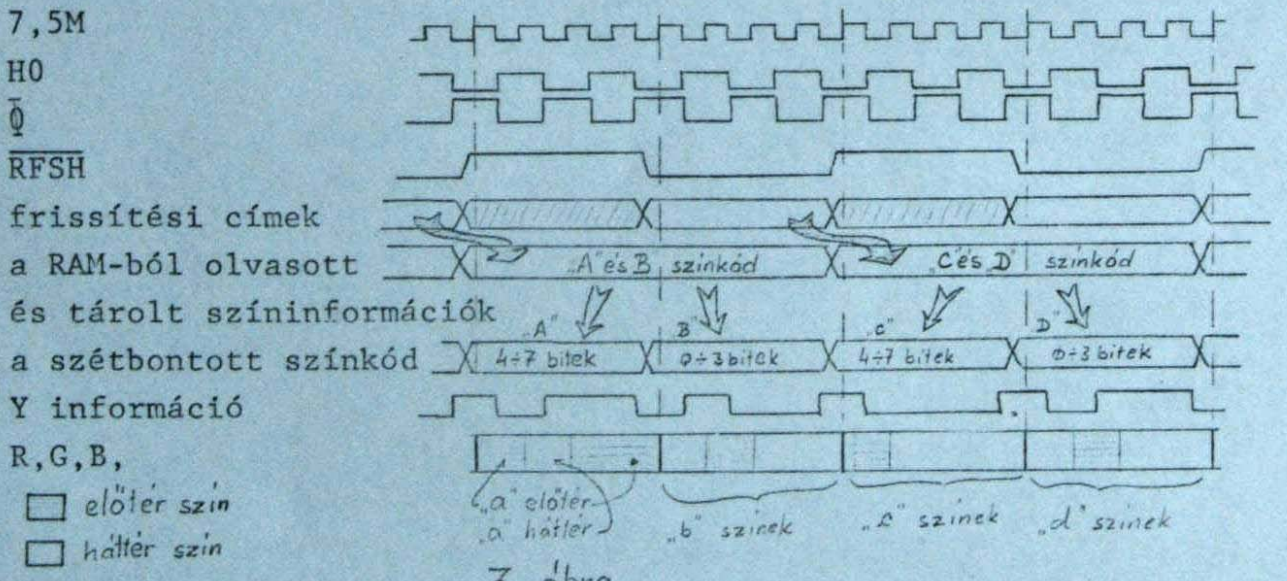
A megvalósítás egyik alapötlete az, hogy ha a processzor csak M1-es (4T-s) utasításokat hajt végre, úgy a frissítési ciklusok alatt a címek folyamatosan növekednek, ezt a színkódok címeinek előállítására használjuk. Ilyenkor a RFSH jel 50%-os kitöltési tényezőjű, ezt a kiolvasott és RFSH-val mintavételezett színinformáció-byte színkódokra való szétválasztására használjuk. Azt kell elérni, hogy egy ilyen periódus két színelem megjelenítési idejével legyen egyenlő. Erre való az órajel előállító áramkör:



6 ábra

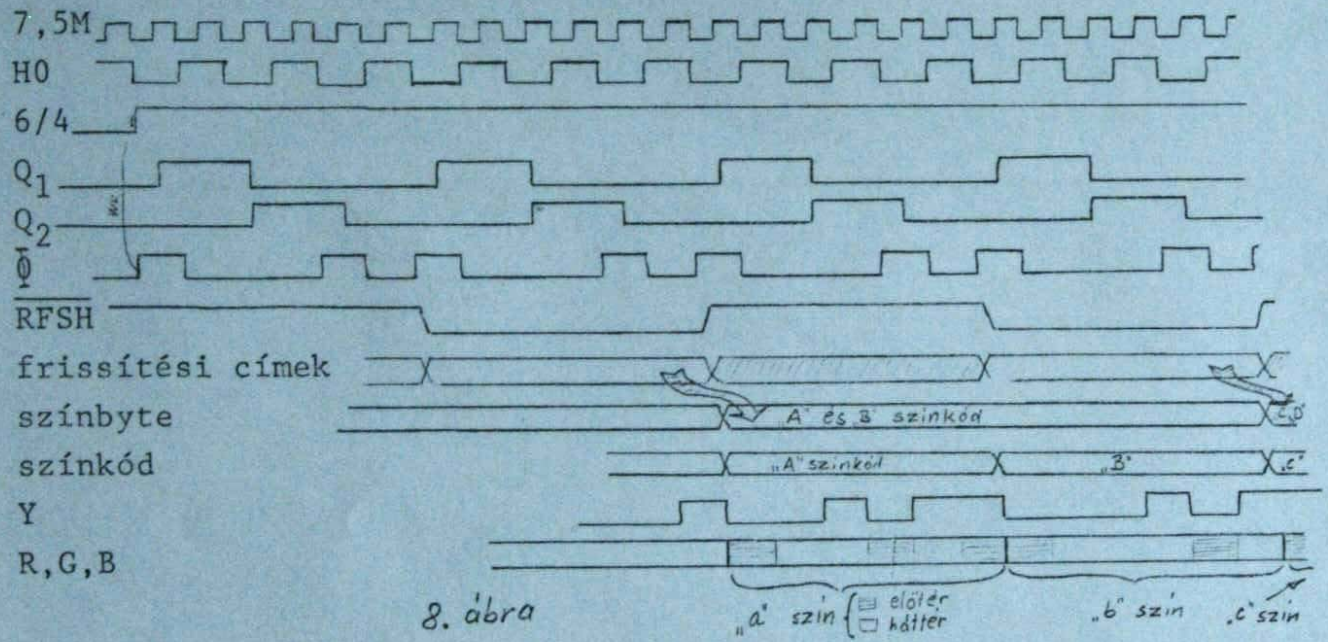
H0 ill.H2 a raszter oszlop számláló 2^0 ill. 2^2 helyértékű bitje (a PRIMO-ból)
6/4 =0 4x4 üzemmód
6/4 =1 6x6 v.6x9 üzemmód.

Látható, hogy 4x4 üzemmód esetén $\overline{Q}=\overline{H0}$, ilyenkor a színes megjelenítés az alábbi időzítés szerint történik:



7. ábra

Természetesen a dolog azért ment ilyen egyszerűen, mert 1 szín bytehoz 1 "Y" byte tartozott. A 6-os szélességű színelem esetében 1 színbyte-hoz 12 "Y" bit azaz 3/2 byte tartozik. Ilyenkor a processzor órajelét a H0 3/2-ed részének kell vegyük, ez szerencsére lehetséges mert úgy az M1, mint a RFSH ciklusok két-két órajelűek.
Ebben az esetben az előző ábra a következő képpen módosul:

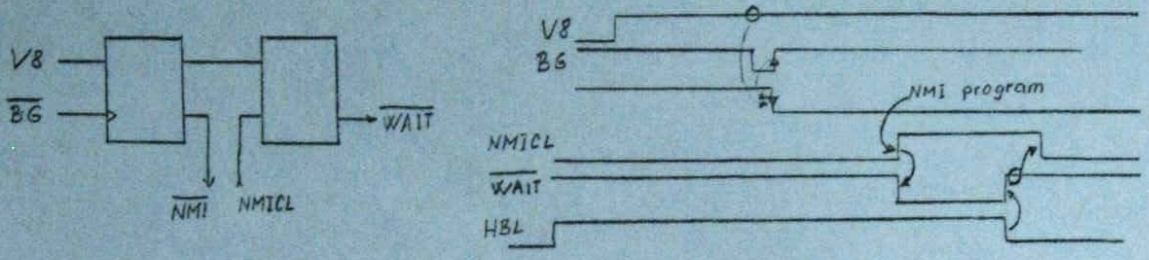


8. ábra

4.3. Szinkronizáló áramkör

A szines adapter processzorában futó program írja be a RAM-ba a színinformációkat, veszi elő ezeket a megjelenítéssel összehangban, generálja a szinkron, kioltó stb. jeleket. Nyílvánvaló, hogy ez egy erősen kötött futásidejű program kell legyen, de a PRIMO-val történő összeszinkronizálás is elengedhetetlen. Az is világos, hogy képenkénti szinkronizálást kell biztosítani, így a programnak egy kis segítséget is nyújtunk. A képkioltás éle (V8) lenne a megfelelő jel, de ez egy aszinkron CMOS számláló kimenete,

időbeli szórása igen nagy lehet. Ezért ezt a sor egy adott időpillanatához (BG - színsegédvivő kapuzó jel) kötöttük. Ez az esemény egy nem maszkolható megszakítást okoz (NMI) jelezve a processzornak a képmegjelenítés végét. (Ezután a képmargó, képkioltás, képszinkron stb. következik.)



9 ábra

A szinkronizálás befejezésére szükség van (lehet) WAIT ciklusok beiktatására, ha a program ciklus és a képciklus méreteit nem lehet pontosan összeegyeztetni. Az NMI program elég pontosan "sejti" hol ér véget a sorkioltás (HBL) az NMI kezdetéhez viszonyítva. Néhány órajelciklussal ez előtt kiadja az NMI CL jelet mely hatására WAIT-be ragad, ennek megszünése a sorkioltás végén lesz.

4.4. Az adatpályák vezérlései

Mivel a legnagyobb összefüggő terület amit címeznünk kell az 4KB, a ROM - bár a program 2KB-ban is elfér, érdemes megtartani ezt a lehetőséget, 12 címvonal szükséges a címzésre. A processzor 16 címvonallal rendelkezik (A0-A15) a fennmaradó 4 címbitet az adatpályák vezérlésére használjuk, az alábbiak szerint:

fogadóregiszter kimenet engedélyezése: $\overline{A13}$
memória kiválasztás: $\overline{A14}$
memória kimenet engedélyezése: A13+$\overline{MRQ}$

memória írás: $\overline{A13}+\overline{RFSH}+\overline{MRQ}$
PROM kiválasztás: A15
PROM kimenet engedélyezés: $\overline{MRQ}$
vezérlő regiszter írás: $\overline{A12}+\overline{WR}$
LUT írás: $\overline{A12}+\overline{RD}$
színbyte beíró jel: $\overline{RFSH}$

4.4.1 Memóriába írás a fogadóregiszterből

A fogadóregiszterbe a byte-okat a H2 mintavételezi, ennek kimenete az A13=1 esetben aktív. A program ilyenkor szintén 4T-s utasításokat hajt végre, az időzítések a 4x4-es megjelenítéssel azonosak. A memória címeit a frissítési ciklusok generálják.
A frissítési A15÷A12 címek értékei 1110 kell legyenek, ezáltal tiltva van a PROM és a RAM kimenete, engedélyezve a fogadóregiszter kimenete és a memóriába történő írás. Az A12=1 miatt a LUT illetve vezérlő regiszter tartalma nem változik.

4.4.2 Vezérlőregiszter írás

Ez a művelet általában belső regiszterbe előkészített értékek gyors egymásutáni kiküldését jelenti, ezért tiltani kell az adatforrásokat: A15÷A12=1011 és WR.
A vezérlőregiszter bitjeinek értelmezése a következő:

2^0 - CLB - összetett kioltó jel - közvetlenül az R,G,B regiszter kimenetét nullázza
2^1 - $\overline{CSY}$ - összetett szinkronjel

2^2 - BRD - képmargó (azaz alsó-felső margó) generálásra szolgál
2^3 - ST4 - szabad bit, esetleg programok PRIMO-ba letöltésére szolgálhat
2^4 - MPX - LUT íráskor a multiplexer címkimenetei a szín-byte alsó öt bitjéből kell származzanak
2^5 - NMICL - a WAIT bemenet aktiválását váltja ki
2^6 - $\overline{BG}$ - a sorkioltás alatti színsegédvivő kikapuzására szolgáló jel
2^7 - 6/4 - képelem méret szélesség

Itt jegyezzük meg, hogy a színes kóder számára szükséges H/2 (fél sorfrkvencia) jel a szinkron jel leosztásából származik.

4.4.3 Memória olvasás - színkód előállítás

Az A15÷A12 bitek állása 1100 a frissítés alatt. Ezáltal a memória kiválasztása és a kimenet engedélyezése megtörténik, a frissítési jel pedig beirja a színkód-párt a szín-byte regiszterbe. Az időzítésekről már szó esett a 4.2 bekezdésben.

4.4.4 LUT írás

A beírandó információt a RAM-ból csak képkioltás alatt írhatjuk be a LUT-ba, hisz ezen kívül folyik a kép vagy margó megjelenítése. Beíráskor a szín-byte regiszterbe (felső öt bitjére) a LUT címe kerül - amit megfelelő PROM címre mutató frissítési ciklus vesz elő a PROM-ba előkészített adatként. Frissítéshez tehát az $A15 \div A12_{REF}$ =0010 címkombinációt kell beállítani. Maga a

LUT írás úgy történik, hogy a memória megfelelő címéről ténylegesen olvasunk úgy, hogy A12=1 ami az olvasott adatot rögtön a LUT-ba írja. Az A15÷12 bitek tehát olvasáskor 1101. Nyílvánvaló, hogy olvasáskor a REF jel úgy vezérli a multiplexert, hogy a felső négy bitet engedi át, gondoskodni kell viszont a multiplexer engedélyezéséről és az 5. címbit áteresztéséről. Ezt tárgyalja a következő fejezet.

4.5. LUT cím, multiplexer, margó (border) generálás

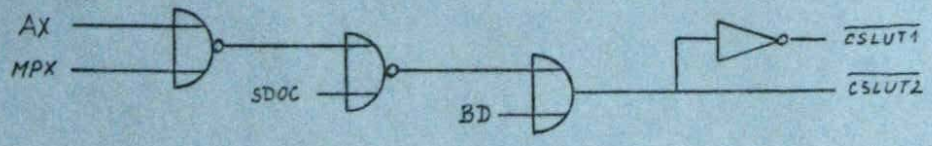
A LUT címek tulajdonképpen négy forrásból kell előálljanak:

Megjelenítés alatt ha	$\overline{RFSH}$=1	1, színbyte felső fele +"y"
	$\overline{RFSH}$=0	2, színbyte alsó fele +"y"
margó alatt	BD=1	3, 00000
LUT írás alatt		4, színbyte felső 5 bitje

(A margó tulajdonképpen a 0 színkód háttércíme)

Látható, hogy négy címbit előállítására megfelel egy LS157-es multiplexer, ahol a multiplexelést maga a RFSH jel vezérli és kimenetét a BD=1 eset tiltja.

Az ötödik címbitet a következő hálózat állítja elő:



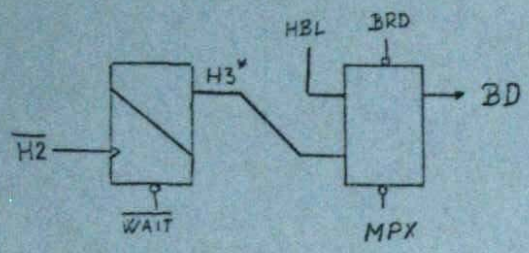
10 ábra

A LUT tulajdonképpen 4 db. 16x4 bites RAM-ból áll, ezért az eddig említett 5. címbit tulajdonképpen a két-két IC kiválasztó kimenetét vezérli. E logikai hálózat kihasználja azt a tényt, hogy a LUT írás (MPX=0) képkioltás alatt történik, így az SDOC videojel alacsonyszintű.

A BD jel megjelenése mindettől függetlenül a 0.IC-tömböt

választja ki. A AX jel tulajdonképpen a színbyte regiszter 2^3 helyiértékű bitje.

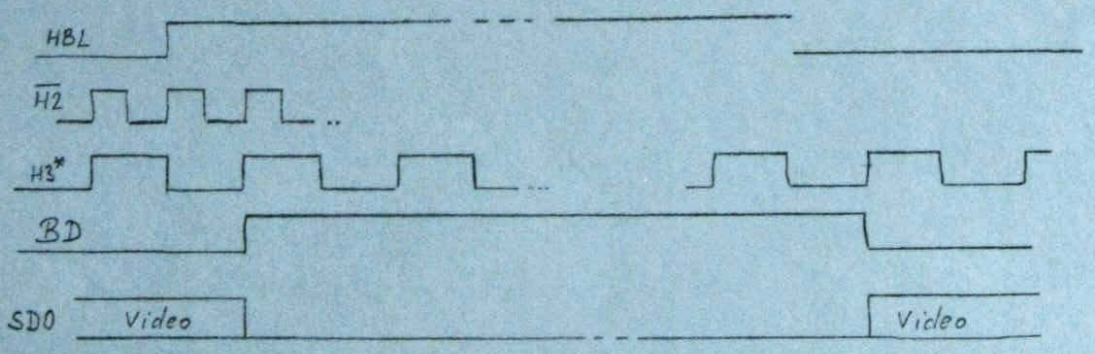
A margót (BD jel) a következő áramkör valósítja meg:



11 ábra

A BRD vezérlőregiszter egyik bitje az aktív valamint a képkioltáson kívüli sorok alatt végig alacsony, így a BD=1 állapot alakul ki.

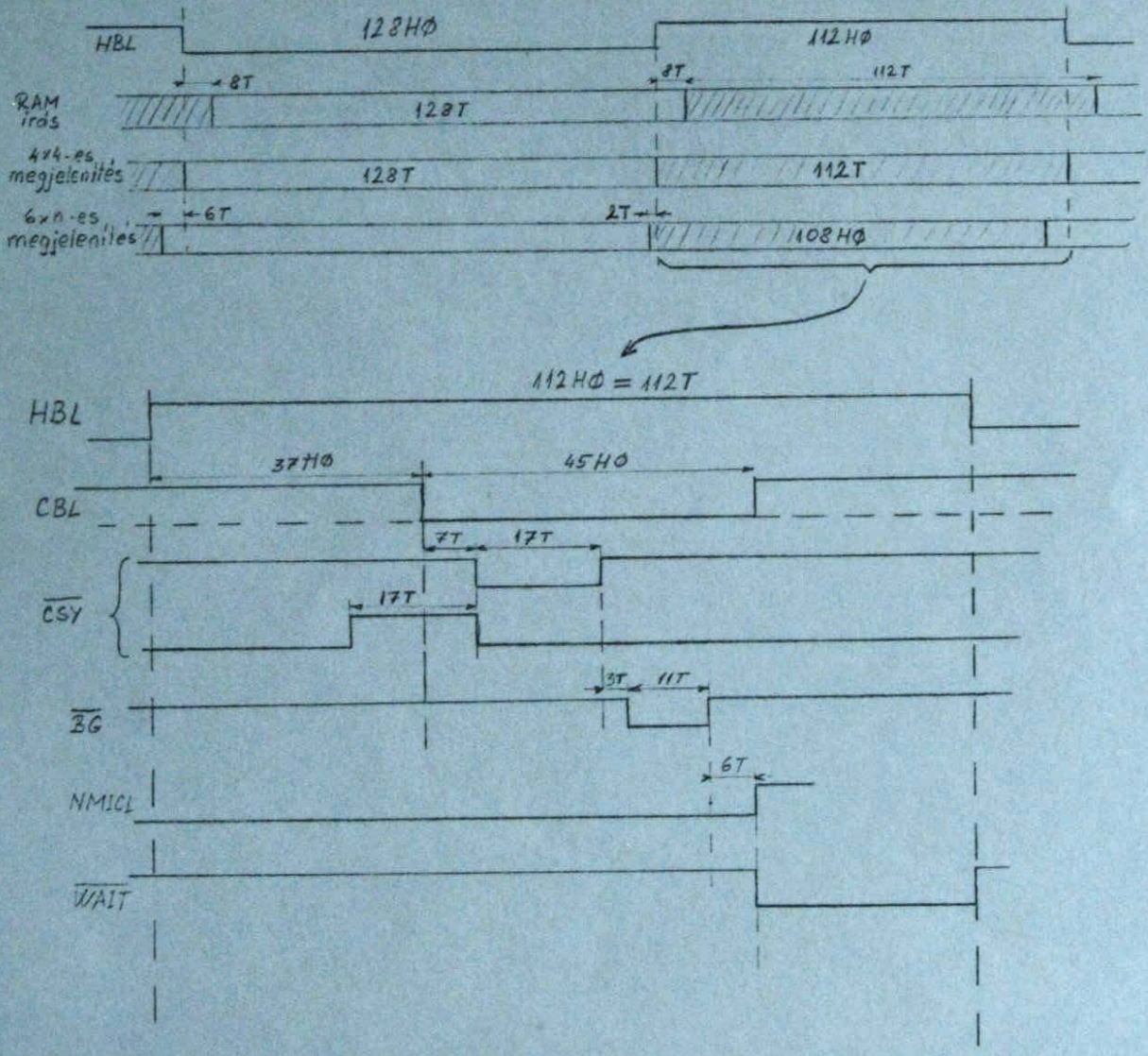
Az MPX=0 (LUT írás) biztosítja a BD megszünését (ilyenkor a BDR mindenképpen magas kell legyen). Az aktív sorokon belül a margót a HBL-ből a (Primo sorkioltó jele) származtatjuk, csak a már ott is meglévő 1 byte-os (8 raszteres) csúszást kell kompenzálni. Ezért itt is előállítjuk a PRIMO-ban lévő H3=H3* 2^3 helyértékű oszlopszámláló jelét, ez biztosítja az említett elcsúszás kompenzálását.



12 ábra

5. Néhány szó a működtető programról

A program struktúráját TV sorokra bontva fogjuk vázolni. A különböző esetek időbeli helyzeteit a következő ábra szemlélteti:



13 ábra

A program főbb lépései:

(NMI) → WAIT HBL↓		sorszámlálók, kezdeti értékek, margó beállítása
	31 sor	szinkron stb. generálás
	3 sor	képkioltás
	3 sor	képszinkron LUT számlálók kezelése, LUT kijelölés
	3 sor	képkioltás LUT írás (8+16+8 byte) (BDR=1,MPX=0)
	15 sor	képkioltás
	1 sor	képkioltás, üzemmód figyelés, számlálók állítása
	(256-N) sor	RAM írás, margó
	N sor	megjelenítés. (ezt a folyamatot majd az NMI szakítja meg)