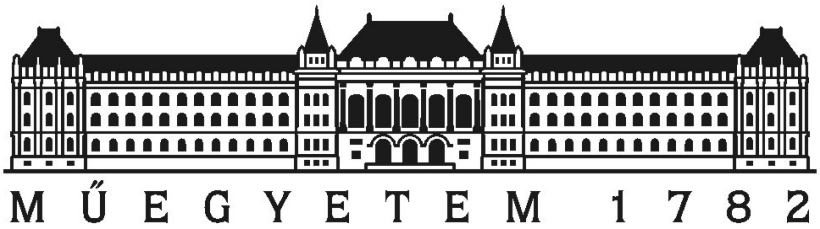


Budapesti Műszaki és Gazdaságtudományi Egyetem

Mikrorendszerek fejlesztése FPGA áramkörökkel
Házi Feladat

Készítette:
Szikra István
URLJRN

1 Feladat

VGA kijelzésű, PS/2-es PC billentyűzet bevitelű számológép elkészítése.

2 Megoldás

A kijelzésre választott megoldás a régi időkből ismert és jól bevált karakteres megjelenítés. (DOS, boot képernyő) Ez több előnnyel is jár. A grafikus megjelenítéssel szemben kisebb memóriára van szükség, mégsem mondunk le teljesen a pixeles felbontásról. A szöveg, számok kiírása is sokkal egyszerűbb, mintha a karaktereket programból kellene generálni.

Egy 8x8 makro pixel-es felbontás csökkentéssel szemben ezzel szebb képet lehet megvalósítani ASCII grafika segítségével. Az ASCII art [1]egy teljes művészeti irányvonalat tudhat magáénak. Igazi jelentősége régebben volt amíg a grafikus kezelő felület nem volt ennyire elterjedt és az emberek DOS Navigator-ban, Norton Commander-ben szerkesztették olvasták a readme-ket, de még ma is a sok nfo fileban találhatunk valamilyen díszítő elemet. Ma már találkozhatunk ASCII megjelenítéssel három dimenzióban kockát forgató programmal, vagy képet konvertáló, de még karakteres videó lejátszó programmal is. A VGA BIOS 8x16-os jól ismert karaktereit választottam fontkészletnek. Ugyanazt a hatást mindenképpen el lehet érni, mint a 8x8-as makro pixellel a #219 es teli, #220-as félig teli, #223-as félig üres és #255/#0/#32 üres karakterek alkalmazásával. Az alkalmazott VGA illesztő áramkör az analóg szín jeleket egy egyszerű ellenállás létrával mint 2 bites DA konverterrel állítja elő. Így a színfelbontás 6 bites. A szabvány 16 színű VGA üzemmód megvalósításához kell egy 4-ből 6-ra dekódoló. A PC-n a szöveges képernyő memórián a karakterek színekkel együtt voltak tárolva a 0xb800[2] címen: alsó byte a karakter kódja, felső byte a szín, melyen belül az alsó nibble a karakter szín, a felső nibble a háttér szín (intenzitás bittel, vagy villogás bittel).

Mivel az FPGA-ba lévő blokkram 18 kbit-es, így szerettem volna kihasználni a plusz paritás biteket, és nem 16 biten hanem 18 biten tárolni a karakter kódot és szín információkat. Több lehetséges megoldás is felmerült:

- csökkentett karakterkészlet: 64 darab –> 6 bit karakter kód, 6 bit szín+6 bit háttér szín
- 8 bit karakter kód, 5+5 bit szín (5-ből 6-ra dekóderrel)
- 8 bit karakter kód, 4+4 bit szín (4-ből 6-ra dekóderrel)+2 villogás üzemmód bit (előtér-háttér váltás, komplementer színek)
- 8 bit karakter kód, 6 bit karakter szín, 3 bit háttér szín, villogás
- 8 bit karakter kód, 6 bit karakter szín, 4 bit háttér szín (3 bit RGB+intenzitás)

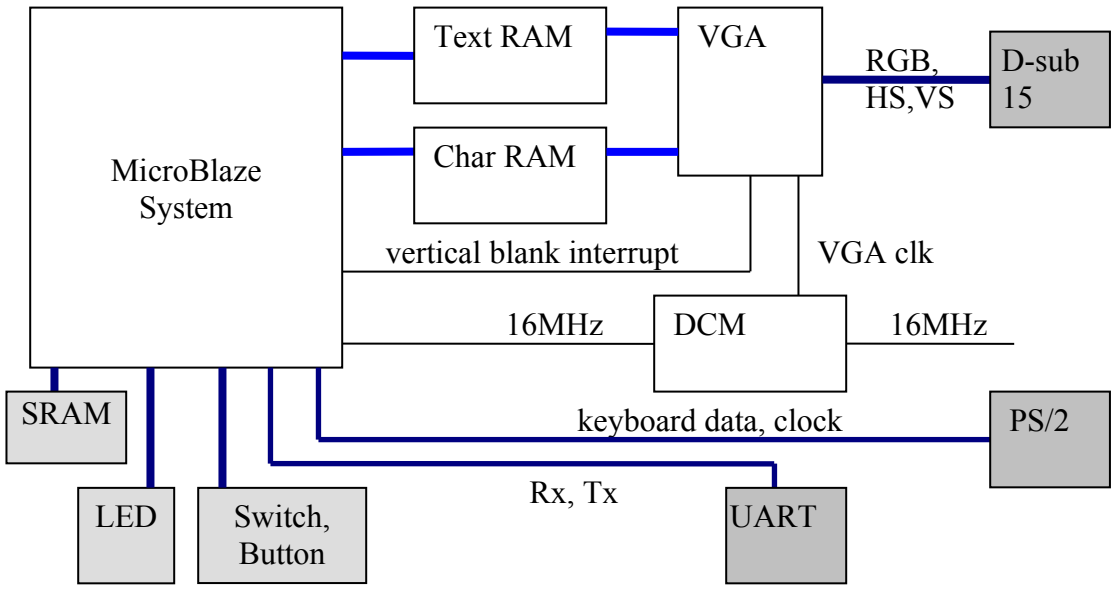
Én ez utóbbit választottam.

Ahogyan a PC-n is, itt is mind a szöveg mind a karakter tábla memóriát elérhetővé kívántam tenni a mikrovezérlő számára. Így még flexibilisebbé lehet tenni a megjelenítést a karakter bitmapek felüldefiniálásával. A legnagyobb korlátozás a grafikus üzemmódhoz képest, hogy egy 8x16-os területen csak két szín használható.

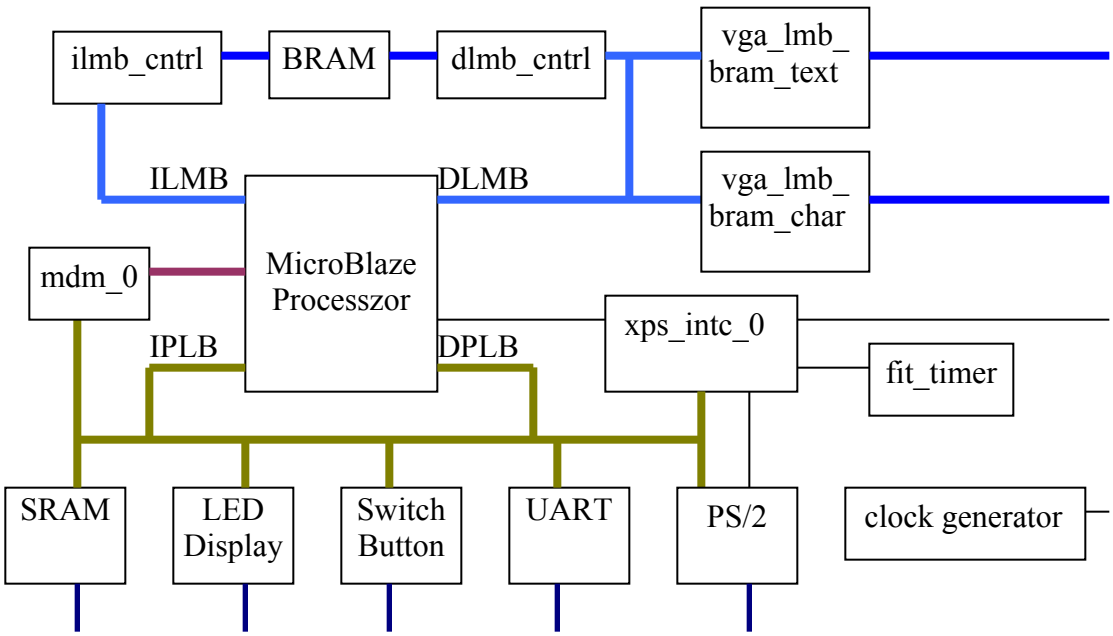
[1] http://www.google.co.uk/search?hl=en&q=ascii+art&meta= (~2 millió találat)
[2] http://bos.asmhackers.net/docs/vga/vgadoc4b/VGABIOS.TXT

3 Blokkvázlat

A teljes rendszer vázlata:



A XPS-ban megvalósított MicroBlazes rendszer:



4 VGA (Video Graphics Array)

A VGA[3] meghajtó tervezéséhez az interneten kerestem leírásokat. Több helyen is találtam információt[4], bár az összes felbontást[5] tartalmazó specifikációt / szabványt nem találtam (hamarabb végeztem a feladattal).

[3] http://en.wikipedia.org/wiki/Video_Graphics_Array

A karakteres üzemmód paraméterei:

Pixelfelbontás: 640 * 400 px (vízszintes * függőleges)
Karakteres felbontás: 80 * 25 char
Font méret : 8 * 16 px
Képfrissítési frekvencia: 70 Hz
HSync polaritás: negatív
VSync polaritás: pozitív
Pixel órajel: 25,395 MHz

4.1 A megvalósításról

A VGA Vezérlő főbb részei:

- Pixel órajel előállítása (DCM)
- Szinkron jelek előállítása (HSync, Vsync)
- Karakter generátor memória
- Szöveg és szín memória
- Memóriák címzése
- HW kurzor
- Pixel, Szín előállítás

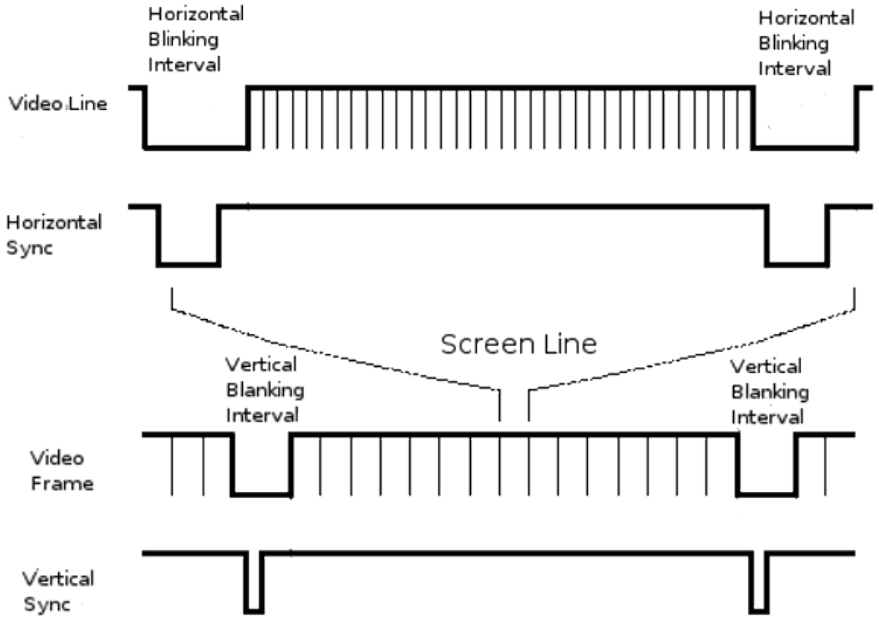
4.2 Pixel órajel előállítása (DCM)

A Spartan 3E-ben található DCM_SP device primitívet alkalmaztam. A kívánt képfrissítési frekvenciához kiszámított pixel órajel frekvenciát lehető legjobban megközelítve. Az FPGA panelen egy 16MHz-es oszcillátor van, ezzel kellett előállítani 25,395 MHz-es órajelet. Ezt a 27/17-es arányú szorzással közelítettem meg: 16MHz*27/17 = 25,411 MHz
(640x480 @60Hz –> 25,175MHz)

4.3 VGA szinkron jelek

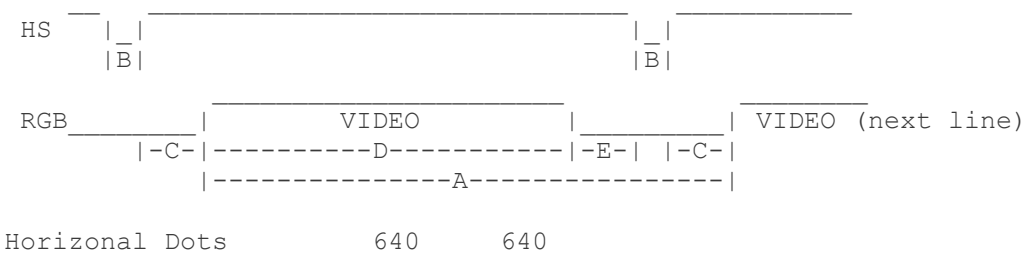
Az aktív videó tartomány és a szinkron jelek között szünet van (ball, jobb, felső, alsó keret).

[4] http://tinyvga.com/vga-timing/640x400@70Hz
http://www.epanorama.net/documents/pc/vga_timing.html
http://www.javiervalcarce.eu/wiki/VGA_Video_Signal_Format_and_Timing_Specifications
http://www.xess.com/appnotes/vga.pdf
[5] http://en.wikipedia.org/wiki/File:Vector_Video_Standards2.svg

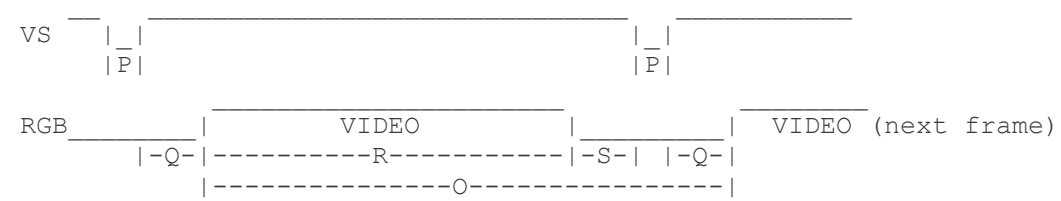


Az időzítések stílusosan ASCII grafikával:

```
Horizonal Dots          640     640
Vertical Scan Lines     400     480
Horiz. Sync Polarity    NEG     NEG
A (us)                  31.77   31.77     Scanline time
B (us)                  3.77    3.77      Sync pulse lenght
C (us)                  1.89    1.89      Back porch
D (us)                  25.17   25.17     Active video time
E (us)                  0.94    0.94      Front porch
```




```
Horizonal Dots          640     640
Vertical Scan Lines     400     480
Vert. Sync Polarity     POS     NEG
Vertical Frequency      70Hz    60Hz
O (ms)                  14.27   16.68     Total frame time
P (ms)                  0.06    0.06      Sync length
Q (ms)                  1.08    1.02      Back porch
R (ms)                  12.72   15.25     Active video time
S (ms)                  0.41    0.35      Front porch
```



Az időzítések paramétereit ezektől eltérő felbontásokra a tinyvga.com-ról töltöttem le. Írtam egy perl scriptet, amivel a HTML filokat egy CSV fileba konvertáltam. A CSV file-t pedig excel táblázatkezelőbe importáltam. A LogSys kártyán található 16MHz-es oszcillátorból az igényelt video pixel órajeléhez legközelebbi előállítható órajel kiszámítására VBScriptet írtam (excel makró).

A szinkron jelek előállításához egy horizontális és egy vertikális számlálót használok (Hcnt, Vcnt). Az előbbi a pixelórajel minden periódusánál lép, komparátorok generálják a horizontális kioltást, szinkron impulzust és a számláló nullázását. A függőleges számláló pedig soronként lép csak (a Hcnt nullázásakor). Szintén komparátorok állítják elő a vertikális kioltást (ezzel együtt egy megszakítást), a szinkron impulzust és a számláló nullázását.

A karakter kép kirakásához felhasználom a horizontális és vertikális számlálókat. A horizontális számláló alsó 3 bitje a pixel előállító pipeline egyes fázisait szabályozza, és vezérli a karakter bitmap 8 vízszintes pixelét előállító shift regisztert. Ez utóbbi felső bitje alapján dől el, hogy az előtér, vagy a háttér színt kell-e kirakni. A vertikális számláló alsó 4 bitje pedig a karaktermemória címzésénél jön jól, mivel ezzel választja ki a karakter 16 sorából a megfelelőt.

4.4 Karakter generátor memória

A fontkészlet feltöltése nyilvánvalóan kritikus tervezési kérdés. Mivel a minden képfrissítésnél szükség van erre a memóriára, így célszerű belső memóriával megvalósítani. Mivel a font bitmapjeit belső blokkramban tárolom így lehetőség van az FPGA konfiguráció betöltéssel együtt ezt is inicializálni. Így nem kell külön flashből betölteni, vagy egyéb módszerrel generálni.

A 8x16 méretű font 256 ASCII karakterrel 4096 byte-os memóriában fér el (32768 bit). Ehhez kettő BRAM-ra van szükség. Célszerű lenne az adatszélességet megosztani a két memória között, hisz így mindkét memória azonos címet és vezérlő jeleket kapna, és az adat be-, kimeneteket is csak össze kéne fűzni. Viszont ez a módszer itt 2 problémát is okoz. Minden címzésnél egy karakter sorát kell felhozni a memóriából. Jelen esetben a fontméret 8x16-os, ezért egyszerre csak 8 pixelre van szükség. Igaz hogy nem használjuk a 9.bitet (bar használhatnánk 9x16-os fontot is), de lehet, hogy a mikroprocesszor oldaláról szeretnénk kihasználni ezeket a plusz biteket. Viszont a 9 bites adatszélességet nem lehet felezni. A másik nehézség a memória inicializálása. A kiválasztott fontkészlet bináris bit térképeit valahogy használható formára kell konvertálni. Célszerű verilog inicializációt választani. Viszont osztott adatszélesség esetén minden byte alsó és felső nibbleje két külön blokkramba kellene kerülnie. Igaz, hogy ez utóbbi nem jelent túl sok gondot mivel ezt a konverziót a data2mem megfelelő bmm file bemenettel el tudja végezni. A másik megoldás a cím szerinti megosztás. Mivel csak két 8 bit adatszélességű ram kimenetét kell multiplexelni ennek az erőforrás igénye nem nagy. Mivel egyszerre csak az egyik ram van engedélyezve, így elméletileg a fogyasztás alacsonyabb.

Sokféle font elérhető interneten. Én a VGA-ROM.F16–ot használom a fntcol16.zip[6] gyűjteményből. Ezeket nem védi szerzői jog.

[6] ftp://ftp.simtel.net/pub/simtelnet/msdos/screen/fntcol16.zip

A fontok konvertálására két programot használtam. Írtam egyet Delphiben (bit2mem.exe) ami egy bináris állományt konvertál VMem formátumra, amit a data2mem értelmezni tud. E helyett használható az srec_cat tool is. A másik program a Xilinx dat2mem eszköze:

```
data2mem -bm vga_txt_p.bmm -bd %1.mem -o v %1.v
```

A bmm filet a következő utasítással állítottam elő:[7]

```
data2mem -mf p chartable PICOBLAZE 0 a charbitmaps b 0x0000 8 s RAMB16
0x0800 8 vgarom_low s RAMB16 0x0800 8 vgarom_high -o p vgarom.bmm
```

4.5 Szöveg és szín memória

Teljes memória igény: 2000*18 bit (80*25*18=36000). Ehhez két darab BRAM szükséges. A core generátor dokumentációjában Minimum Area módszernek hívott megoldás alkalmazását választottam az egyszerűség és alacsony erőforrás igényen túl azért is, mert a 2-szer 9 bit megosztás lehetőséget ad a teljes memória tartomány kihasználására. Ezen túl egyszerűbb verilogban inicializálni a memóriát, mert elkülönül a karakter kód tartomány és a szín tartomány.

A karakter szó bitkiosztása a következő:

Bit	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	BI	BR	BG	BB	FR1	FR0	FG1	FG0	FB1	FB0	C7	C6	C5	C4	C3	C2	C1	C0

A BI,BR,BG,BB alkotja a háttérszínt: intenzitás, RGB szín
Az FR1, FR0, FG1, FG0, FB1, FB0 a karakter színét adja meg 6 bites RRGGBB formában.
A C7..C0 a 8 bites ASCII karakter kód. (C0 az LSB)

A kikeverhető színek jelentős részét definiáltam a vga.h-ban, itt van néhány:

```
#define VGAF_WHITE      0x3f
#define VGAF_BBLUE      0x03
#define VGAF_BLUE       0x02
#define VGAF_DBLUE      0x01
#define VGAF_GREEN      0x08
#define VGAF_RED        0x20
#define VGAF_YELLOW   (VGAF_GREEN |VGAF_RED  )
```

…

4.6 Memória címzése

4.6.1 Címzés frame előállítás közben

A képernyő pásztázása során a pixel előállításához szükség van az előtér (karakter) és a háttér színére valamint a karakter bitmap (font) megfelelő sorára. Ehhez előtte a szöveg memóriából be kell tölteni a képernyőn megjelenítendő karakter kódját, ugyanis ebből állítható elő a karakter bitmap sor címe a karakter rom-ban. Először viszont az éppen kijelzendő karakter indexét kell legenerálni a szöveg memória címzéséhez. Tehát az alapvető lépések valahogy így néznek ki:

- Frame kezdet (vég) előállítás: aktív terület kezdetének detektálása, adatok előállításának kezdete, init.
- Text ram címének előállítása (txtAddr = 80*row + column, column= Hcnt / 8 = Hcnt >> 3, row = Vcnt / 16 = Vcnt >> 4). Viszont ezt nem szorzással hozzuk létre, hanem

[7] http://home.mnet-online.de/al/BRAM_Bitstreams.html

báziscím használatával (txtAddr=txtRowAddr+column) ahol a báziscímet szöveg soronként növeljük 80-nal: 0. sornál (vagy előtte) nullázzuk, 16 pixel soronként növelés 80-nal (összeadó). A növelés történhet pl. HSync-nél (Hend-nél), az init pedig VSync-nél (Vend).

- A beolvasott karakter kód alapján a karakter memória címzése, és olvasása.
- A beolvasott karakter bitmap sor betöltése shift regiszterbe, a kishiftelt bitnek (MSB-vel kezdve) megfelelő szín kiválasztása, és kirakása a képernyőre.

4.6.2 Memória címzés microblaze-ből

A 32 bites adatszélességnek az alsó bitjei tartalmaznak adatokat (18 bit szöveg memóriánál, 9 bit karakter memóriánál). Az egymást követő karakterek címe 4-essével növekszik (x,y:0,0 pozíció címe = báziscím, x,y:1,0 pozíció címe = báziscím+4, x,y:0,1 pozíció címe = báziscím+4*80) Erre a szoftverben makrókat irtam:

```
#define VGATEXT    ((volatile tu32*)XPAR_VGA_LMB_BRAM_TEXT_BASEADDR)
#define VGACHAR    ((volatile tu32*)XPAR_VGA_LMB_BRAM_CHAR_BASEADDR)
```

A hardverben ez úgy van megoldva, hogy az LMB-BRAM kontrollerből érkező memória cím alsó két bitje nincs bekötve, hanem a harmadik címvezeték van a memória cím LSB-jére kötve.

Megjegyezném még, hogy az új Xilinx EDK a régebbi verzióktól eltérően nem engedi az LMB-BRAM kontroller jeleit külső jelként kivezetni. A régi XPS verziók csak figyelmeztettek, hogy nem BRAM van bekötve, viszont a 11.3-as verzió már hibával leáll. Ehhez a kontroller tcl scriptjét módisítanom kellett. Másik megoldás egy BRAM wrapper[8] használata lett volna.

4.7 HW kurzor:

A VGA vezérlőben található még egy villogó hardveres kurzor is. A kurzor alakja (blokk, aláhúzás), pozíciója (x,y) és láthatósága nem külön vezetékeken állítható, hanem a text memória egy kijelölt címén van eltárolva. Ez nagyban megkönnyíti a VGA vezérlő illesztését a mikroprocesszorhoz, mivel nem kell új periféria illesztőt készíteni, illetve jeleket huzalozni. A kurzor használatához minden adott, programból csak a szöveg memória egy kitüntetett a képernyőn nem látható tartományát kell írni, olvasni. Ez a memória utolsó adata (2048 karakteres memória esetén a 2047-es indexű terület). Mivel a szöveges képernyő csak 80*25=2000 karakteres, így a fennmaradó memória bármire felhasználható. (pl. UART buffernek…)

A kurzor karakter szó bitkiosztása a következő:

Bit	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	X	mode	invisible	Y6	Y5	Y4	Y3	Y2	Y1	Y0	X7	X6	X5	X4	X3	X2	X1	X0

A mode a kurzor megjelenítési módja: 0=aláhúzás, 1=blokk.

Az invizible tulajdonképpen az Y7, amivel a kurzor pozícióját a látható képernyőn kívülre helyezzük, ezzel téve láthatatlanná ill. ezzel kapcsoljuk ki a kurzort.

Az Y6..Y0 a kurzor sor (row) pozícióját adja meg.

Az X7..X0 a kurzor oszlop pozícióját adja meg.

A kurzor kezelése szoftverből:

```
#define VGACURSORIDX 0x7ff
/// set cursor mode
```

[8] http://forums.xilinx.com/xlnx/board/message?board.id=EMBEDDED&thread.id=2159

```
void vgaSetCurMode(tu32 mode)
{
  VGATEXT[VGACURSORIDX] = (VGATEXT[VGACURSORIDX] & 0x0ffff) | (mode<<16);
}
void vgaSetCurPos(tu16 pos)
{
  VGATEXT[VGACURSORIDX] = (VGATEXT[VGACURSORIDX] & 0xf0000) | pos;
}
tu16 vgaGetCurPos()
{
  return VGATEXT[VGACURSORIDX];// & 0x0ffff;
}
```

4.8 Pixel előállítás

A karakter memóriából a megfelelő karakter bitmap sort egy shift regiszterbe olvassuk. Beolvasás után minden órajelnél eggyel léptetjük az MSB felé. Mindig az MSB alapján döntjük el, hogy előtér (karakter) színt, vagy háttér színt kell-e kirakni. (Ezeket a színeket már a karakter kódjával együtt beolvastunk a szöveg memóriából.) Amikor a kiválasztott szín megvan, már csak a kurzort kell figyelembe venni a kimeneti RGB jelek előállításánál. Ha éppen a látható kurzort rajzoljuk, akkor invertálni kell a színt (komplementert képezni).

5 PS/2 PC billentyűzet

A Xilinx EDK-ban van az IP core könyvtárban PS2 vezérlő, ami kétirányú kommunikációra is képes. A kommunikáció legjelentősebb része a billentyűzetről a gazda rendszerbe érkező adatok, melyek a billentyűk lenyomásakor generált scan-kódokat tartalmazzák. A vissza irányú kommunikációra leginkább azért van szükség, hogy a billentyűzeten található LEDeket tudjuk ki-be kapcsolni. Ugyanis a billentyűzet nem kezeli sem a CapsLock, NumLock és egyéb módosító billentyűket. A PC-n mindez a BIOS illetve az operációs rendszer feladata. Mint a különböző területi billentyűzet kiosztások (ékezetes karakterek, z-y pozíció, stb.) kezelése. A megszokott kis nagy betű váltás amit a CapsLock, vagy a Shift lenyomásával érhetünk el, a szám billentyűk és -=`[];',./\ billentyűk alternatív jelei közötti váltás amit (csak) a Shift nyomva tartásával érhetünk el, az AltGr módosító billentyű nyomva tartásával elérhető újabb karakterkészletre váltás megvalósítása mind a mi feladatunk. Ezen kívül ott van még a tetszőleges ASCII karakter begépelésének lehetősége az Alt és numerikus padon található számok által. Ezekből a házi feladat megoldása során nem volt célom mindent megvalósítani, csak a számológép funkcionális használhatóságához szükséges egy részhalmazát.

Az említett kommunikációs eseményeken kívül van még más is[9]: echo, billentyűzetismétlési gyakoriság beállítás, adat újraküldés…

Az billentyűzetkezelés elengedhetetlen része megoldható akár szoftveresen is, (GPIO interfészen keresztül) (ez a megoldás bármely mikrokontrolleren használható).

Az AT billentyűzet kezelését legjobban összefoglaló leírást a Beyonnd Logic[10] oldalán találtam. Az egyes gombokhoz rendelt scan-kódokat is innen másoltam, és

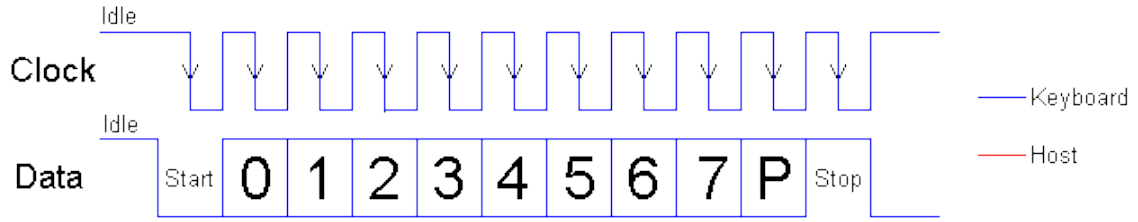
[9] http://www.computer-engineering.org/ps2keyboard/
[10] http://www.beyondlogic.org/keyboard/keybrd.htm

készítettem belőlük excel táblázatot. Ebből utána egy scan2ascii konvertáló LUT-t készítettem a szoftveres dekódoláshoz.

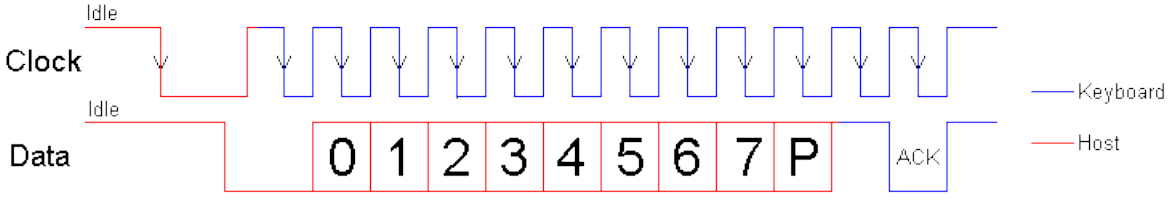
```
const char scan2ascii[128]={
///   0     1     2     3     4     5     6     7
    0  ,  0  ,  0  ,  0  ,   0 ,   0 ,   0 ,  0  , /// 00
    0  ,  0  ,  0  ,  0  ,   0 ,   0 ,  '`',  0  , /// 08
    0  ,  0  ,  0  ,  0  ,   0 ,  'q',  '1',  0  , /// 10
    0  ,  0  ,  'z',  's',   'a',  'w',  '2',  0  , /// 18
    0  ,  'c',  'x',  'd',   'e',  '4',  '3',  0  , /// 20
    0  ,  ' ',  'v',  'f',   't',  'r',  '5',  0  , /// 28
    0  ,  'n',  'b',  'h',   'g',  'y',  '6',  0  , /// 30
    0  ,  0  ,  'm',  'j',   'u',  '7',  '8',  0  , /// 38
    0  ,  ',',  'k',  'i',   'o',  '0',  '9',  0  , /// 40
    0  ,  '.',  '/',  'l',   ';',  'p',  '-',  0  , /// 48
    0  ,  0  , '\'',  0  ,   '[',  '=',  0  ,  0  , /// 50
    0  ,  0  ,  0  ,  ']',    0 , '\\',  0  ,  0  , /// 58
    0  ,  0  ,  0  ,  0  ,   0 ,   0 ,  0  ,  0  , /// 60
    0  ,  '1',  0  ,  '4',   '7',   0 ,  0  ,  0  , /// 68
   '0',  '.',  '2',  '5',   '6',  '8',  0  ,  0  , /// 70
    0  ,  '+',  '3',  '-',   '*',  '9',  0  ,  0    /// 78
};
```

Lehetne a táblázat méretén csökkenteni a dekódoló algoritmus bonyolításával: pl. az első sor és a hetedik oszlop törlésével `scan2ascii`[(sc/8-1)*7+sc%8]. Több ilyen táblázatot is fel lehet venni a különböző módosító billentyűkre (Shift, AltGr).

A kommunikáció szinkron soros, ahol az órajelet a billentyűzet generálja. A billentyűzetről érkező adatokat az órajel lefutó élénél kell mintavételezni. Az adatbitek sorrendje: start bit (0), 8 darab adatbit LSB-vel kezdve (Least Significant Bit), páratlan paritás bit, stop bit (1).



Mivel az órajelet mindig a billentyűzet generálja, így a billentyűzetnek küldött adatoknál, a gazdának kérnie kell az adatküldést az órajel alacsonyra húzásával (open collector-os meghajtás) (legalább 60 µs időre). A billentyűzet 10 ms-on belül elkezdi az adatok fogadását az órajel generálásával. Az adatok fogadása után a billentyűzet generál egy elfogadó jelzést (Ack).



A küldött scan kódok nem mind egy byte-osak, annak ellenére, hogy nincs 256 gomb a billentyűzeten. Vannak két byte-os kiterjesztett kódok, ezek 0xE0-val kezdődnek,

sőt vannak ennél is hosszabbak: Print Screen: E0,12,E0,7C, vagy a Pause: E1,14,77,E1,F0,14,F0,77. Valamint minden billentyű felengedést a scan-kódba egy F0 beszúrása jelöl. Itt van a scan kódokat 16 bites értékké alakító függvény:

```
tu16 ProcessKeybScanCode(void)
{
  tu8 ki = keybIdx;
  if (ki==0) return 0; //nothing to do

  tu16 ret = 0;

  switch (keybData[0]){
    /// kivetelek
    case 0xE1: /// Pause : E1,14,77,E1,F0,14,F0,77
          if (ki>=8) {
            ret = 0x0100;
            ki = 8; /// processed chars
          }
    break;
    case 0xE0: /// E0xx v. E0F0xx extended codes
      if (ki>=2) {
        if (keybData[1] == 0xf0){ /// E0F0xx
          if (ki>=3) {
            ret = keybData[2] | 0xfe00;
            ki = 3; /// processed chars
          }
///       } else if (keybData[1] == 0x12) { /// E012E07C PrintScreen // will be split
        } else { /// E0xx
           ret = keybData[1] | 0x0e00;
           ki = 2; /// processed chars
        }
      }
    break;
    case 0xF0: /// F0xx   /// felengedes
      if (ki>=2) {
        ret = keybData[1] | 0xf000;
        ki = 2; /// processed chars
      }
    break;
    default:
        ret = keybData[0];
        ki = 1; /// processed chars
  }
  if ( (ret) ) { /// volt feldolgozott adat
    microblaze_disable_interrupts();
      tu8 i;
      for (i = ki; ki<keybIdx; i++){
         keybData[i-ki] = keybData[i];
      }
    keybIdx-=ki; /// critical section
    microblaze_enable_interrupts();
  }
  return ret;
}
```

Itt pedig az értelmező, parancsvégrehajtó:

```
void ProcessKeyb(tu16 scancode)
{
  p.pos = vgaGetCurPos();
  switch(scancode){
    case 0xfe70: /// insert
      ///insert = !insert; /// toggle: insert / overwrite
     overwrite = !overwrite;  vgaSetCurMode(overwrite); /// block cursor
    break;
/*** Not Used (Yet)
    case 0x0012: /// shift
    case 0x0059: /// right shift
      shift=1;   /// Capital letters, alternate chars
    break;
    case 0xf012: /// shift fel
```

```
    case 0xf059: /// right shift fel
      shift=0;
    break;
*/
    case 0x0e75: /// up
      if (p.y>0) p.y--;      vgaSetCurPos(p.pos);
    break;
    case 0x0e72: /// down
      if (p.y<19) p.y++;     vgaSetCurPos(p.pos);
    break;
    case 0x0e6b: /// left
      if (p.x>55) p.x--;     vgaSetCurPos(p.pos);
    break;
    case 0x0e74: /// right
      if (p.x<79) p.x++;     vgaSetCurPos(p.pos);
    break;
    case 0x0e7d: /// pgup
      p.y=0;                 vgaSetCurPos(p.pos);
    break;
    case 0x0e7a: /// pgdown
      p.y=19;                vgaSetCurPos(p.pos);
    break;
    case 0x0e6c: /// home
      p.x=55;                vgaSetCurPos(p.pos);
    break;
    case 0x0e69: /// end
      p.x=79;                vgaSetCurPos(p.pos);
    break;
    case 0x0066: /// backspace
      if (p.x>55) {
        p.x--;  DelChar();  vgaSetCurPos(p.pos);
      }
    break;
    case 0x0e71: /// del
      DelChar();
    break;
    case 0x005a: /// enter
    case 0x0e5a: /// enter
      if (p.y<19) {
        p.y++;  p.x=55;      vgaSetCurPos(p.pos);
      }
    break;
    case 0x0076: /// esc
      ClearInput();
    break;

    case 0x0e4a: /// / (numpad)
      InsertChar('/');
    break;
    default:
      if ((scancode & 0xff80)==0x0000) {// lenyomott std keys
        char ac = scan2ascii[scancode & 0x7f];
        if (ac) { /// lathato karakter
          InsertChar(ac);
        }
      }
  }
}
```

Egyéb protokoll és parancs leírások:
http://www.computer-engineering.org/ps2protocol/
http://www.computer-engineering.org/ps2keyboard/scancodes2.html
Néhány neten találhatók vhdl PS/2 periféria:
http://www.xess.com/projects/ps2_ctrl.pdf
http://www.javiervalcarce.eu/wiki/VHDL_Macro:_PS2

6 Számok kezelése, ábrázolása, konvertálása

Decimális számok kezelésére több módszer is van. Az egyik, amit én is alkalmaztam, az a decimális és bináris számok közötti oda-vissza konverzió, és bináris értékekkel történő számolás. Egy másik lehetőség az lett volna, hogy folyamatosan valamilyen decimális számábrázolással való számolás. Persze erre a mikroprocesszor, vagy a c fordító nem képes. Így egyrészt minden műveletet saját magunk kell megírnunk, de magát a decimális értékek tárolását is nekünk kell kitalálni. Erre az egyik megoldás, hogy mint ASCII string tároljuk, és hajtunk rajta végre műveleteket, egy másik megoldás a széles körben használt BCD[11] kódolás, további lehetőség a DPD[12] kódolás, ami tömörebb a BCD-nél. Még akár decimális lebegőpontos[13] számábrázolást is választhatunk. Azt itt jegyezném meg, hogy míg egész számoknál túlcsordulás mentes esetben az a+b-a=b mindig igaz addig lebegőpontos számoknál a kerekítési hibák miatt erre nem számíthatunk. Legtöbb esetben a kerekítési hibák nem okoznak gondot, de ha nem figyelünk oda rájuk, akkor komoly gondjaink akadhatnak. Probléma az is, hogy amit például tízes számrendszerben le tudunk írni véges tizedes jegyekkel, azt nem feltétlenül lehet kettes számrendszerben leírni véges számú számjegyekkel. Egy egyszerű példa erre a 0.4-3*0.1 művelet, melynek eredményére kaphatunk 0.099999999999-et. Felhívnám arra is a figyelmet, hogy a bináris decimális átalakítás lebegőpontos számoknál közel sem triviális. Ki tudná fejből megmondani, hogy 2^{47} hányas számjeggyel kezdődik tízes számrendszerben, vagy 10^{23}-nak mi az első négy számjegye bináris alakban. Ezekre az átalakításokra táblázatokat is igénybe vesznek az inttostr() és barátaik. Vannak algoritmusok[14], melyek segítségünkre lehetnek a kerekítési hibák mérséklésére pl. az összeadás művelet során.

6.1 BCD és bináris konverzió alapja

Triviális, hogy az eltérő számábrázolással leírt értékeknek meg kell egyezniük (egész számok esetén).

$v = d_N*10^N+.. +d_2*10^2+ d_1*10^1+ d_0$

$= b_M*2^M+.. +b_2*2^2+ b_1*2^1+ b_0 = h_O*16^O+.. +h_2*16^2+ h_1*16^1+ h_0$

A $d_N..d_0$ a decimális számjegyek (digitek). d_iЄ [0..9] iЄ[0..N] (d_N!=0)

A $b_M..b_0$ a bináris számjegyek (digitek). b_iЄ [0,1] iЄ[0..M] (b_M!=0)

A $h_O..h_0$ a hexadecimális számjegyek (digitek). h_iЄ [0..9,a..f] iЄ[0..O] (b_O!=0)

A fenti képlet felírható más alakban is

$v = ((…(d_N*10+d_{N-1})*10+… +d_2)*10+ d_1)*10+ d_0$

$= ((…(b_M*2 +b_{M-1})*2+… +b_2)*2+ b_1)*2+ b_0$

Ha a fenti képletet iteratívan valósítjuk meg a bináris leírásmód alapján, akkor egy számot csak kettővel kell tudni szorozni, és egyet, vagy nullát hozzáadni. Ezt alkalmazzuk a decimális számjegyek előállítására is. Az első lépések:

$v[0] = b_M =: d_0[0]$ //1

$v[1] = v[0]*2 + b_{M-1} =: d_0[1] = d_0[0]*2+ b_{M-1}$ //3

$v[2] = v[1]*2 + b_{M-2} =: d_0[2] = d_0[1]*2+ b_{M-2}$ //7

[11] http://en.wikipedia.org/wiki/Binary-coded_decimal
[12] http://en.wikipedia.org/wiki/Densely_Packed_Decimal
[13] http://en.wikipedia.org/wiki/Decimal_floating_point
[14] http://en.wikipedia.org/wiki/Kahan_summation_algorithm

A következő lépésben már történhet átvitel a decimális digiten.
$v[3] = v[2]*2 + b_{M-3} =: d_1[3]*10+d_0[3]$
A $d_0[3] =(d_0[2]*2+b_{M-3})\%10$, az átvitel $c_{0,1}[3]=(d_0[2]*2+b_{M-3})/10$, $d_1[3]= 0*2+c_{0,1}[3]$.
$v[4] = v[3]*2 + b_{M-4} =: d_1[4]*10+d_0[4]$
A $d_0[4] =(d_0[3]*2+b_{M-4})\%10$, az átvitel $c_{0,1}[4]=(d_0[3]*2+b_{M-4})/10$, $d_1[4]= d_1[3]*2+c_{0,1}[4]$.
Általánosan felírva:
A $d_0[i] =(d_0[i-1]*2+b_{M-i})\%10$, az átvitel $c_{0,1}[i]=(d_0[i-1]*2+b_{M-i})/10$,
$d_k[i]=(d_k[i-1]*2+c_{k-1,k}[i])\%10$, $c_{k,k+1}[i]=(d_k[i-1]*2+ c_{k-1,k}[i])/10$,
k=1..N
$(d_k[0]:=0)$, $v=v[M]$, $d_k=d_k[M]$
Mivel a $c_{k,k+1}[i]$ átvitel mindig csak 0 vagy 1 értékű lehet ezért ez nem befolyásolja a következő átvitelt (mivel (2k)/10 = (2k+1)/10). Így $c_{k,k+1}[i]=(d_k[i-1]*2)/10$, vagyis $c_{k,k+1}[i]$ = $(d_k[i-1]>=5)$? 1: 0.
Jól látható, hogy egy decimális szám kettővel való szorzása végrehajtható, a számjegyein végzett elemi műveletekkel.

D[i]	c[i+1]	d[i+1]=d[i]*2%10	BCD	Érték
0	0	0	00	0
1	0	2	02	2
2	0	4	04	4
3	0	6	06	6
4	0	8	08	8
5	1	0	10	16
6	1	2	12	18
7	1	4	14	20
8	1	6	16	22
9	1	8	18	24

BCD kódolásnál az 10 értéke 16, vagyis, ha a számjegy ötös vagy a feletti, akkor az eredményhez hozzá kell adni 6-ot, és az átvitel és az új számjegy is generálódik. Ezzel ekvivalens, ha a szorzás előtt adunk a számjegyhez 3-at (3*2=6).

Tehát egy BCD szám kettővel való szorzása úgy végezhető el, hogy minden 4-nél nagyobb számjegyéhez (nibble) hozzáadunk 3-at, majd mint bináris számot shifteljük ballra eggyel. Így a binárisból BCD-re alakítás tulajdonképpen a $((\ldots(b_M*2 +b_{M-1})*2+\ldots +b_2)*2+ b_1)*2+ b_0$ képletnek a kiszámítása BCD számokon végzett műveletekkel (BCD aritmetika).

6.2 Konvertáló függvények

A bemenet értelmezéséhez néhány karakter osztály definiálása:

```
#define isDecChar(c)    ( (c>='0') && (c<='9'))
#define isHexExtCharL(c)        ( (c>='a') && (c<='f') )
#define isHexExtCharU(c)        ( (c>='A') && (c<='F') )
#define isHexChar(c)    ( isDecChar(c) || isHexExtCharL(c) || isHexExtCharU(c) )
#define isNumChar(c)    ( isHexChar(c) || (c=='$') || (c=='x') || (c=='h') )
```

6.2.1 Binary nibble to Hexadecimal ASCII char

```
char bin2hex(tu8 bin)
{
  bin &= 0x0f;
  if (bin>0x09) {
```

```
    bin+='A'-0x0A;
  } else {
    bin+='0';
  }
  return bin;
}
```

6.2.2 Hexadecimal ASCII char to Binary nibble

```
tu8 hex2bin(char aC)
{
      aC &=0x4f;
      if (aC & 0xf0) aC -= 0x41-0x0a;// H = A-41h +0Ah
      return aC;
}
```

Az előző függvény használható a Decimal ASCII char to BCD nibble átalakításra is.

6.2.3 Hexadecimal ASCII string to binary

```
    for (i=s; i<=e; i++){
      if (isHexChar(line[i])) {
        val = (val<<4) | hex2bin(line[i]);  /// HP: MSB-felol pakolni a
byte tombot
      } else {
        err |= 2; /// nem hexa karakter a szamban
      }
    }
```

Ez a részlet a ParseNumLine függvény része, és egyszerűen 4 bittel shifteli a biáris értéket. Itt csak natív int típusokkal dolgozunk, de az átalakítás átírható High Precision számok kezelésére is, ennen az esetben a bináris érték mint egy byte tömb kerül tárolásra, és az átalakítás során ennek a tömbnek az indexét kell változtatni (csökkenteni vagy növelni, attól függően, hogy little endian, vagy big endian ábrázolást választunk)

```
        val[j] = (hex2bin(line[i])<<4) | hex2bin(line[i+1]);
```

6.2.4 Decimal ASCII string to binary

```
    for (i=s; i<=e; i++){
      if (isDecChar(line[i])) {
        val = val <<1;
        val = val+(val<<2) + line[i]-'0';  /// HP: soros atalakitas,
array shifter
      } else {
        err |= 2; /// nem decimal karakter a szamban
      }
    }
```

Ez a részlet a ParseNumLine függvény része, a bináris értéket tízzel szorozza, és hozzáadja a következő számjegy értékét. A High Precision változat itt csak kicsit bonyolultabb az előzőnél. Egy HP összeadót és shiftelőt kell megírni. A nagy pontosságú egész műveletekhez mint pl. a szorzás ajánlom a LibTomMath[15] könyvtárat.

[15] http://math.libtomcrypt.com/

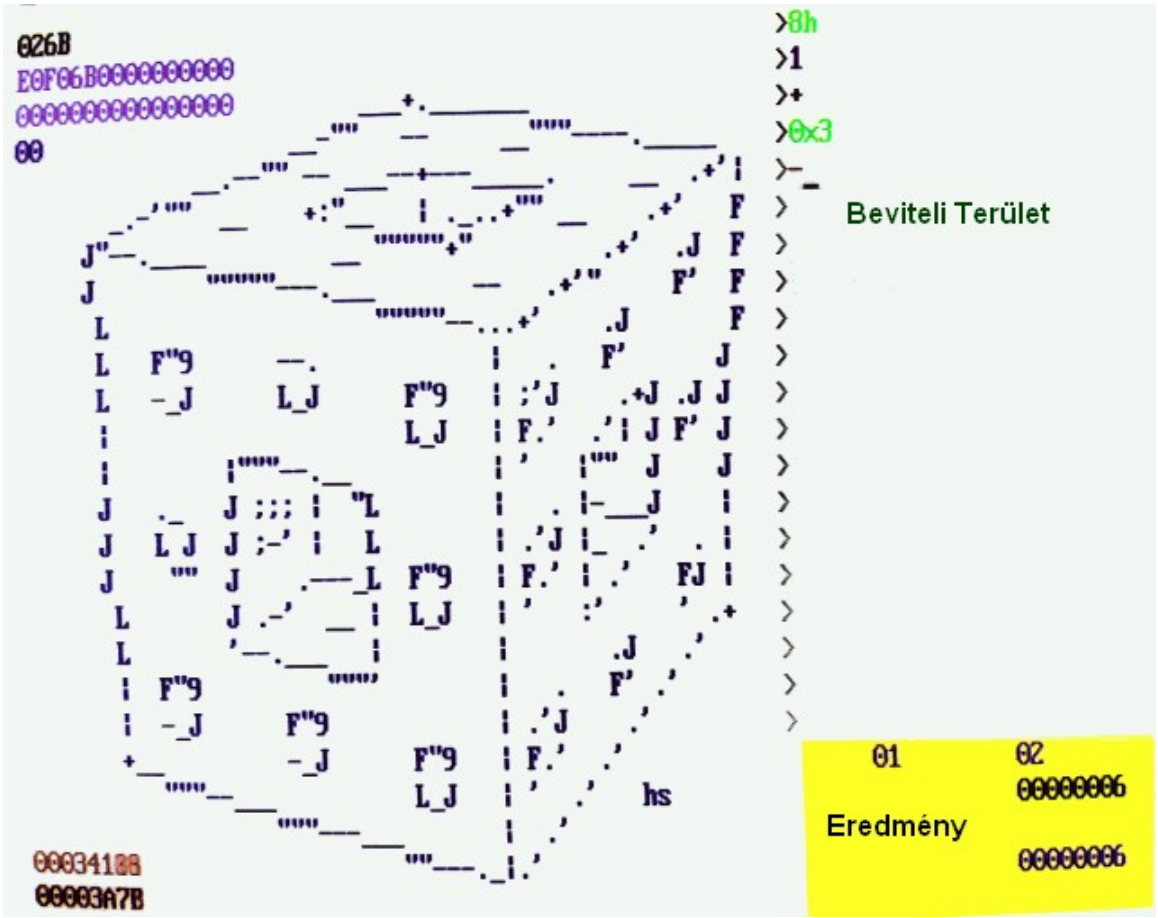
6.2.5 Binary to Hexadecimal ASCII string

```
char WriteHexXY(tu8 x,tu8 y, tu8* value, tu8 digits, tu32 color)
{
  color=color<<8;
  for ( ; digits; ){
    digits--;
    vgaPutXY(x+ digits, y, bin2hex(value[digits/2]) | (color));
    digits--;
    vgaPutXY(x+ digits, y, bin2hex(value[digits/2]>>4) | (color));
  }
}
```

6.2.6 Binary to Decimal ASCII string

Az átalakítás két részből áll, vagyis egyenértékű az alábbi két művelet végrehajtásával: Binary to BCD és Binary to Hexadecimal ASCII string

7 User Manual



A számológép Lengyel suffixes/postfixes[16] (Reverse Polish notation[17]) üzemmódban működik.

[16] http://www.szamologep.eoldal.hu/oldal/rpn_reverse_polish_notation
[17] http://en.wikipedia.org/wiki/Reverse_Polish_notation

7.1 A postfixes műveletvégzés

A számoknak kell elöl szerepelni, utána pedig a rajtuk végzett műveleteknek. A számokat a program egy 8 elemes stack-be tárolja, és a műveletvégzés mindig az utolsó elem(ek)en hajtódik végre. A műveleteket a számok után (alatti sorba) kell írni, és mindig az előző (két) számon, vagy eredményen hajtódnak végre. Ennek a leírásmódnak az előnye, hogy nem igényel zárójelezést.
Néhány példa:
3+4*2
3
4
2
*
+

(3+4)*2
3
4
+
2
*

7.2 Bevitel

20 sorba lehet számokat, vagy műveleteket gépelni (minden sor egy számot, vagy egy műveletet tartalmazhat). Az üres sorok nem hajtódnak végre, tetszőleges sorba lehet adatokat bevinni. A számok tagolásához és a műveletek pozicionálásához az üres karakter (space) szabadon használható, ezeket minden sorból feldolgozás során a program eltávolítja (figyelmen kívül hagyja). Az Escape billentyű lenyomására törlődik a beviteli mező. Az Insert gomb lenyomásával lehet váltani beszúrás és felülírás bevitel között. A kurzor alakja jelöli az éppen aktuális állapotot: aláhúzás =beszúrás, blokk = felülírás. A beviteli területen a navigáció a nyilak, home, end, page up, page down billentyűkkel lehetséges, ill. az enter a következő sor elejére helyezi a kurzort. Javítani (törölni) a del és a backspace gombokkal lehet.

7.2.1 Számok

Jelenleg csak fixpontos 32 bites előjel nélküli egészekkel dolgozik. A program kis munkával átírható előjeles számok kezelésére, illetve 64 bites vagy akár annál nagyobb számok kezelése is megoldható nagyobb programmemóriával.

7.2.1.1 Formátum

Számokat két formátumban lehet bevinni: decimális, hexadecimális (az oktális, és bináris bevitel is megoldható egy kis további programozással ha igény van rá). A két fajta számot eltérő színnel jelöli a program a bevitel során: a decimális zöld, a hexa lila. (A fenti képen a kisebb tintahasználat miatt a képernyő invertáltja szerepel.)

A decimális számok csak 0-tól 9-ig terjedő számjegyeket és üres karaktert (space-t) tartalmazhatnak. Ettől eltérő esetben a szám nem értelmezett, és a program pirosra

színezi a sort jelezve a hibás bevitelt. Tizedes pont/vessző, negatív előjel nincs, illetve az ″e″ sem értelmezett (1000=1e3).

Három fajta hexa szám formátumot ismer fel a program: $12ab, 0x12ab, 12abh
A hibás formátumot sötétpiros színnel jelöli.

7.2.2 Műveletek

A nem értelmezett műveletet szintén pirosas színre színezi a program. Ha nincs elegendő adat a stackben a művelet végrehajtásához, akkor a sor sötét cián színűre vált, míg normális esetben világosra. Egy operandusú műveletnél az eredménnyel lecseréli a stack utolsó elemét (a bemenetet), így a stack mérete nem változik. Két operandusú műveletnél az utolsó két elemet kiveszi a stackből, majd az eredményt visszahelyezi a stackre, ezzel a stack mérete eggyel csökken.

Az alábbi műveletek használhatóak:

not – negálás a stack utolsó elemének minden bitje az ellentettjére vált.
lshift – ballra tolás: az utolsó előtti elemet az utolsónak megfelelő bittel balra tolja
rshift – jobbra tolás
and – bináris és
or – bináris vagy
xor – bináris kizáróvagy
+ - összeadás
- - kivonás (az utolsó előttiből vonja ki az utolsót)
* - szorzás
/ - osztás
mod – maradék osztás

7.3 Státusz kijelzés

A stack feltöltöttségét, és maximális kihasználtságát az eredmény kijelző terület felső sorában található két szám jelzi. Az első az aktuálisan stack-ben található értékek számát jelzi (végeredménynél ez 1), a második szám pedig azt mutatja meg, hogy a teljes műveletvégzés során mennyi volt a legnagyobb egyszerre a stack-ben lévő adatok száma. A ball felső sarokban a billentyűzetről beolvasott scan-kódok láthatóak. A ball alsó sarokban pedig egy milliszekundum számláló és alatta egy frame számláló található (vertical blanknél növekszik).

8 Felhasznált irodalmak

A lábjegyzetekben említetteket itt nem sorolom fel újra.

- Peter Norton: Az IBM PC programozása, ISBN:963 10 9421 9, Műszaki könyvkiadó Budapest, 1992 (Programmer's Guide to the IBM PC)
- http://osdir.com/ml/python.myhdl/2005-08/msg00017.html (Dual-Port Block RAM generations with Xilinx ISE)
- XPS General Purpose Input/Output (GPIO) (v2.00.a)
  DS569 July 17, 2009
  \Xilinx\11.1\EDK\hw\XilinxProcessorIPLib\pcores\xps_gpio_v2_00_a\doc\xps_gpio.pdf
  xps_gpio.pdf

- XPS Interrupt Controller (v2.00a)
  DS572 July 20, 2009
  \Xilinx\11.1\EDK\hw\XilinxProcessorIPLib\pcores\xps_intc_v2_00_a\doc\xps_intc.pdf
  xps_intc.pdf
- Fixed Interval Timer (FIT)
  fit_timer.pdf
- Block RAM:
  \Xilinx\11.1\EDK\hw\XilinxProcessorIPLib\pcores\bram_block_v1_00_a\doc\bram_block.pdf
- Spartan-3E FPGA Family: Data Sheet
  DS312 (v3.8) August 26, 2009
  Block RAM
- Spartan-3 Generation FPGA User Guide
  UG331 (v1.5) January 21, 2009
  Chapter 3: Using Digital Clock Managers (DCMs)
  Chapter 5: Using Block RAM
  Chapter 9: Using Carry and Arithmetic Logic
- Block Memory Generator v3.3
  DS512 September 16, 2009
- Platform Specification Format Reference Manual
  UG642, EDK 11.2
  \Xilinx\11.1\EDK\doc\usenglish\psf_rm.pdf
- OS and Libraries Document Collection
  UG 643 September 16, 2009
  \Xilinx\11.1\EDK\doc\usenglish\oslib_rm.pdf
- Embedded System Tools Reference Guide
  UG111 September 16, 2009
  \Xilinx\11.1\EDK\doc\usenglish\est_rm.pdf
  Xilinx Microprocessor Debugger (XMD)
- OPB IPIF Architecture
  DS414 August 18, 2004
  \Xilinx\11.1\EDK\doc\usenglish\opb_ipif_arch.pdf
- MicroBlaze Processor Reference Guide
  UG081 (v10.2)
  \Xilinx\11.1\EDK\doc\usenglish\mb_ref_guide.pdf
  (MicroBlaze Instruction Set)
- SDK
  \Xilinx\11.1\EDK\doc\usenglish\SDK_doc\index.html

9 Mellékletek

9.1 VGA.v

```
 0: `timescale 1ns / 1ps
 1: //////////////////////////////////////////////////////////////////////////////////
 2: // Create Date:    04:23:44 11/03/2009
 3: // Module Name:    VGA
 4: //////////////////////////////////////////////////////////////////////////////////
 5:
 6: //`define VGACOLORTEST
 7: //`define VGACHRROMTEST
 8: //`define VGACHRROMADDRTEST
 9:
10: module VGA(
11:           clk, rst,
12:           en, /// stand by
13:           R, G, B, HS, VS, /// VGA signals
14:           //chrClk,
15:           txtRd, txtAddr, txtD, /// text RAM
16:           chrRd, chrAddr, chrD, /// character ROM
17:           VB /// vertical blank
18:     );
19:
20:     input clk;
21:     input rst;
22:     input en;
23:
24:     output [1:0] R;
25:     output [1:0] G;
26:     output [1:0] B;
27:     output HS;
28:     output VS;
29:
30:    /// character ROM:
31:    //output chrClk; /// clock
32:    output chrRd; /// Read (Block RAM Enable)
33:    output [11:0]chrAddr; /// 12 bit = 4kB
34:    input  [7:0]chrD; /// 8 bit ( 8x16 Font )
35:    output txtRd; /// Read (Block RAM Enable)
36:    output [10:0]txtAddr; /// 11 bit = 2k *18 bit
37:    input  [17:0]txtD; /// 18 bit ( color + ASCII code )
38:
39:    output VB;
40:
41:   parameter Hres = 640; /// vizszintes lathato felbontas [px]
42:   parameter Vres = 400; /// fuggoleges lathato felbontas [px]
43:   parameter HSpolXor = 1; /// neg HS polaritas
44:   parameter VSpolXor = 0; /// pos VS polaritas
45:
46:   parameter rightBorder = 8+8+8 +4; /// [px] == Front Porch +4
47:   parameter HsyncLen = 96;          /// [px]
48:   parameter leftBorder = 40+8  -4;  /// [px] == Back Porch -4
49:
50:   parameter bottomBorder = 7+5; /// [line] == Front Porch
51:   parameter VsyncLen = 2;       /// [line]
52:   parameter topBorder = 28+7;   /// [line] == Back Porch
53:
54:   parameter HCMSB = 9; /// Horizontal counter bit count-1 (MSB index)
55:   parameter VCMSB = 9; /// Vertical counter bit count-1 (MSB index)
56:
57:   parameter BLINKBITS = 6;    /// cursor Blink time divider counter
58:   parameter textaddrMSB = 10;
59:   parameter txtResX = 80;     /// Hres/8
60:
61:   /// 640x400 => 800x449, 640x480 => 800x525
62:   reg [HCMSB:0]Hcnt; /// 0..1023
63:   reg [VCMSB:0]Vcnt; /// 0..1023 (vertical blank...)
```

```
64:
65: ////////////////////////////////////////////////////////////////
66: /// horizontal counter + sync:
67:   reg Hend;
68:   always @(posedge clk)
69:     Hend <= (Hcnt == Hres+rightBorder+HsyncLen+leftBorder-1-1 )? 1 : 0;
70:
71:   always @(posedge clk, posedge rst) begin
72:     if (rst)
73:           Hcnt <= 0;
74:    else
75:           Hcnt <= Hend ? 0 : Hcnt+1; //(Hcnt < Hres+rightBorder+HsyncLen+leftBorder) )? Hcnt+1 : 0;
76:   end
77:   /// clk  /  \__/  \__/  \__/  \__/  \__/  \__/  \__/  \__/  \_
78:   /// cntD  XXX797XXX798XXX799XXX 0 XXX 1 XXX 2 XXX 3 XXX 4 XXX   /// az X a valtast jeloli
79:   /// cntQ X 796 X 797 X 798 X 799 X  0  X  1  X  2  X  3  X 4    -- 800 total
80:   /// endD _____________///   \\\_____________________________   =?= 798
81:   /// endQ ___________________/     \_______________________________
82:
83:   /// cntD  >XX<  >XX<  >XX<  >XX<  >XX<  >XX<  >XX<  >XX<  >XX<
84:   /// cntQ >< 796>< 797>< 798><799 ><  0 ><  1 ><  2 ><  3 >< 4   -- 800 total
85:
86:   wire HSi;/// internal Horizontal Sync
87:   assign HSi = ( (Hcnt >= Hres+rightBorder) &&
88:                  (Hcnt < Hres+rightBorder+HsyncLen) )? 1 : 0;
89:
90:   assign HS = (HSi ^ HSpolXor) & en;
91:
92: ////////////////////////////////////////////////////////////////
93: /// vertical counter + sync:
94:   reg Vend;
95:   always @(posedge clk)
96:     Vend <= (Vcnt == Vres+bottomBorder+VsyncLen+topBorder-1-1 )? 1 : 0;
97:
98:   always @(posedge clk, posedge rst) begin
99:     if (rst)
100:       Vcnt <= 0;
101:    else
102:      if (Hend)
103:         Vcnt <= Vend ? 0 : Vcnt+1;
104:  end
105:
106:  wire VSi; /// internal Vertical Sync
107:  assign VSi = ( (Vcnt >= Vres+bottomBorder) &&
108:                 (Vcnt < Vres+bottomBorder+VsyncLen) )? 1 : 0;
109:
110:  assign VS = (VSi ^ VSpolXor) & en;
111:
112: ////////////////////////////////////////////////////////////////
113: /// data timing / pipeline flags
114:   wire f_txtAddrSet;
115:   wire f_txtDataSample, f_txtColorSample, f_chrLineSample, f_videoBlankSample;
116: /** alternativ verzio jelei:
117:   wire f_txtRowAddrInit, f_txtRowAddrInc;
118:   assign f_txtRowAddrInit = Vend && Hend;
119:   assign f_txtRowAddrInc = (Vcnt[3:0]==4'b1111) && Hend;
120: //assign f_txtRowAddrInit = (Vcnt==0) && (Hcnt==0);
     //assign f_txtRowAddrInc = (Vcnt[3:0]==4'b1111) && (Hcnt==0);
121: */
122:  assign f_txtAddrSet = Hcnt[2:0] == 0;
123:  assign f_txtDataSample = Hcnt[2:0] == 1;
124:  assign f_txtColorSample = Hcnt[2:0] == 4;
125:
126:  assign chrRd = Hcnt[2:0] == 2;
127:  assign f_chrLineSample = Hcnt[2:0] == 4;
128:  assign f_CursorSample = Hcnt[2:0] == 4;
129:
130:  assign f_videoBlankSample = Hcnt[2:0] == 4;
131:
132: ////////////////////////////////////////////////////////////////
```

```
133: ///  vertical blank interrupt
134:   reg rVB;
135:   always @(posedge clk)
136:     if (f_videoBlankSample)
137:       rVB <= (!(Vcnt<Vres));
138:   assign VB = rVB;
139: 
140: //////////////////////////////////////////////////////////////////
141: ///  blinking cursor
142:   wire [BLINKBITS-1:0] wblnk;
143:  FScnt #(.cbits(BLINKBITS)) blinkcnt (.clk(clk), .rst(rst), .ce(Vend & Hend), .q(wblnk));//framenkent lep
144:   /// kurzor poziciojat Vert Blanknel olvassuk
145:   wire curBlink;
146:   assign curBlink = wblnk[BLINKBITS-1] ; /// MSB
147:   wire curRead;
148:   assign curRead =  VSi;
149: 
150: //////////////////////////////////////////////////////////////////
151: /// video data blankolas kesleltetese (n px tolas jobbra)(left border n pixellel nagyobb)
152:   reg vidblnk; /// neve megteveszto lehet, az aktiv tartomanyt jeloli (todo:refactor)
153:   always @(posedge clk)
154:     if (f_videoBlankSample)
155:       vidblnk <= (en && (Hcnt<Hres) && (Vcnt<Vres));
156: 
157: //////////////////////////////////////////////////////////////////
158: /// Text RAM
159:   reg [textaddrMSB:0]rtxtRowAddr; /// base address: oszlop kezdocime
160:   always @(posedge clk) begin
161:     if (Hend) begin  /// minden sor vegen
162:       if (Vend) /// utolso sornal nullazzuk
163:          rtxtRowAddr <= 0;
164:       else
165:          if (Vcnt[3:0]==4'b1111) /// minden 16 sornal (16. sor vegen (#15), nem a #0 sor elejen)
166:            rtxtRowAddr <= rtxtRowAddr + txtResX; /// noveljuk 80-nal
167:     end
168:   end
169: /** alternativ verzio:
170:   always @(posedge clk)
171:     if (f_txtRowAddrInit) /// utolso sornal nullazzuk
172:      rtxtVAddr <= 0;
173:    else
174:       if (f_txtRowAddrInc) /// minden 16 sornal noveljuk 80-nal (16 sor vegen, nem a 0. sor elejen)
175:        rtxtVAddr <= rtxtVAddr + txtResX;
176: */
177: 
178:   reg [textaddrMSB:0]rtxtAddr; /// 11 bit: 80*25=2000 karakter (+szin)
179:   always @(posedge clk) begin
180:     if (curRead)
181:       rtxtAddr <= 11'h7ff; /// kurzor cime: 2047, TODO: parameter
182:     else
183:       if (f_txtAddrSet)
184:        rtxtAddr <= rtxtRowAddr + Hcnt[HCMSB:3];
185:   end
186:   assign txtAddr = rtxtAddr;
187: 
188:   assign txtRd = f_txtDataSample || curRead; /// curRead: az elso olvasas ervenytelen lesz, de nem szamit
189: /*
190:   reg [17:0]txtword;
191:   always @(posedge clk)
192:     if (f_txtDataSample)
193:       txtword <= txtD;
194: */
195:   reg [9:0]txtcolor;
196:   always @(posedge clk)
197:     if (f_txtColorSample)
198:       txtcolor <= txtD[17:8];
199:   /// a szint kesobb kell valtani (mintavetelezni), mint a karakter kodjat
200:   /// a szint a karakter bitmap soranak eloallitasaval egyszerre kell valtani
201: 
202:   wire [5:0]fgcolor; /// RGB
```

```
203:   wire [5:0]bgcolor; /// RGB
204:   wire [3:0]bgcc; /// 4 bit bgcolorcode
205: //  wire [7:0]charcode; /// 256 char table
206: //  assign {bgcc,fgcolor,charcode} = txtword;
207:   assign {bgcc,fgcolor} = txtcolor;
208: /*  assign bgcolor = bgcc[3] ? {bgcc[2],bgcc[2],bgcc[1],bgcc[1],bgcc[0],bgcc[0]} :
209:                              {bgcc[2],0      ,bgcc[1],0      ,bgcc[0],0      } ;  */
210:   assign bgcolor =  {bgcc[2],bgcc[3],bgcc[1],bgcc[3],bgcc[0],bgcc[3]} ;
211:
212: ////////////////////////////////////////////////////////////////
213: /// Character Ram
214: `ifdef VGACHRROMADDRTEST
215:   assign chrAddr = {Vcnt[7:4],Hcnt[6:3],Vcnt[3:0]}; /// adresss = teszt karakter tabla
216: `else
217:   //assign chrAddr = {charcode,Vcnt[3:0]}; /// adresss
218:   assign chrAddr = {txtD[7:0],Vcnt[3:0]}; /// adresss = ASCII karakter kod*16+ line %16
219:   /// a karakter soranak kivalasztasahoz a Vcnt-t hasznaljuk ami a kepernyosor vegen/elejen novekszik
220: `endif
221:   /// elso kozelites: CharROM[ (y/16*16 + x/8&15 )*16 + (y%16) ]
222:   /// masodik : CharROM[ (txtword & 0xff)*16 + (y%16) ]
223:   ///           where txtword = txtRAM[ (y/16)*80+(x/8) ]
224:   /// chrD - character data (egy orajel kesleltetes!)
225:   reg [7:0]chrbmline;
226:   always @(posedge clk)
227:     if (f_chrLineSample)
228:       chrbmline <= chrD;
229:     else
230:       chrbmline <= chrbmline << 1;
231:   /// chrbmline shiftelese, mindig MSB a kimenet,
232:   /// a karakter kepernyore rajzolasnal MSB first
233:
234: `ifdef VGACOLORTEST
235: /// Color Test Patter:
236:   wire [5:0]color;
237:   assign color = {Vcnt[7:6],Hcnt[8:5]};
238:   assign {R,G,B} = vidblnk ? color : 0; /// horizontal, vertical blank
239:
240: `elsif VGACHRROMTEST
241: /// Char ROM Test:
242:   wire [5:0]color;
243:   assign color = (chrbmline[7]) ? 6'b11_10_00: 6'b000000;
244:   assign {R,G,B} = vidblnk ? color : 0; /// horizontal, vertical blank
245:
246: `else
247: /// eles megjelenites szinekkel, kurzorral:
248:
249: ////////////////////////////////////////////////////////////////
250: /// Cursor : text ram last word18: ([17] ), [16] mode (block/underline),[15] enabled, [14:8] row, [7:0]
column
251:   reg [7:0]curCol;
252:   reg [7:0]curRow; /// eleg 7 bit is, az MSB beallitasaval el lehet tuntetni a kepernyorol a kurzort
253:   reg [1:0]curMode; /// ha curMode[0] ([16.bit]) == 1 -> block cursor
254:   always @(posedge clk)
255:     if (curRead) /// az elso olvasas ervenytelen lesz, de nem szamit (vertical blankben vagyunk)
256:       {curMode, curRow, curCol } <= txtD[17:0];
257:
258:   reg cursorselect; /// a kurzor kivalasztas terben es idoben
259:   always @(posedge clk)
260:     if (f_CursorSample) /// 8 pixelenkent mintavetelezni (f_chrLineSample-nal szinkronban)
261:       cursorselect <= curBlink /// eppen "vilagit" a kurzor
262:                       && (Hcnt[HCMSB:3] == curCol) && (Vcnt[VCMSB:4]==curRow) /// (block cursor)
263:                       && ((Vcnt[3:0]>=14) | curMode[0]); /// (also ket pixel sor cursor)
264:
265: ////////////////////////////////////////////////////////////////
266: /// color
267:   reg [5:0]color;
268:   wire colorselect;
269:   assign colorselect = chrbmline[7];// ^ (cursorselect); /// eloter-hatter szincsere ha kurzor
270:   always @(posedge clk) begin
271:     if (vidblnk)
```

```
272: //    color <= colorselect ? fgcolor : bgcolor;
273:    color <= cursorselect ? ~ (colorselect ? fgcolor : bgcolor)://szin invertelas (komplementer) ha kurzor
274:                              (colorselect ? fgcolor : bgcolor);
275:   //color <= (colorselect ? fgcolor : bgcolor) ^ (cursorselect ? 6'b111111 : 0); /// inverz kurzor
276:     else
277:       color <= 0;
278:   end
279:
280:   assign {R,G,B} = color; /// horizontal, vertical blank
281: `endif
282:
283: endmodule
```

9.2 Main.v

```
  0: `timescale 1ns / 1ps
  1: //////////////////////////////////////////////////////////////////////////////////
  2: // Create Date:    20:34:15 11/01/2009
  3: // Module Name:    main
  4: //////////////////////////////////////////////////////////////////////////////////
  5: module main(
  6:               rx,
  7:               tx,
  8:               fpga_0_LEDs_Displays_GPIO_IO_O_pin,
  9:               fpga_0_Switches_Buttons_GPIO_IO_I_pin,
 10:               fpga_0_SRAM_Mem_A_pin,
 11:               fpga_0_SRAM_Mem_CEN_pin,
 12:               fpga_0_SRAM_Mem_OEN_pin,
 13:               fpga_0_SRAM_Mem_WEN_pin,
 14:               fpga_0_SRAM_Mem_DQ_pin,
 15:               clk,
 16:               rst,
 17:               vgaR, vgaG, vgaB, vgaHS, vgaVS,
 18: //              ps2_clk1, ps2_data1,
 19:               ps2_clk, ps2_data
 20:     );
 21:   input clk;
 22:   input rst;
 23:   input rx;
 24:   output tx;
 25:   output [0:24] fpga_0_LEDs_Displays_GPIO_IO_O_pin;
 26:   input  [0:11] fpga_0_Switches_Buttons_GPIO_IO_I_pin;
 27:   output [15:31] fpga_0_SRAM_Mem_A_pin;
 28:   output fpga_0_SRAM_Mem_CEN_pin;
 29:   output fpga_0_SRAM_Mem_OEN_pin;
 30:   output fpga_0_SRAM_Mem_WEN_pin;
 31:   inout  [0:7] fpga_0_SRAM_Mem_DQ_pin;
 32:
 33:   output [1:0] vgaR;
 34:   output [1:0] vgaG;
 35:   output [1:0] vgaB;
 36:   output vgaHS;
 37:   output vgaVS;
 38:
 39:   input  ps2_clk;
 40:   input  ps2_data;
 41:
 42: //  inout  ps2_clk1; inout  ps2_data1;
 43:
 44:
 45: /************************************************************************************
 46:  graphical resolution   640x400   640x480   800x600            1024x768  1152x864  1280x1024 1400x1050
                 1600x1200 1920x1200
 47:  text mode resolution    80 x 25   80 x 30   100x37.5  128x48    144x54    160x64    175x65.6  200x75
                 240x75
 48:  txt ram [*18 bit]      2000     2400       3800      6144               7776      10240 !            11550
!                15000 !             18000 !
 49:
 50:  refresh rate [Hz]      70        60        60        60        60        60        60        60        60
```

```
 51:  pixel clock [MHz]      25.175    25.175    40                  65                  81.62       108
122.61                 162                 193.16
 52:  clk MUL/DIV (16 MHz)   11/7      11/7      5/2         4/1         31/6        27/4        23/3
10/1        12/1
 53:  H x V counter range    800x449   800x525   1056x628    1344x806    1520x895    1688x1066   1880x1087
2160x1250   2592x1242
 54:
 55:  refresh rate [Hz]      85        75        75        75        75        75        75        75
 56:  pixel clock [MHz]      31.5      31.5      49.5                78.8                108                 135
                  155.85      202.5
 57:  clk MUL/DIV (16 MHz)   2/1       2/1     31/10       5/1         27/4        17/2        29/3        25/2
 58:  H x V counter range    832x445   840x500 1056x625    1312x800    1600x900    1688x1066   1896x1094
2160x1250
 59:
 60:  refresh rate [Hz]                85        85        85        85        85        85
 61:  pixel clock [MHz]                36                  56.25               94.5        119.65      157.5
                     179.26
 62:  clk MUL/DIV (16 MHz)             9/4     7/2         6/1         15/2        10/1        11/1
 63:  H x V counter range              832x509 1048x631    1376x808    1552x907    1728x1072   1896x1096
 64:
 65:  refresh rate [Hz]                100       100       100       100       100       100
 66:  pixel clock [MHz]                43.16     68.18       113.31     143.47      190.96          214.39
 67:  clk MUL/DIV (16 MHz)             27/10   17/4        7/1         9/1         12/1        27/2
 68:  H x V counter range              848x509 1072x636    1392x814    1568x915    1760x1085   1912x1103
 69:
 70:  refresh rate [Hz]                          56
 71:  pixel clock [MHz]                          36
 72:  clk MUL/DIV (16 MHz)                       9/4
 73:  H x V counter range                        1024x625
 74:
 75:  ********************************************************************************/
 76: /// Orajel eloallitas
 77:   parameter VGACLKMUL = 27;
 78:   parameter VGACLKDIV = 17;
 79: //  parameter VGACLKMUL = 7;//27;
 80: //  parameter VGACLKDIV = 1;//17;
 81:
 82:   wire vgaclk;
 83:   wire clkfb;
 84:
 85:
 86:   DCM_SP #(
 87:       .CLKFX_DIVIDE(VGACLKDIV),   // Can be any integer from 1 to 32
 88:       .CLKFX_MULTIPLY(VGACLKMUL) // Can be any integer from 2 to 32
 89:   ) DCM_SP_inst (
 90:       .CLKFX(vgaclk),   // DCM CLK synthesis out (M/D)
 91:       .CLK0(clkfb),     // 0 degree DCM CLK output
 92:       .CLKFB(clkfb),
 93:       .CLKIN(clk),   // Clock input (from IBUFG, BUFG or DCM)
 94:       .RST(rst)        // DCM asynchronous reset input
 95:   );
 96:
 97: /// Memoria bekotes
 98:   /// VGA text RAM
 99:   wire [10:0] txtaddr; /// 2048 deep
100:   wire [17:0] txtD; /// 18 bit wide
101:   wire txtRd;
102:   /// MB text RAM
103:   wire twr, ten;
104:   wire tclk;
105:   wire [10:0] taddr; /// 2048 deep
106:   wire [1:0]faketaddr;
107:   wire [17:0] tuP2BR; /// 18 bit wide
108:   wire [17:0] tBR2uP; /// 18 bit wide
109:   /// VGA Character "ROM"  /// Dual VGA displaynel lenne csak olvashato
110:   wire [11:0] craddr; /// 4096 deep
111:   wire [7:0]  crD; /// 8 bit wide
112:   wire crRd;
113:   /// MB char "ROM"
114:   wire cwr, cen;
```

```
115:   wire cclk;
116:   wire [11:0] caddr; /// 4096 deep
117:   wire [1:0]fakecaddr;
118:   wire [8:0] cuP2BR; /// 9 bit wide
119:   wire [8:0] cBR2uP; /// 9 bit wide
120:
121:   charram chrF8x16(
122:       .clkA(vgaclk), .addrA(craddr), .DA(9'h1ff), .QA(crD), .wrA(0), .enA(crRd),
123: //      .clkB(1), .addrB(12'hfff), .DB(9'h1ff), .QB(), .wrB(0), .enB(0) /// not Used (Yet)
124:       .clkB(cclk), .addrB(caddr), .DB(cuP2BR), .QB(cBR2uP), .wrB(cwr), .enB(cen)
125:   );
126:
127:   textram2k txt80x25(
128:       .clkA(vgaclk), .addrA(txtaddr), .DA(9'h1ff), .QA(txtD), .wrA(0), .enA(txtRd),
129:       .clkB(tclk), .addrB(taddr), .DB(tuP2BR), .QB(tBR2uP), .wrB(twr), .enB(ten)
130:   );
131:
132: /// VGA periferia
133:   wire vertblnk;
134:   wire vgaen;
135:   VGA #(
136: /*
137:             .Hres(1024),
138:             .Vres(768),
139:             .HSpolXor(1), /// neg HS
140:             .VSpolXor(0), /// pos VS
141:             .rightBorder(72 +4), // px
142:             .HsyncLen(112), // px
143:             .leftBorder(184  -4), // px
144:             .bottomBorder(1), // line
145:             .VsyncLen(3), // line
146:             .topBorder(42), // line
147:             .HCMSB(10), /// Horizontal counter bit count-1 (MSB index)
148:             .VCMSB(9), /// Vertical counter bit count-1 (MSB index)
149:             .BLINKBITS(6),
150:             .textaddrMSB(10),
151: */
152:             .txtResX(80)
153:        ) VGA1(
154:             .clk(vgaclk), .rst(rst), .en(vgaen),
155:             .R(vgaR), .G(vgaG), .B(vgaB), .HS(vgaHS), .VS(vgaVS),
156:                         .txtRd(txtRd), .txtAddr(txtaddr), .txtD(txtD),
157:                         .chrRd(crRd), .chrAddr(craddr), .chrD(crD),//.chrClk(crclk),
158:             .VB(vertblnk)
159:                          );
160:
161: // MicroBlaze System
162: (* BOX_TYPE = "user_black_box" *)
163:   system mbsys (
164:     .fpga_0_RS232_RX_pin(rx),
165:     .fpga_0_RS232_TX_pin(tx),
166:     .fpga_0_LEDs_Displays_GPIO_IO_O_pin(fpga_0_LEDs_Displays_GPIO_IO_O_pin),
167:     .fpga_0_Switches_Buttons_GPIO_IO_I_pin(fpga_0_Switches_Buttons_GPIO_IO_I_pin),
168:     .fpga_0_SRAM_Mem_A_pin(fpga_0_SRAM_Mem_A_pin),
169:     .fpga_0_SRAM_Mem_CEN_pin(fpga_0_SRAM_Mem_CEN_pin),
170:     .fpga_0_SRAM_Mem_OEN_pin(fpga_0_SRAM_Mem_OEN_pin),
171:     .fpga_0_SRAM_Mem_WEN_pin(fpga_0_SRAM_Mem_WEN_pin),
172:     .fpga_0_SRAM_Mem_DQ_pin(fpga_0_SRAM_Mem_DQ_pin),
173:     .fpga_0_clk_1_sys_clk_pin(clkfb),
174:     .fpga_0_rst_1_sys_rst_pin(rst)
175:     ,
176:     .vga_lmb_bram_text_BRAM_Rst_A_pin(),
177:     .vga_lmb_bram_text_BRAM_Clk_A_pin(tclk),
178:     .vga_lmb_bram_text_BRAM_EN_A_pin(ten),
179:     .vga_lmb_bram_text_BRAM_WEN_A_pin(twr),
180:     .vga_lmb_bram_text_BRAM_Addr_A_pin({taddr,faketaddr}),
181:     .vga_lmb_bram_text_BRAM_Din_A_pin(tBR2uP),
182:     .vga_lmb_bram_text_BRAM_Dout_A_pin(tuP2BR)
183:               ,
184:     .vga_lmb_bram_char_BRAM_Rst_A_pin(),
```

```
185:     .vga_lmb_bram_char_BRAM_Clk_A_pin(cclk),
186:     .vga_lmb_bram_char_BRAM_EN_A_pin(cen),
187:     .vga_lmb_bram_char_BRAM_WEN_A_pin(cwr),
188:     .vga_lmb_bram_char_BRAM_Addr_A_pin({caddr,fakecaddr}),
189:     .vga_lmb_bram_char_BRAM_Din_A_pin(cBR2uP),
190:     .vga_lmb_bram_char_BRAM_Dout_A_pin(cuP2BR)
191:               ,
192:     .ps2_gpio_GPIO_IO_I_pin( {ps2_clk, ps2_data})
193:               ,
194: //    .xps_ps2_0_PS2_1_DATA(ps2_data1), .xps_ps2_0_PS2_1_CLK(ps2_clk1),
195:
196:     .vblank(vertblnk)
197:
198:   );
199:
200:   assign vgaen = 1;
201:
202: endmodule
```

9.3 charram.v

```
  0: `timescale 1ns / 1ps
  1: //////////////////////////////////////////////////////////////////////////////////
  2: // Create Date:    21:17:54 11/16/2009
  3: // Module Name:    charram
  4: //////////////////////////////////////////////////////////////////////////////////
  5: module charram(
  6:     clkA, addrA, DA, QA, wrA, enA,
  7:     clkB, addrB, DB, QB, wrB, enB
  8:     );
  9:     input clkA;
 10:     input [11:0] addrA; /// 4096 deep
 11:     input [8:0]  DA; /// 9 bit wide
 12:     output [8:0] QA; /// 9 bit wide
 13:               input wrA;
 14:               input enA;
 15:
 16:     input clkB;
 17:     input [11:0] addrB;
 18:     input [8:0]  DB;
 19:     output [8:0] QB;
 20:               input wrB;
 21:               input enB;
 22:
 23: /**
 24:       2 ways to split memory blocks:
 25:     - by address space
 26:       - by adress MSB (paging)
 27:       same adress, data in, out lines (data out needs multiplexer)
 28:       RAM blocks (pages) needs chip select, adress decoder
 29:       both RAM uses full 9 bit width
 30:     - by data width
 31:       memories provide only part of the data,
 32:       both RAM block work all the time
 33:       same adress lines
 34:       RAM has half data width : {dh[7:4], dl[3:0]}
 35:       no mux needed
 36:       but 9/2 =4 -> not full RAM capacity can be used
 37:       (it's not a problem for 8x16 characters)
 38:       but generating the initial content for charcater map (bitmap typeface)
 39:       is more comlicated than dividing the data into 2 RAMs in the middle
 40:       data needs to be interleaved (dividing the first line of character into 2 nibbles,
           and using the second line to get a complete byte)
 41:       ('data2mem' utility can do this)
 42: */
 43:
 44: /**
 45:  Dividing by Adress Space:
 46: */
 47:    wire enAl,enAh,enBl,enBh;
```

```
48:    assign enAl=enA & (addrA[11]==0);
49:    assign enAh=enA & (addrA[11]==1);
50:    assign enBl=enB & (addrB[11]==0);
51:    assign enBh=enB & (addrB[11]==1);
52:
53:    wire wrAl,wrAh,wrBl,wrBh;
54:    assign wrAl=wrA & (addrA[11]==0);
55:    assign wrAh=wrA & (addrA[11]==1);
56:    assign wrBl=wrB & (addrB[11]==0);
57:    assign wrBh=wrB & (addrB[11]==1);
58:
59:    wire [8:0]QAl;
60:    wire [8:0]QAh;
61:    wire [8:0]QBl;
62:    wire [8:0]QBh;
63:    assign QA= (addrA[11]==0) ? QAl : QAh; /// MUX
64:    assign QB= (addrB[11]==0) ? QBl : QBh; /// MUX
65:
66:    RAMB16_S9_S9 #(
67:       .INIT_A(9'h000),  // Value of output RAM registers on Port A at startup
68:       .INIT_B(9'h000),  // Value of output RAM registers on Port B at startup
69:       .SRVAL_A(9'h000), // Port A output value upon SSR assertion
70:       .SRVAL_B(9'h000), // Port B output value upon SSR assertion
71:       .WRITE_MODE_A("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE
72:       .WRITE_MODE_B("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE
73:       .SIM_COLLISION_CHECK("ALL") // "NONE", "WARNING_ONLY", "GENERATE_X_ONLY", "ALL"
74:    ) vgarom_low (
75:       .CLKA(clkA),     // Port A Clock
76:       .ADDRA(addrA[10:0]),  // Port A 11-bit Address Input
77:       .DIA(DA[7:0]),       // Port A 8-bit Data Input
78:       .DIPA(DA[8]),     // Port A 1-bit parity Input
79:       .DOA(QAl[7:0]),   // Port A 8-bit Data Output
80:       .DOPA(QAl[8]), // Port A 1-bit Parity Output
81:       .WEA(wrAl),       // Port A Write Enable Input
82:       .ENA(enAl),       // Port A RAM Enable Input
83:       .SSRA(0),     // Port A Synchronous Set/Reset Input
84:
85:       .CLKB(clkB),     // Port B Clock
86:       .ADDRB(addrB[10:0]),  // Port B 11-bit Bddress Input
87:       .DIB(DB[7:0]),       // Port B 8-bit Data Input
88:       .DIPB(DB[8]),     // Port B 1-bit parity Input
89:       .DOB(QBl[7:0]),   // Port B 8-bit Data Output
90:       .DOPB(QBl[8]), // Port B 1-bit Parity Output
91:       .WEB(wrBl),       // Port B Write Enable Input
92:       .ENB(enBl),       // Port B RBM Enable Input
93:       .SSRB(0)     // Port B Synchronous Set/Reset Input
94:    );
95:
96:    RAMB16_S9_S9 #(
97:       .INIT_A(9'h000),  // Value of output RAM registers on Port A at startup
98:       .INIT_B(9'h000),  // Value of output RAM registers on Port B at startup
99:       .SRVAL_A(9'h000), // Port A output value upon SSR assertion
100:      .SRVAL_B(9'h000), // Port B output value upon SSR assertion
101:      .WRITE_MODE_A("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE
102:      .WRITE_MODE_B("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE
103:      .SIM_COLLISION_CHECK("ALL") // "NONE", "WARNING_ONLY", "GENERATE_X_ONLY", "ALL"
104:   ) vgarom_high (
105:      .CLKA(clkA),     // Port A Clock
106:      .ADDRA(addrA[10:0]),  // Port A 11-bit Address Input
107:      .DIA(DA[7:0]),       // Port A 8-bit Data Input
108:      .DIPA(DA[8]),     // Port A 1-bit parity Input
109:      .DOA(QAh[7:0]),   // Port A 8-bit Data Output
110:      .DOPA(QAh[8]), // Port A 1-bit Parity Output
111:      .WEA(wrAh),       // Port A Write Enable Input
112:      .ENA(enAh),       // Port A RAM Enable Input
113:      .SSRA(0),     // Port A Synchronous Set/Reset Input
114:
115:      .CLKB(clkB),     // Port B Clock
116:      .ADDRB(addrB[10:0]),  // Port B 11-bit Bddress Input
117:      .DIB(DB[7:0]),       // Port B 8-bit Data Input
```

```
118:       .DIPB(DB[8]),    // Port B 1-bit parity Input
119:       .DOB(QBh[7:0]),   // Port B 8-bit Data Output
120:       .DOPB(QBh[8]), // Port B 1-bit Parity Output
121:       .WEB(wrBh),      // Port B Write Enable Input
122:       .ENB(enBh),      // Port B RBM Enable Input
123:       .SSRB(0)    // Port B Synchronous Set/Reset Input
124:    );
125:
126: /**
127:   VGA-ROM.F16 data from fntcol16.zip
128:               ftp://ftp.simtel.net/pub/simtelnet/msdos/screen/fntcol16.zip
129:   (IBM compatible VGA BIOS)
130:   (bitmap typefaces are not protected by copyright laws)
131:  */
132: defparam vgarom_low.INIT_00  = 256'h000000007E818199BD8181A5817E0000_8100000000000000000000000000081;
133: defparam vgarom_low.INIT_01  = 256'h0000000010387CFEFEFEFE6C00000000_000000007EFFFFE7C3FFFFDBFF7E0000;
134: defparam vgarom_low.INIT_02  = 256'h000000003C1818E7E7E73C3C18000000_000000000010387CFE7C381000000000;
…
195: defparam vgarom_low.INIT_3F  = 256'h0000000000FEC6C6C66C381000000000_0000000000000000000000000DC760000;
196:
197: defparam vgarom_high.INIT_00  = 256'h0000000076CCCCCCCCCCCC0000CC0000_00007C060C3C66C2C0C0C0C2663C0000;
198: defparam vgarom_high.INIT_01  = 256'h0000000076CCCCCC7C0C78006C381000_000000007CC6C0C0FEC67C0030180C00;
…
260: defparam vgarom_high.INIT_3F  = 256'haa0000000000000000000000000000055_00000000007C7C7C7C7C7C7C00000000;
261:
262: endmodule
```

9.4 textram2k.v

```
  0: `timescale 1ns / 1ps
  1: //////////////////////////////////////////////////////////////////////////////////
  2: // Create Date:    16:06:16 11/14/2009
  3: // Module Name:    textram2k
  4: //////////////////////////////////////////////////////////////////////////////////
  5: module textram2k(
  6:     clkA, addrA, DA, QA, wrA, enA,
  7:     clkB, addrB, DB, QB, wrB, enB
  8:     );
  9:     input clkA;
 10:     input [10:0] addrA; /// 2048 deep
 11:     input [17:0]  DA; /// 18 bit wide
 12:     output [17:0] QA; /// 18 bit wide
 13:               input wrA;
 14:               input enA;
 15:
 16:     input clkB;
 17:     input [10:0] addrB;
 18:     input [17:0]  DB;
 19:     output [17:0] QB;
 20:               input wrB;
 21:               input enB;
 22:
 23: /**
 24:   Bit kiosztas:
 25:   1111 1111
 26:   7654 321098 76543210
 27:   irgb rrggbb CCCCCCCC
 28:   back  fore    ASCII
 29:     ground      code
 30:      color
 31:   ram 1 data  [7:0]  == char code (ASCII)
 32:   ram 2 data  [15:8] == { bg color g,b , char color ([13:8])}
 33:   ram 2 parity [16]  == bg color r
 34:   ram 1 parity [17]  == bg intensity
 36: */
 37:
 38:    RAMB16_S9_S9 #(
 39:       .INIT_A(9'h000),  // Value of output RAM registers on Port A at startup
 40:       .INIT_B(9'h000),  // Value of output RAM registers on Port B at startup
 41:       .SRVAL_A(9'h000), // Port A output value upon SSR assertion
```

```
42:       .SRVAL_B(9'h000), // Port B output value upon SSR assertion
43:       .WRITE_MODE_A("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE
44:       .WRITE_MODE_B("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE
45:       .SIM_COLLISION_CHECK("ALL"), // "NONE", "WARNING_ONLY", "GENERATE_X_ONLY", "ALL"
46:       // The next set of INITP_xx are for the parity bits
47:       // Address 0 to 511
48:       .INITP_00(256'h0000000000000000000000000000000000000000000000000000000000000000),
49:       .INITP_01(256'h0000000000000000000000000000000000000000000000000000000000000000),
50:       // Address 512 to 1023
51:       .INITP_02(256'h0000000000000000000000000000000000000000000000000000000000000000),
52:       .INITP_03(256'h0000000000000000000000000000000000000000000000000000000000000000),
53:       // Address 1024 to 1535
54:       .INITP_04(256'h0000000000000000000000000000000000000000000000000000000000000000),
55:       .INITP_05(256'h0000000000000000000000000000000000000000000000000000000000000000),
56:       // Address 1536 to 2047
57:       .INITP_06(256'h0000000000000000000000000000000000000000000000000000000000000000),
58:       .INITP_07(256'h0000000000000000000000000000000000000000000000000000000000000000)
59:
60:    ) vgatxt_char (
61:       .CLKA(clkA),    // Port A Clock
62:       .ADDRA(addrA[10:0]),  // Port A 11-bit Address Input
63:       .DIA(DA[7:0]),      // Port A 8-bit Data Input
64:       .DIPA(DA[17]),    // Port A 1-bit parity Input
65:       .DOA(QA[7:0]),   // Port A 8-bit Data Output
66:       .DOPA(QA[17]), // Port A 1-bit Parity Output
67:       .WEA(wrA),      // Port A Write Enable Input
68:       .ENA(enA),      // Port A RAM Enable Input
69:       .SSRA(0),    // Port A Synchronous Set/Reset Input
70:
71:       .CLKB(clkB),    // Port B Clock
72:       .ADDRB(addrB[10:0]),  // Port B 11-bit Bddress Input
73:       .DIB(DB[7:0]),      // Port B 8-bit Data Input
74:       .DIPB(DB[17]),    // Port B 1-bit parity Input
75:       .DOB(QB[7:0]),   // Port B 8-bit Data Output
76:       .DOPB(QB[17]), // Port B 1-bit Parity Output
77:       .WEB(wrB),      // Port B Write Enable Input
78:       .ENB(enB),      // Port B RBM Enable Input
79:       .SSRB(0)    // Port B Synchronous Set/Reset Input
80:    );
81:
82:    RAMB16_S9_S9 #(
83:       .INIT_A(9'h000),  // Value of output RAM registers on Port A at startup
84:       .INIT_B(9'h000),  // Value of output RAM registers on Port B at startup
85:       .SRVAL_A(9'h000), // Port A output value upon SSR assertion
86:       .SRVAL_B(9'h000), // Port B output value upon SSR assertion
87:       .WRITE_MODE_A("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE
88:       .WRITE_MODE_B("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE
89:       .SIM_COLLISION_CHECK("ALL"), // "NONE", "WARNING_ONLY", "GENERATE_X_ONLY", "ALL"
90:
91:       // The following INIT_xx declarations specify the initial contents of the RAM
92:       // Address 0 to 511
93:       .INIT_00( 256'h3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c
                    _3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c),
94:       .INIT_01( 256'h3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c
                    _3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c),
…
159:       .INIT_3F( 256'h18_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c
                    _3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c_3c),
160:
161:       // The next set of INITP_xx are for the parity bits
162:       // Address 0 to 511
163:       .INITP_00(256'h0000000000000000000000000000000000000000000000000000000000000000),
164:       .INITP_01(256'h0000000000000000000000000000000000000000000000000000000000000000),
165:       // Address 512 to 1023
166:       .INITP_02(256'h0000000000000000000000000000000000000000000000000000000000000000),
167:       .INITP_03(256'h0000000000000000000000000000000000000000000000000000000000000000),
168:       // Address 1024 to 1535
169:       .INITP_04(256'h0000000000000000000000000000000000000000000000000000000000000000),
170:       .INITP_05(256'h0000000000000000000000000000000000000000000000000000000000000000),
171:       // Address 1536 to 2047
```

```
172:       .INITP_06(256'h0000000000000000000000000000000000000000000000000000000000000000),
173:       .INITP_07(256'h0000000000000000000000000000000000000000000000000000000000000000)
174:
175:    ) vgatxt_color (
176:       .CLKA(clkA),     // Port A Clock
177:       .ADDRA(addrA[10:0]),  // Port A 11-bit Address Input
178:       .DIA(DA[15:8]),       // Port A 8-bit Data Input
179:       .DIPA(DA[16]),     // Port A 1-bit parity Input
180:       .DOA(QA[15:8]),    // Port A 8-bit Data Output
181:       .DOPA(QA[16]), // Port A 1-bit Parity Output
182:       .WEA(wrA),       // Port A Write Enable Input
183:       .ENA(enA),       // Port A RAM Enable Input
184:       .SSRA(0),     // Port A Synchronous Set/Reset Input
185:
186:       .CLKB(clkB),     // Port B Clock
187:       .ADDRB(addrB[10:0]),  // Port B 11-bit Bddress Input
188:       .DIB(DB[15:8]),       // Port B 8-bit Data Input
189:       .DIPB(DB[16]),     // Port B 1-bit parity Input
190:       .DOB(QB[15:8]),    // Port B 8-bit Data Output
191:       .DOPB(QB[16]), // Port B 1-bit Parity Output
192:       .WEB(wrB),       // Port B Write Enable Input
193:       .ENB(enB),       // Port B RBM Enable Input
194:       .SSRB(0)     // Port B Synchronous Set/Reset Input
195:    );
196:
197: // Verilog initialization records.
198: // Release 11.1 - Data2MEM L.57, build 1.6 Jun 5, 2009
199: // Copyright (c) 1995-2009 Xilinx, Inc.  All rights reserved.
200: // Command: data2mem -bm vga_txt_p.bmm -bd bootscr.dat.mem -o v bootscr.dat.v
201: // Created on Friday November 27, 2009 - 11:00:41 pm
202: defparam vgatxt_char.INIT_00  = 256'h2020202020202020202020202020202020202020202020202020202020202020;
…
265: defparam vgatxt_char.INIT_3F  = 256'h2c2020202020202020202020202020202020202020202020202020202020202020;
266:
267: endmodule
```

9.5 FScnt.v

```
…
17: module FScnt(clk, rst, ce, q, cy);
18:   parameter cbits = 16;
19:   parameter initval = 16'h0000;
20:               input clk;
21:               input rst;
22:               input ce;
23:               output [cbits-1:0] q;
24:               output cy;
25:
26:               reg [cbits-1:0] counter;
27:               wire [cbits:0] cntnext;
28:
29:               assign cntnext = counter + 1;
30:
31:               always @(posedge clk, posedge rst) begin
32:                   if (rst)
33:                       counter <= initval; /// aszinkron
34:                   else
35:                       if (ce)
36:                           counter <= cntnext[cbits-1:0];
37:               end
38:
39:               assign q = counter;
40:               assign cy = cntnext[cbits];
41:
42: endmodule
```

9.6 Excel macro

```
Private Sub btnFindFreq_Click()
```

```
 sfreq = Cells(37, 3)
 mindelta = sfreq
 mdr = 3
 mdc = 3
 For r = 3 To 34
  For c = 3 To 33
    delta = Abs(Cells(r, c) - sfreq)
    If mindelta > delta Then
      mindelta = delta
      mdr = r
      mdc = c
    End If
  Next
 Next
 Cells(38, 3) = sfreq
 Cells(39, 3) = " " + CStr(Cells(2, mdc)) + "/ " + CStr(Cells(mdr, 2))
 Cells(40, 3) = Cells(mdr, mdc)
 Cells(41, 3) = Cells(2, 2) * Cells(2, mdc) / Cells(mdr, 2)
 Cells(mdr, mdc).Interior.ColorIndex = Cells(37, 3).Interior.ColorIndex
End Sub
```

9.7 Perl scrip (proc.pl)

```
if ((scalar(@ARGV) < 1) || ($ARGV[0] eq '') ) {
     print "Usage: $0 <file>\n";
     exit;
}
$fn = $ARGV[0];
$fn =~ /^([\d]*)x([\d]*)@([\d]*)Hz$/;
print "$1, $2, $3";

open ffile, $fn or die $!;
@fln=<ffile>;
$il = @fln[23];
close ffile;

$il =~ /^.*rate[\D]*([\d]+)[\D]+/i; # Hz
print ", $1";
#$il =~ /Vertical[\D]*([\d]+)[\D]*([\d\. kmhz]+)/i;
$il =~ /Vertical[\D]*([\d\.]+)[\D]+/i; # kHz
print ", $1";
$il =~ /Pixel [\D]*([\d\.]+)[\D]+/i; # MHz
print ", $1";

$il =~ /horizontal sync pulse is ([\w]+)/i; # positive / negative
print ", $1";
$il =~ /horizontal.*Visible[\D]*([\d]+)[\D]+([\d\.]+).*Vertical/i; # us
print ", $1, $2";
$il =~ /horizontal.*Front[\D]*([\d]+)[\D]+([\d\.]+).*Vertical/i; # us
print ", $1, $2";
$il =~ /horizontal sync.*Sync[\D]*([\d]+)[\D]+([\d\.]+).*Vertical/i; # us
print ", $1, $2";
$il =~ /horizontal.*Back[\D]*([\d]+)[\D]+([\d\.]+).*Vertical/i; # us
print ", $1, $2";
$il =~ /horizontal.*Whole[\D]*([\d]+)[\D]+([\d\.]+).*Vertical/i; # us
print ", $1, $2";


$il =~ /vertical sync pulse is ([\w]+)/i; # positive / negative
print ", $1";
$il =~ /vertical sync.*Visible[\D]*([\d]+)[\D]+([\d\.]+)/i; #ms
print ", $1, $2";
$il =~ /vertical sync.*Front[\D]*([\d]+)[\D]+([\d\.]+)/i;#ms
print ", $1, $2";
$il =~ /vertical sync.*Sync[\D]*([\d]+)[\D]+([\d\.]+)/i;#ms
print ", $1, $2";
$il =~ /vertical sync.*Back[\D]*([\d]+)[\D]+([\d\.]+)/i;#ms
print ", $1, $2";
$il =~ /vertical sync.*Whole[\D]*([\d]+)[\D]+([\d\.]+)/i;#ms
print ", $1, $2\n";
```

9.8 Perl scrip hivó Batch file

```
@echo off
set proc=perl proc.pl

for %%f in (*.) do (
  if exist %%f (
rem    echo %%f >>res.txt
    %proc% %%f >>res.csv
  )
```

9.9 BRAM init generáló Batch file

```
:l1
echo Bin to hexadecimal VMem format
bin2mem.exe %1 %1.mem 4096
goto l4
:l1b
srec_cat %1 -binary -o %1.mem -vmem 8
goto l4
:l2
echo generate BMM file
data2mem -mf p chartable PICOBLAZE 0 a charbitmaps b 0x0000 8 s RAMB16 0x0800 8 vgarom_low
s RAMB16 0x0800 8 vgarom_high -o p vgarom.bmm
goto l4
:l4
data2mem -bm vga_txt_p.bmm -bd %1.mem -o v %1.v
```

9.10 LMB ctrl tcl patch (lmb_bram_if_cntlr_v2_1_0.tcl.patch)

```
*** old\lmb_bram_if_cntlr_v2_1_0.tcl    Sun Nov 01 23:44:04 2009
--- lmb_bram_if_cntlr_v2_1_0.tcl        Sun Nov 01 22:39:15 2009
***************
*** 60,66 ****
      set instname   [xget_value $mhsinst "parameter" "INSTANCE"]

      if {[bus_is_connected $mhsinst "BRAM_PORT"] == 0} {
!       error "The BRAM_PORT interface is not correctly connected. To use the interface the bus must be
connected to a BRAM." "" "mdt_error"
      } else {
        set busif  [xget_value $mhsinst "bus_interface" "BRAM_PORT"]
        if {[string length $busif] == 0} {
--- 60,67 ----
      set instname   [xget_value $mhsinst "parameter" "INSTANCE"]

      if {[bus_is_connected $mhsinst "BRAM_PORT"] == 0} {
! #      error "The BRAM_PORT interface is not correctly connected. To use the interface the bus must be
connected to a BRAM." "" "mdt_error"
!               puts  "WARNING: The BRAM_PORT interface is not correctly connected. To use the interface
the bus must be connected to a BRAM."
      } else {
        set busif  [xget_value $mhsinst "bus_interface" "BRAM_PORT"]
        if {[string length $busif] == 0} {
```

10 Tartalomjegyzék