Beágyazott és Ambiens Rendszerek Laboratórium

BMEVIMIA350

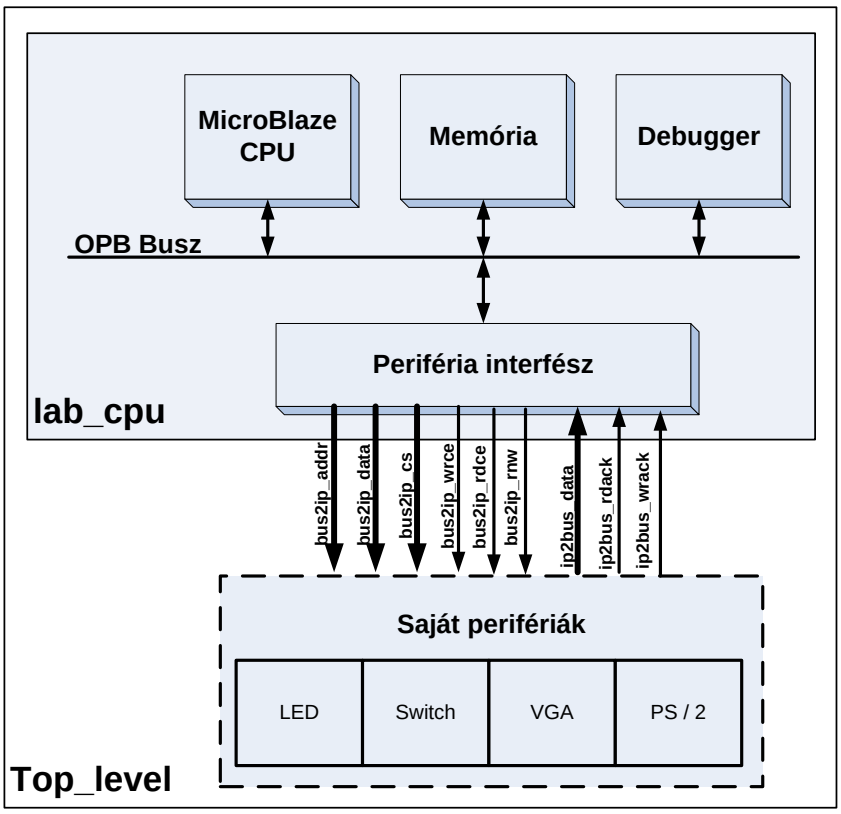
Mérési feladatok az 4., 5. és 6. mérési alkalomhoz

A mérés tárgya: FPGA áramkörön megvalósított mikroprocesszoros mintarendszer fejlesztése, a szoftver alkalmazás használatának megismerése

A 4 - 6. laboratóriumi mérés során egy FPGA eszközön megvalósított mikroprocesszoros rendszer hardver elemeinek fejlesztésére és az elkészített rendszerben a szoftver alkalmazás kialakítására kerül sor. A mikroprocesszoros rendszer egy *egyszerű alfanumerikus terminál* alkalmazás, amely alapvetően a perifériaillesztés megoldásának lehetőségeit mutatja be.

A kiépített rendszer a Xilinx beágyazott rendszer fejlesztési környezetének technológiáján alapul. A mikroprocesszor a 32 bites MicroBlaze. Ez a processzor az FPGA logikai elemeiből épül fel, ezért lágyprocesszornak hívjuk. (Vannak olyan processzorok is, amelyek közvetlenül az FPGA-t hordozó szilíciumba ültetnek, ezeket kemény magos processzornak nevezzük, pl. a Xilinx Virtex sorozat egyes elemeiben az IBM PowerPC405 processzor).

A MicroBlaze processzor mellett kiépített rendszer blokkvázlata az alábbi ábrán látható.



A mikroprocesszoros rendszer

A rendszer komponensek a 32 bites **PLB buszra** (Peripheral Local Bus, áramkörön belüli periféria busz) kapcsolódnak. Ez egy 32 bites adatúttal és 32 bites címzéssel rendelkező, szinkron működésű rendszerbusz.

A rendszer fő komponense a **MicroBlaze** processzor, melynek alkalmazói programja az áramkörön belüli RAM memóriából fut. Ezt a **RAM memóriát** használjuk program és adatmemóriaként is. A blokkvázlat harmadik, **Debugger** nevű eleme a mikroprocesszoros rendszer programfejlesztéséhez, a végrehajtás nyomkövetéséhez és hibakereséséhez ad támogatást. A mérés szempontjából a rendszer további fontos eleme a **periféria interfész**, amelyhez **saját perifériák** csatlakoznak. (Összesen négy darab.)

Periféria illesztés a mikroprocesszoros rendszerhez

Az *PLB buszra* történő perifériaillesztés komplikált valamint igencsak veszélyes, hiszen egy nem megfelelően implementált periféria a busz és egyben a *teljes rendszer lefagyásához* vezethetne. Az egyszerűbb és biztonságos fejlesztés céljából hoztuk létre a periféria interfész modult (amely IPIF-nek nevezett szabványos felületet biztosít.) Feladatai:

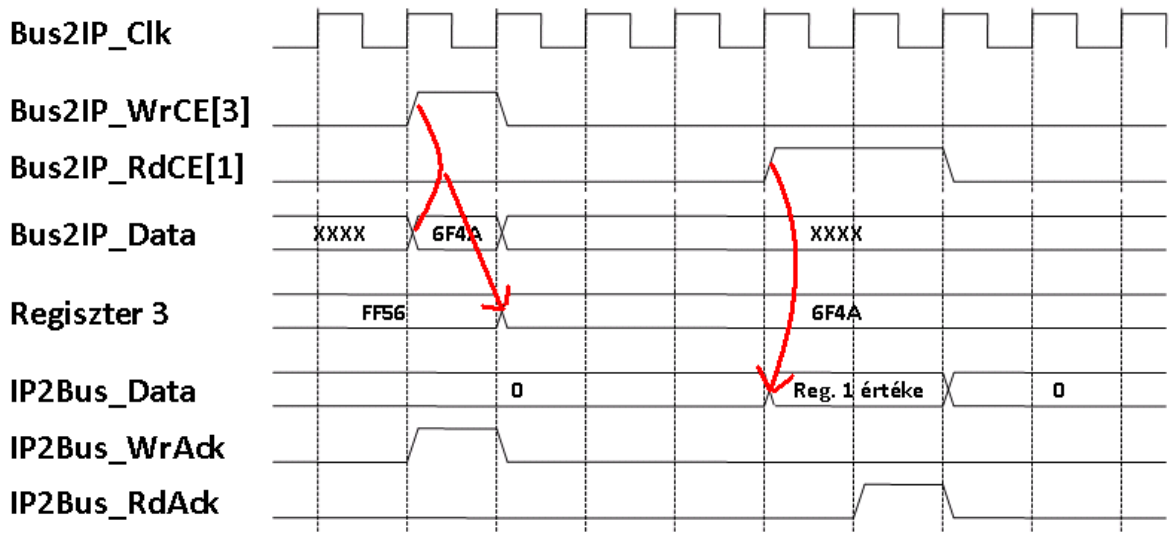
- buszciklusok előírás szerinti figyelése, dekódolása
- a blokkszintű kiválasztó jelek dekódolása (több periféria blokk illesztése esetén)
- az adatforgalom irányának jelzése
- buszciklus sikeres lezárása.

A periféria interfész a következő egyszerűsített *csatlakozási felületet* biztosítja:

- **clk:** A rendszer működtető jele, amely **25 MHz** frekvenciájú. Ez az órajel a fő ütemező jele a MicroBlaze mikroprocesszoros rendszernek, vagyis ez lesz az utasításvégrehajtási sebesség is. Ugyanez az órajel az OPB busz órajele is, valamint a perifériák is erről járnak. A külső 16 MHz-es órajelből egy DCM (Digital Clock Managemnet - órajel előállító modul) segítségével állítjuk elő a 25 MHz-es jelet. Minden adatváltozás a rendszerben (az aszinkron alaphelyzetbeállítást kivéve) az **órajel felfutó élére** történik. Az adatok mintavételezésére is a felfutó él használható.
- **rst:** A rendszer alaphelyzetbeállító jele. Aszinkron aktív magas rendszer reset.
- **bus2ip_addr[31:0]:** A rendszer teljes 32 bites címbusza, minden érvényes cím egy bájtot címez meg. A 4 GB címtartomány tartalmazza a processzor promgrammemória területét, az adatmemóriaterületet, a debugger modul címterületét és az IPIF_IF periféria illesztő által biztosított periféria címterületeket. A buszciklusok ideje alatt a címvonalak értéke stabil, a periféria regiszterek, saját memória területek címzésére stabilan használhatók. Teljes, 32 bites címbusz, byte-os busz címzési lehetőséggel, ami a memória használatnál szükséges. **Fontos!** A perifériainterfész csak 32 bites, szavas hozzáférést biztosít, ezért a perifériák címzésénél az alsó 2 címvonal értéke mindig 0 (*bus2ip_addr[1:0]=2'b00.*)

- **bus2ip_cs[3:0]:** Az periféria interfész **négy saját periféria** illesztését teszi lehetővé. Az aktív magas kiválasztójelek a teljes ciklus alatt stabil állapotúak. Minden perifériának **saját** kiválasztó jele van, tehát a **bus2ip_cs[0:3]** jelekből maximum egy magas értékű.
- **bus2ip_wrce[3:0] / bus2ip_rdce[3:0]:** Aktív magas írás és olvasás engedélyező jelek. . Minden perifériának **saját** kiválasztó jele van, amely jelzi a perifériára történő írást és az onnan történő olvasást.
- **ip2bus_rdack/ ip2bus_wrack:** A buszciklus végrehajtását nyugtázó jel. A PLB buszon minden megkezdett buszátvitel lezárásához a megcímzett egységből egy *aktív magas értékű* buszciklus lezáró jel szükséges, ez biztosítja a szinkronizációt a master és a slave egység között. A lezáró jel minimális hossza egy órajel ciklus. A rendszerben lévő periféria egységek egyszerű regiszter interfésszel rendelkeznek, ezért a szükséges lezáró jel a legrövidebb időn belül, azaz a kiválasztó jel megjelenése utáni órajelben visszaadható. Ennek megfelelően a méréshez kialakított mintarendszerben a buszciklusok mindegyik periféria esetén 2 órajelciklus hosszúak lesznek. Az olvasási ciklus-t a **ip2bus_rdack** jellel, míg az írási ciklust a **ip2bus_wrack** jellel kell nyugtázni.
- **bus2ip_data[31:0]:** 32 bites kimenő adatbusz
- **ip2bus_data[31:0]:** 32 bites bemenő adatbusz
- **bus2ip_rnw:** Buszciklus üzemmód kiválasztó jel. Érvényes a teljes buszciklus ideje alatt.(Jelen rendszerben nem használt)
    - „0”: irási buszciklus
    - „1”: olvasási buszciklus

Az alábbi ábrán egy 3. regiszterbe történő írást, majd pedig egy 1. regiszterből történő olvasást láthatunk.



Minta 1: 8 bites LED vezérlése a perifériaillesztő **nulladik** (bus2ip_cs[0]) portjára kötve:

A kimeneti adatvonalak használata egyszerű, a regiszterek *írás engedélyezését*

- a perifériaillesztő megfelelő írás kiválasztó (`bus2ip_wrce[xxx]==1`)
- és a címvonalak (jelen esetben ez nincs)

ÉS kapcsolatával vezéreljük.

```
...

reg [7:0] led_reg;                                  //Egy 8 bites regiszter létrehozása
always @ (posedge clk)
if (rst==1)
   led_reg <= 0;                                    //Reset esetén a LED-ek értéke 0

else if (bus2ip_wrce[0])                            //Ha ez a peréfiéria és írás
   led_reg <= bus2ip_data[7:0];                     //Mentsük el az adatbusz tartalmát a
                                                    //regiszterbe

assign xleds = led_reg;                             //A LED-ek kimenetre kötése

always @ (posedge clk)
   ip2bus_wrack <=bus2ip_wrce[0] & ~ip2bus_wrack;  //Generálunk egy ACK pulzust
```

Megjegyzés: A mintában nem használtuk fel, a *bus2ip_addr* jelet. Ez azt eredményezni, hogy amennyiben a *0.-ik* periféria bármely címét írjuk, a *led_reg* regiszter felülíródik.

Minta 2: 8 bites kapcsoló állásának beolvasása, amely a perifériaillesztő **legfelső** (bus2ip_cs[3]) portjára van kötve.

A bemeneti busz kezelése több periféria használata esetén bonyolultabb, ebben az esetben a több forrásból érkező buszjeleket csak egy bemeneti buszmultiplexerrel használhatjuk.

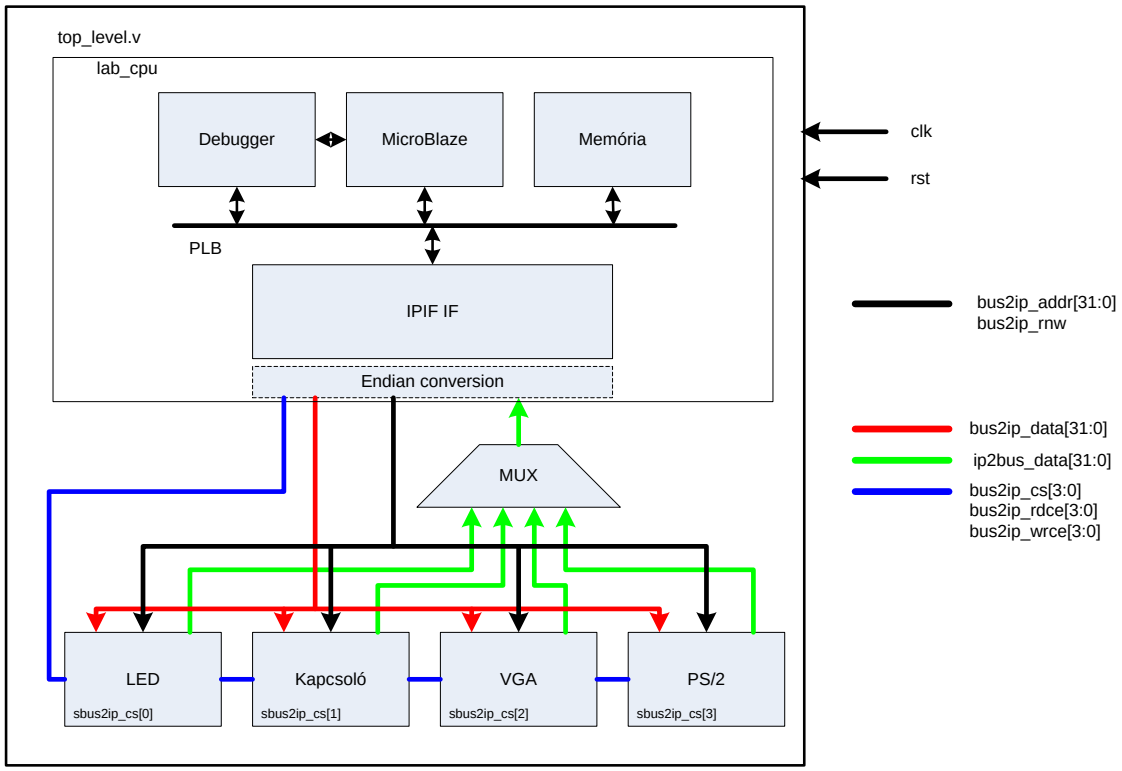
```
...

assign ip2bus_data[31:0] = (bus2ip_rdce[3]) ? { 24'b0 ,switch} : 32'b0;

always @ (posedge clk)
   ip2bus_rdack <=bus2ip_rdce[3] & ~ip2bus_rdack;          //Generálunk egy ACK pulzust

...
```

A megvalósítandó rendszer

A megvalósítandó rendszer (természetesen) egy számológép. A számológép bemenetét jelen esetben egy PS/2 kompatibilis billentyűzet adja, kimenete pedig egy VGA monitor. A számításokat az FPGA-ban lévő MicroBlaze végzi. Egyszerű teszt perifériaként 8 kapcsolót valamint 8 LED-et is illesztünk a rendszerhez, amelyek a fejlesztés során debuggolásra használhatók. A perifériákat az alábbi perifériaillesztő portra kötjük:

- **bus2ip_cs[0]: 8 LED**
- **bus2ip_cs[1]: 8 DIP kapcsoló**
- **bus2ip_cs[2]: VGA interfész**
- **bus2ip_cs[3]: PS/2 interfész**

A megvalósítandó rendszer blokkvázlata:



A négy megvalósítandó periféria modul a közös IPIF-en (Periféria Interfészen) keresztül kapcsolódik a rendszer buszhoz. Mint látható az *IPIF kimenőjeleit* a vezérlőjeleket és a 32 bites kimenő adatbuszt (bus2ip_data[31:0]) minden modul megkapja. A megvalósítandó perifériktól történő olvasás adatutjánál (ip2bus_data[31:0]) viszont egy multiplexret(MUX) kell kilakítanunk, mivel az IPIF interfésznek csak egyetlen adatbemenete van. **Fontos** megjegyezi, hogy amennyiben nincs aktív periféria olvasás ciklus a ip2bus_data[31:0] vonalra **logikai nullát** kell helyeznünk.

Abban az esetben, ha minden adatciklus megvalósul 1 órajel alatt (jelen esetben ezt tervezzük), az *ip2bus_rdack /ip2bus_wrack* könnyedén előállítható.
(Bármely perifériára történő olvasás/írás után 1 órajel hosszúságú ACK generálása)

```
always @ (posedge clk)
begin
   ip2bus_rdack <= |bus2ip_rdce & ~ip2bus_rdack;
   ip2bus_wrack <= |bus2ip_wrce & ~ip2bus_wrack;
end
```

Az adatbuszok szélessége

Az adatbuszok az IPIF_IF egységben a szokásos módon a kétirányú adatátvitel igényeinek megfelelően **szétválasztott formában** állnak rendelkezésre. A **teljes 32 bites adatbusz** adatvonalaiból a beépített perifériák csak a legkisebb helyiértékű bájtnak megfelelő **alsó 8 adatvonalat** használják, annak ellenére a rendszerben teljes szavas átviteleket hajtunk végre. Ennek megfelelően az írási buszciklusoknál a kimeneti adat felső 3 bájtját nem használjuk, eldobjuk, értéke tetszőleges lehet, általában nulla bitekkel töltjük fel. Az adat beolvasásnál hasonló módon az érvényes 8 bites adatot a felső 24 biten nullákkal kiegészítjük, így biztosítva a beolvasott adat érvényességét. A teljes 32 bites adatátvitel használata biztosítja, hogy a 8 bites adatátvitel mindig a legkisebb helyiértékű bájt vonalon történjen.

Címtartomány

A négy periféria egységet külön-külön címtartományban érhetjük el. Minden periféria **256 MBytos** címtartománnyal gazdálkodik. (bus2ip_addr [27:0]) A báziscímeket az alábbi táblázat mutatja:

- **bus2ip_cs[0]: 0x10000000 base, 256 Mbyte, LED**
- **bus2ip_cs[1]: 0x20000000 base, 256 Mbyte, SWITCH**
- **bus2ip_cs[2]: 0x30000000 base, 256 Mbyte, VGA FRAME BUFFER**
- **bus2ip_cs[3]: 0x40000000 base, 256 Mbyte, PS/2**

A regiszterek címdekódolása két módon képzelhető el:

- **Teljes címdekódolás:** amikor minden címvonalat bevonunk a dekódolásba. Ekkor csak a valódi címekre kiadott átviteleket hajtjuk végre, azt tekintjük érvényesnek.
- **Egyszerűsített címdekódolás:** Az egyszerűsítés érdekében a magasabb helyiértékű címvonalakat kihagyjuk. Ezáltal egyszerűbb dekóder megvalósítást nyerünk. Azonban figyelni kell az eredetileg kijelölt érvényes címen kívül további, a kihagyott vonalak által meghatározott modulus szerint ismétlődő címeken is lehetséges a kommunikáció.

Ne felejtsük el hogy a perifériainterfész csak 32 bites, szavas hozzáférést biztosít, ezért a perifériák címzésénél az alsó 2 címvonal értéke mindig *bus2ip_addr[1:0]=2'b00.*

Példa: Ha a nyolc LED-ből 4x2-őt szeretnénk külön-külön kezelni egyszerűsített címdekódolással, 2 címbit felhasználásával: (2 db led egy címen.)

- LED 0 – 0x10000000 alapcímen és 0x10000010 és 0x10000020 ….
- LED 1 – 0x10000004 alapcímen és 0x10000014 és 0x10000024 ….
- LED 2 – 0x10000008 alapcímen és 0x10000018 és 0x10000028 ….
- LED 3 – 0x1000000C alapcímen és 0x1000001C és 0x1000002C érhető el.

Akkor a címdekódert az alábbiak szerint lehet megvalósítani:

```
wire [3:0] led_cs;
assign led_cs[0] = bus2ip_wrce[0] & bus2ip_addr[3:2] == 2'b00
assign led_cs[1] = bus2ip_wrce[0] & bus2ip_addr[3:2] == 2'b01
assign led_cs[2] = bus2ip_wrce[0] & bus2ip_addr[3:2] == 2'b10
assign led_cs[3] = bus2ip_wrce[0] & bus2ip_addr[3:2] == 2'b11
```

vagy kicsit szebben...

```
wire [3:0] led_cs;
assign led_cs[3:0] = {4{bus2ip_wrce[0]}} & (4'b0001 << bus2ip_addr[3:2])
```

A megvalósított periféria egységek

A mérendő mikroprocesszoros rendszerben 4 periféria egységet építünk be. Ezek a fejlesztőkártya lehetőségeihez kapcsolódva a következő funkciókat valósítják meg:

LED-ek - bus2ip_cs[0]

Egyszerű 8 bites általános célú kimeneti egység a LED-ek vezérlésére. A periféria egy 8 bites regiszterből áll, melyet a bus2ip_cs[0] aktív értéke választ ki. A regiszter a rendszer reset hatására törlődik.

DIP Kapcsolók - bus2ip_cs[1]

Egyszerű 8 bites általános célú bemeneti egység a kapcsolók értékének beolvasására.

PS/2 bemeneti interfész - bus2ip_cs[3]

A PC billentyűzet egy összetett bemeneti interfész, ami egy egyedi soros kétirányú adatátviteli protokollal rendelkezik.

Az előkészített periféria a mintarendszerben *két 8 bites bemeneti olvasható regiszter címmel rendelkezik,* amelyet az alábbiak szerint illesztettünk:

- Ezek egyike **egy 16 bájtos FIFO**, amelybe a billentyűzet által küldött adatok érkeznek. A FIFO mérete elegendő tartalékot jelent a szoftveres kiolvasás sebessége esetén is az adatvesztés elkerülésére. Az elkészített periféria a FIFO-ba ASCII kódokat és a felengedés kódot tölti csak be.

  A FIFO olvasása különbözik a regiszterek olvasásától, mivel ebben az esetben gondoskodni kell az egyedi olvasási parancsok kiadásáról, ami a busz olvasás jel törlése után kiadott egy órajel pulzusnyi paranccsal oldható meg.

```
reg [1:0] ps2_fifo_rd_dl;
always @ (posedge clk)
begin
   ps2_fifo_rd_dl[0] <= bus2ip_rdce[3] & bus2ip_addr[2];
   ps2_fifo_rd_dl[1] <= ps2_fifo_rd_dl[0];
end
assign ps2_fifo_rd = (ps2_fifo_rd_dl==2'b01);
```

- A FIFO állapotáról, a perifériához tartozó **státusz regiszter** ad információt. A státuszregiszter bitjei a FIFO tele és a FIFO üres állapotáról tájékoztatnak.

A regiszter kiosztás: (BASE = A periféria báziscím. Jelen esetben: 0x40000000)

	Cím	**Bit7**	**Bit6**	**Bit5**	**Bit4**	**Bit3**	**Bit2**	**Bit1**	**Bit0**
Státusz	BASE+0x0	-	-	-	-	-	-	FULL	EMPTY
FIFO	BASE+0x4	D7	D6	D5	D4	D3	D2	D1	D0

A megvalósítandó periféria : VGA képernyővezérlő

A képernyővezérlő egy kis felbontású grafikus kijelző interfészt valósít meg. A vezérlést a LOGSYS Spartan-3E FPGA kártya bővítő csatlakozójához illesztett *VGA kimenet* biztosítja.

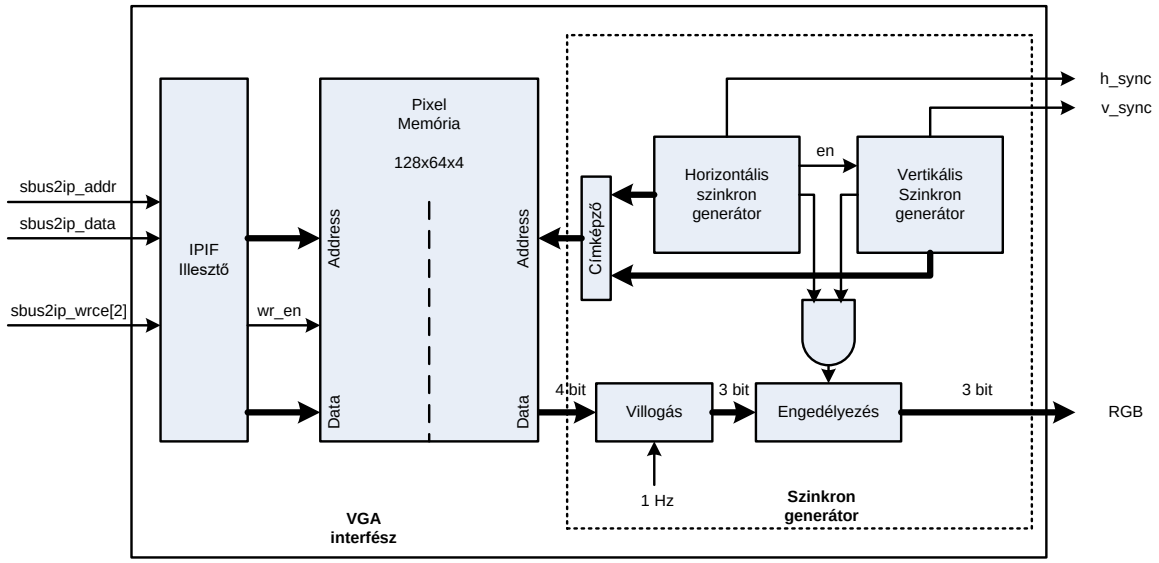
A megvalósítandó periféria összesen ***7 különböző*** szín megjelenítésére alkalmas, a kikapcsolt (fekete) mellett. A 7+1=8 szín információ 3 biten tárolható pixelenként, a szokásos RGB szerint.

Opcionális feladat. A megjelenítésnél opcionálisan **villogást** is választhatunk, amelyet egy **negyedik bittel** vezérlünk.

A VGA vezérlő felépítése

A VGA vezérlő 3 fő blokkból épülhet fel.

- **IPIF illesztő:** Feladata az IPIF buszra történő illesztés, legfőképpen címgenerálás és beírójel (wr_en) előállítása.
- **Pixel memória (frame_buffer):**Feladata a megjelenítendő pixelinformáció tárolása. (*Ez rendelkezésre áll.*)
- **Szinkron generátor:** Feladata aVGA szabványnak megfelelő időzítés előállítása.



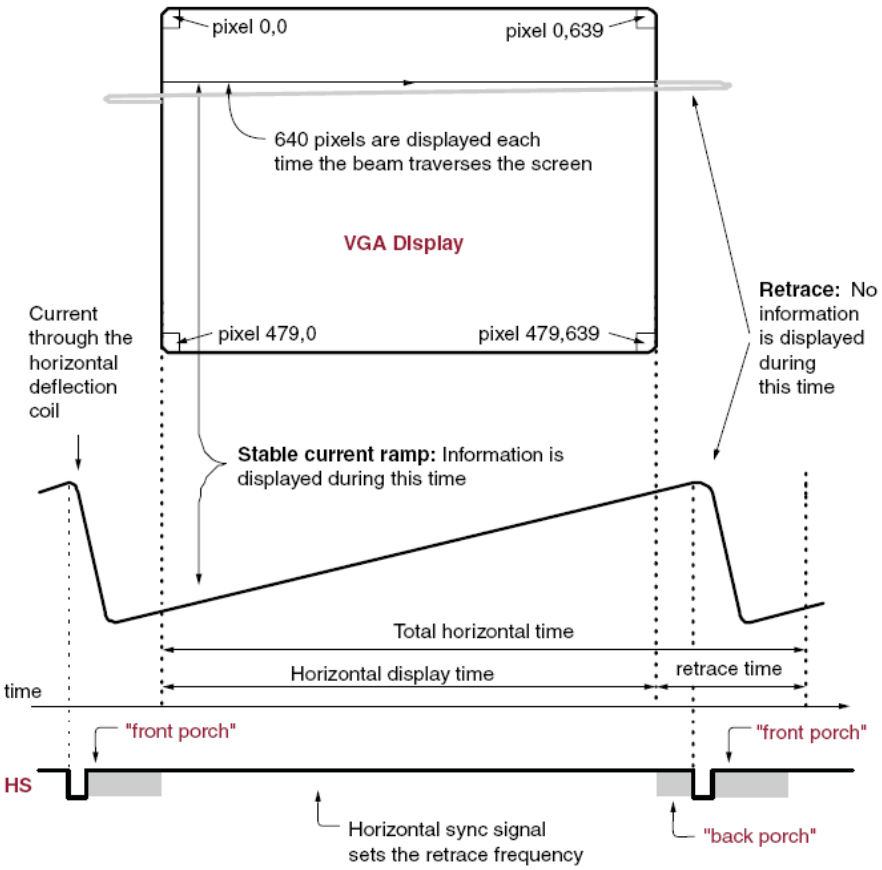
Szinkron generátor

A horizontális és vertikális szinkronjelek időzítéseinek beállításával tetszőleges felbontású és képfrissítésű megjelenítést beállíthatunk. A monitorok ezen két szinkronjel frekvenciája alapján azonosítják az általunk választott megjelenítési szabványt. A mérés során a szabványos felbontások közül a hagyományos **640x480 pixeles, 60Hz** frissítésű képernyő időzítést használjuk, **25MHz pixel órajel** használata mellett. A választott időzítéshez azonban nem használjuk a teljes felbontást, mert a teljes felbontáshoz szükséges képtartalom memória nem áll rendelkezésünkre. Emiatt a sor és oszlop méretét is nyolcadára csökkentettük, így habár a 640x480 felbontás időzítését használjuk, a monitort 8x8 pixeles egységekben vezéreljük. Tehát a 64 összetartozó pixel mindig azonos színű,

valamit kijátszásukkor ugyanazt a memóriát címezzük meg. Ennek megfelelően a választott képernyő kijelzés az eredeti 640x480 pixel helyett csak **80x60** felbontású.

Első lépésben – az egy sorban lévő pixeleket számoló - **pixelszámláló** jellegzetes értékeivel a **soridőzítést** és a **sorszinkron (horizontális szinkron)** jelet kell megtervezni. A teljes sorok elteltét számláló **sorszámláló** jellegzetes értékei alapján megtervezhető a a szükséges **képjel időzítés** és a **képszinkronjel (vertikális szinkron)** előállítása. A sorszámláló és a képszámláló állapota alapján az **aktív képtartalom** kijelzésének ideje kijelölhető. (A blokk diagramon „Engedélyezésnek" nevezett modulra azért van szükség, mert az inaktív képtartomány alatt csak **fekete szín** játszható ki.)

A horizontális időzítés magyarázatát az alábbi ábra mutatja:



Egy soridő (az ábrán: Total horizontal time) az alábbi részekből tevődik össze:

1. **Aktív tartomány** (Horizontal display time): Amikor is a monitor megjeleníti az RGB vonalon érkező adatokat
2. **Inaktív tartomány** (Retrace time): Ez idő alatt, a katódsugár kioltódik, és visszafut a következő sor elejére. Ezen idő 3 további részből tevődik össze:
    a. **„Back porch":** Az utolsó pixel után, szinkron jel előtti idő
    b. **Szinkron jel :**
    c. **„Front porch":** A szinkronjel utáni, nulladik pixel előtti idő

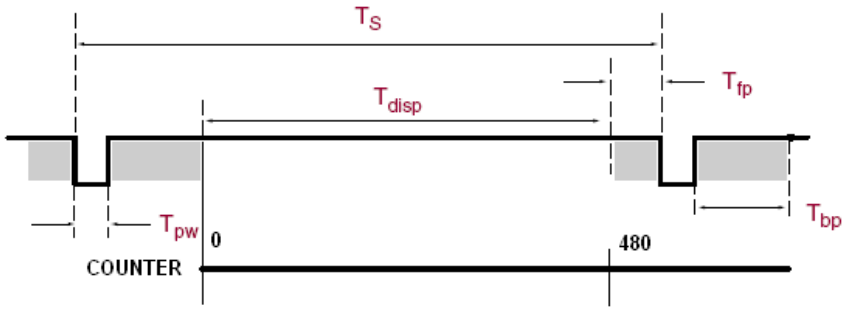
A vertikális szinkronjel felépítése teljesen hasonlít a horizontális szinkronhoz, de az nem pixeleket, hanem teljes sorokat számol.

Megjegyzés: Mindkét szinkronjel kis kitöltési tényezőjű, negatív impulzusú jel. (Sok ideig magas)

Az egyes horizontális időtartamokat az alábbi táblázat foglalja össze. Az időtartamokat célszerűen egy számláló számolja.

Megjegyzés: A képernyő időzítés tervezését érdemes a bal felső sarok (0,0) indexű pixelével kezdeni.

Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 µs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 µs	640
T_{PW}	Pulse width	64 µs	1,600	2	3.84 µs	96
T_{FP}	Front porch	320 µs	8,000	10	640 ns	16
T_{BP}	Back porch	928 µs	23,200	29	1.92 µs	48



Frame buffer

A megadott felbontással a **képernyő memória** minimális mérete 80x60x4= 9600 bit. A könnyebb címezhetőség érdekében a képernyő memória szervezésénél a minimális 80x60x4 helyett 128x64x4 méretben valósítjuk meg, ami a bináris címzéshez jobban illeszkedik. Ezáltal nem csak a processzor által generált címzés lesz nagyon egyszerű, hanem a kijelzés megvalósításához a pixelcímzés is közvetlenül, az időzítést meghatározó *pixel és sorszámlálók bitjeiből generált címzéssel* megvalósítható. A képernyő memóriát megvalósító *frame_buffer.v* file-t rendelkezésre bocsátjuk. (Forráskódja a mellékletben megtalálható.)

IPIF Illesztő

Feladata a korábban ismertetésre került IPIF interfész illesztése a képernyő memóriához. A címvonalakat úgy érdemes dekódolni, hogy az alábbi címtérképet kapjuk: (BASE = A periféria báziscím. Jelen esetben: 0x30000000)

- BASE +0x0000 1. sor - 1. „nagypixel”
- BASE +0x0140 1. sor - 80. „nagypixel” (utolsó látott pixel)

- BASE +0x01FC 1. sor - utolsó (128.) „nagypixel”
- BASE + 0x0200 2. sor - 1. „nagypixel”
- BASE + 0x77FC utolsó látható „nagypixel”
- BASE + 0x7FFC utolsó pixel.

A perifériák tesztelése

A megvalósított perifériákat két módon tesztelhetjük:

1. Az **XMD** (Xilinx Microprocessor Debugging Engine) segítségével csatlakozhatunk a rendszerben kialakított Debugger egységhez. A debugger egyszerűbb parancsok végrehajtására képes pl: periféria cím írása/olvasása, program letöltése, futtatás, reset etc.
2. A **Xilinx EDK** (Embedded Development Kit) beágyazott processzoros fejlesztő környezet segítségével C nyelven írhatunk programot a MicroBlaze processzorra, amely kezeli az egyes perifériákat. *Az EDK részletesebb bemutatása a 4. mérési alkalommal történik.*

A leggyakrabban használt XMD parancsok:

help	Segítség kérése
dow file.elf	Programkód letöltése
run	Letöltött programkód indítása
stop	MicroBlaze megállítása
rst	MicroBlaze reset
mrd address [length]	Adott címről történő olvasás pl. mrd 0x20000000[1]
mrd address data	Címre történő írás pl: mwr 0x10000000 0xAA [2]

[1] Kapcsolók állapotának beolvasása
[2] Minden második LED kigyújtása

A 3. mérés során megvalósítandó feladatok

1. Feladat: Tervezze meg a 640x480 pixel felbontású, 25 MHz pixel órajel frekvenciájú VGA kijelző vezérlő időzítő áramköreit. A 640x480 pixel felbontású VGA üzemmód időzítési követelményeinek megfelelően állítsa elő a VGA monitor horizontális és vertikális szinkronizáló jeleit.

A pixel vagy a sorszámláló bitjeit felhasználva készítsen **VGA mintagenerátort,** amely színes oszlopokat vagy színes sorokat jelenít meg.

2. feladat: Tervezze meg az 1 feladatban kifejlesztett mintagenerátor felhasználásával a képernyő grafikus memóriájának tartalmát megjeleníteni képes VGA vezérlőt. A grafikus memória tartalma egy csökkentett felbontású, az aktív képet tekintve 80x60x4 bites címezhető képkockát tartalmazó memória. A memória fizikai szervezése 128x64x4, ami (a sok nem használt memória hely elvesztése árán) lehetővé teszi a pixel és sorszámlálók bitjeinek egyszerű felhasználását a képkockák címzésekor.

- Válassza ki a pixelszámláló bitjeiből azokat, amelyekkel az eredeti 640 pixelszám ideje alatt 0-79 számú egy soron belüli nagy képkockák megcímezhetők.
- Hasonló módon válassza ki a sorszámláló bitjei közül azokat, amelyek az eredetileg 480 aktív sort tartalmazó képeltérítés ideje alatt a 0-59 számú nagyméretű sorok címzését biztosítják.

Fontos ! Mivel a memória szervezése egy 128x64x4 méretű területnek megfelelő, a kiválasztott bitekből a szükséges címinformáció könnyedén előállítható.

Az elkészült VGA vezérlőt a mellékelt egyéb perifériával együtt ágyazza be a referencia tervbe. A VGA vezérlőt kezelje *külön modulként* ! Az elkészített rendszer teszteléséhez a MicroBlaze processzor memóriája egy egyszerű programot tartalmaz, melynek funkciója a következő: (*A forráskód a mellékletben megtalálható*)

Szoftver:

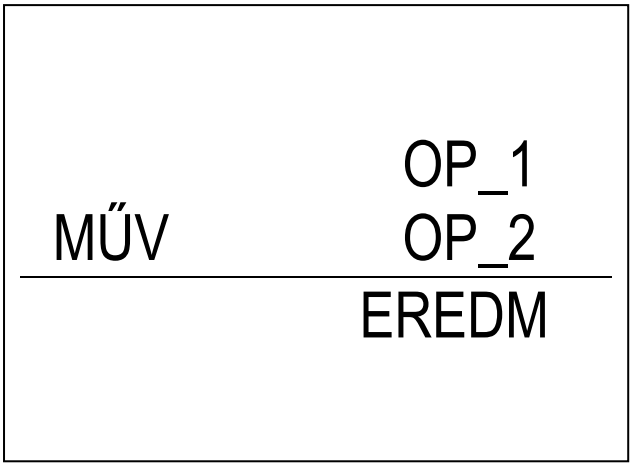
- LED
  - kb. másodperces bináris számláló, ha SW[0]==0
  - utoljára fogadott ASCII kód, ha SW[0]==1
- VGA
  - 128x64 szavas, 4 bit/szó. Egy sor 128 szó. Teljesen kék, a felső két sor kb. másodpercenként villog (fekete/fehér).
- PS/2: 2 olvasható regiszter
  - BASE+0x0: státusz
  - BASE+0x4: adat

A 4. mérés során megvalósítandó feladatok

A 3. mérés a mikroprocesszoros mintarendszer hardver komponenseinek fejlesztését célozta. Ennek alapján előállt a teljes mintarendszer, ami tartalmazza mindazokat a komponenseket, amelyek segítségével a mikroprocesszoros rendszer tetszőleges terminál jellegű alkalmazás megvalósítására képes. A választott alkalmazás az első mérésen megvalósított kalkulátor jellegű alkalmazás, csak egy magasabb szintű ember-gép interfész alkalmazásával.

A kitűzött feladat specifikációja a következő: A rendszerhez illesztett PS/2 szabványú billentyűzet numerikus billentyűit (0..9) és a műveleti jeleket (+, -, ×, ÷) felhasználva készítsünk egy kalkulátor programot, amely a mintarendszer 80x60 képkocka felbontású VGA képernyőjén karakteres kijelzéssel jelzi a műveleteket.

A várt kijelzési kép:



A kijelzés karakterei a választott képkocka méretnek megfelelő felbontással generált felbontású karakterképek 8x8-as felbontású bitképei. A karakter táblázat a mérés során rendelkezésre áll.

A VGA kijelzés képernyő memóriája a választott memória kiosztás szerint 128x64x4 bit felbontású, azaz a valóban kijelzett 80x60 képkocka helyett a sor a memóriában 128 képkocka hosszú és összesen 64 sor van. Ebből természetesen a megjelenítő hardver, amelynek címgenerátora az előző mérésen lett megtervezve csak az sorok első 80 képkockáját és az első 60 sort jeleníti meg. A 80x60 képkocka a rendelkezésre álló 8x8 felbontású karakter bitképekkel a teljes képernyőn soronként 10 karaktert és összesen 7 sort tartalmazhat. A képkockák színét a 3+1 színbit együttesen határozza meg. A feladatban ezek tetszőlegesen megválaszthatók.

A billentyűzet kezelése nagyobb gondosságot igényel. A billentyűzet, mint bemeneti periféria 2 db 8 bites regiszter címen érhető el. Ebből az első a státuszregiszter, a második a karakterkódokat tartalmazó 16 mélységű FIFO. Nem megfelelő kezelés esetén a FIFO megtelhet. A FIFO állapotáról a státuszregiszter tájékoztat, a FULL és EMPTY bitekkel. A karakter kiolvasások előtt a státuszregiszter olvasása lehetővé teszi a PS/2 periféria biztonságos kezelését.

A PS/2 billentyűzet a szabvány szerint a billentyű lenyomáskor elküldi a leolvasási kódot, folyamatosan nyomva tartva ezt ismételgeti kb. 100ms időnként, majd felengedéskor a felengedési kódot és még egyszer a leolvasási kódot. Ezeket a kódokat a PS/2 hardver a billentyű ASCII kódjává kódolja, a felengedést a 0xF0 + ASCII kóddal jelezve.

A szoftver javasolt felépítése:

A képernyőkezelés alapfüggvénye a képkocka írásához kapcsolódik. A képkocka címzése a választott képernyő memóriaterület kiosztás miatt egyszerű. A 80x60 képkocka felbontású megjeleníthető terület címzése a kívánt sor indexének (S, 0-59) és a soron belüli pozíciójának (P, 0-79) felhasználásával a 128x64 db 32 bites szót elfoglaló memóriaterületen belül a (128*S+P)*4 összefüggéssel számolható.

A képkocka színe 8 értékre állítható, ebből az [0,0,0] érték a kioltott, fekete szín, a többi értéket a hardverben választott [R,G,B] jelek bekötési sorrendje szabja meg, a [1,1,1] természetesen a mindhárom szín bekapcsolásával előálló fehér. A 4 bit, a villogást befolyásolja.

A képkocka írás alapfüggvénye felhasználható a teljes képernyő törlésére, adott terület törlésére, illetve adott terület új tartalommal történő felülírására. Ennek megfelelően egy karakter kiírása az adott 8x8 képkocka méretű terület törlésével, majd a karakter bitképének kiírásával történhet. A karakterek bittérképét a függelék tartalmazza. Javasolt az alábbi függvények definiálása:

void SetPixel (uint32 x, uint32 y, uint32 color)	Egy pixel bekapcsolása (x=<80 & y=<60)
void ClearDisplay (uint32 color)	Teljes képernyő törlése egy adott színre
void PutChar (char c, uint32 x, uint32 y, uint32 color)	Egy adott pozícióra történő karakter rajzolás (x<= 10 & y<=7)
void PutNumber(int number, uint32 x, uint32 y, uint32 color)	Egy adott pozícióra történő szám kiírása. Ebben a függvényben szoftveres BIN -> ASCII átalakítás szükséges.

A billentyűzet beolvasás a PS/2 periféria státuszregiszterének ellenőrzésével kezdhető. A program indulásakor, ha a státuszregiszter szerint a FIFO nem üres, érdemes azt maximum 16 olvasással kiüríteni. Ezután minden esetben, ha a FIFO nem üres, akkor egy értékes billentyű lenyomás ASCII kódját, ill. a bevezetőben ismertetett minták egyikét tartalmazza. Javasolt az alábbi függvény definiálása:

int GetChar(void)	Ha van lenyomott billenyű a FIFO-ban, akkor azzal tér vissza egyébként -1 -el.

A teljes működéshez tartozik a paraméterek beolvasása, ennek megfelelően az operandusok és a műveleti kód felismerése. A lezáró parancs legyen az „Enter” billentyű, ha szükséges, akkor ez az operandus és a műveleti kód bevitele között is használható. A korábbi megoldáshoz hasonlóan elegendő a feladatot egy digit pontosságra megoldani, de a szoftveres megvalósítás lehetővé teszi akár a több számjegyes műveletvégzést is. Az osztás továbbra is csak egészosztás legyen.

A második operandus beolvasását követő ENTER kód után történhet az eredmény számítása, majd a kijelzéshez szükséges karakterek kiírása. A bemeneti számjegy tartomány túllépésekor tetszőleges, elfogadható viselkedés beépíthető, a nullával történő osztáskor hibajelzés kiadása szükséges.

Az eredmény kijelzése után a kalkulátor a képernyőtartalom megtartásával álljon alaphelyzetben, készen arra, hogy egy új használat adatait beírjuk.

Függelékek

A legfelső modul HDL kódja

```
`timescale 1ns / 1ps

module top_level(
   input        xclk,
   input        xrst,

   output [7:0] xleds,
   input  [7:0] xsw,

   output [5:0] xrgb,
   output       xhs,
   output       xvs,

   input        xps2_c,
   input        xps2_d
);

// Generating 25 MHz system clock
wire clk;
clk_gen clk_gen(
   .xclk(xclk),
   .clk(clk)
);


wire rst;
assign rst = xrst;

wire        bus2ip_clk;
wire        bus2ip_rst;
wire [31:0] bus2ip_addr;
wire        bus2ip_rnw;
reg         ip2bus_ack;
wire [31:0] bus2ip_data;
wire [3:0]  bus2ip_cs;
wire [31:0] ip2bus_data;
reg         ip2bus_rdack;
reg         ip2bus_wrack;
wire [ 3:0] bus2ip_rdce;
wire [ 3:0] bus2ip_wrce;

// Intsantiating CPU module
lab_cpu lab_cpu(
    .sys_clk(clk),
    .sys_rst(rst),
    .bus2ip_clk(),
    .bus2ip_reset(),
    .bus2ip_addr(bus2ip_addr),
    .bus2ip_cs(bus2ip_cs),
    .bus2ip_rnw(bus2ip_rnw),
    .bus2ip_data(bus2ip_data),
    .bus2ip_be(),
    .bus2ip_rdce(bus2ip_rdce),
    .bus2ip_wrce(bus2ip_wrce),
    .ip2bus_data(ip2bus_data),
    .ip2bus_rdack(ip2bus_rdack),
    .ip2bus_wrack(ip2bus_wrack),
    .ip2bus_error(1'b0)
);


// Acknowledge is generated one clock cycle after CS is set by the bus
// This allows easy register and BRAM reads
always @ (posedge clk)
begin
   ip2bus_rdack <= |bus2ip_rdce & ~ip2bus_rdack;
   ip2bus_wrack <= |bus2ip_wrce & ~ip2bus_wrack;
end

// LED output register is written when CS[0] is active
```

```
// the lowest 8 bits of the 32 bit data bus is used, occupies the full address range
reg [7:0] led_reg = 0;
always @ (posedge clk)
if (rst==1)
   led_reg <= 0;
else if (bus2ip_wrce[0])
   led_reg <= bus2ip_data[7:0];

assign xleds = led_reg;

// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// VGA
// Instantiating VGA core
// Mapped into CS[2] address range, write only


assign xrgb = { {2{rgb_int[2]}}, {2{rgb_int[1]}}, {2{rgb_int[0]}} };

// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// Instantiating PS/2 core
// Mapped into CS[3] address range, read only
wire [31:0] ps2_status, ps2_data;
ps2_if ps2_if(
   .clk(clk),
   .rst(rst),
   .ps2_c(xps2_c),
   .ps2_d(xps2_d),
   .fifo_rd(ps2_fifo_rd),
   .status(ps2_status),
   .data(ps2_data)
);
reg [1:0] ps2_fifo_rd_dl;
always @ (posedge clk)
begin
   ps2_fifo_rd_dl[0] <= bus2ip_rdce[3] & bus2ip_addr[2];
   ps2_fifo_rd_dl[1] <= ps2_fifo_rd_dl[0];
end
assign ps2_fifo_rd = (ps2_fifo_rd_dl==2'b01);

// READ multiplexer for the CPU bus
reg [31:0] bus_din_mux;
always @ ( * )
case ({bus2ip_rdce, bus2ip_addr[2]})
   5'b00010: bus_din_mux <= {24'h0, led_reg};
   5'b00011: bus_din_mux <= {24'h0, led_reg};
   5'b00100: bus_din_mux <= {24'h0, xsw};
   5'b00101: bus_din_mux <= {24'h0, xsw};
   // 6'b0100x: no readable VGA registers
   5'b10000: bus_din_mux <= ps2_status;
   5'b10001: bus_din_mux <= ps2_data;
   default : bus_din_mux <= 32'b0;
endcase
assign ip2bus_data = bus_din_mux;


endmodule
```

A frame-buffer HDL forráskódja

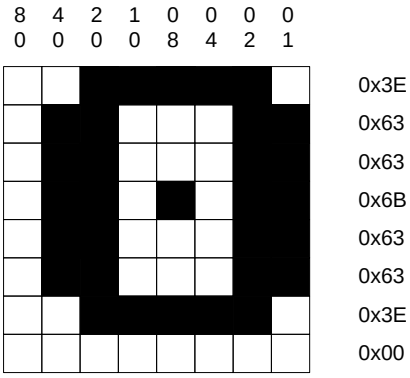
```
`timescale 1ns / 1ps
module frame_buffer(
   input        clk,
   input        rst,
   input        wr_en,
   input [12:0] wr_addr,
   input [ 3:0] wr_data,
   input [12:0] rd_addr,
   output [3:0] rd_data
);

// Creating 128x64 8-bit-word array
reg [3:0] dp_ram[8191:0];

// Implementing dual-port BlockRAM
// Both read and write ports are synchronous
// Read port has one clock cycle latency from valid adress
reg [3:0] rd_reg;
always @ (posedge clk)
begin
   if (wr_en)
      dp_ram[wr_addr] <= wr_data;

   rd_reg <= dp_ram[rd_addr];
end

assign rd_data = rd_reg;

endmodule
```

A harmadik mérésen alkalmazható teszt kód C forráskódja

```
#include "xparameters.h"
#include "stdio.h"
#include "xutil.h"

//====================================================
volatile int *ptr_led = 0x10000000;
volatile int *ptr_sw  = 0x20000000;
volatile int *ptr_vga = 0x30000000;
volatile int *ptr_ps2 = 0x40000000;

int main (void) {
   int i,k;
   int ascii;
   int pixel;
   int sw, ps2_status;

   k=0; ascii=0; pixel=0; sw=0; ps2_status=0;

   for (i=0;i<128*64;i++)
   {
      *(ptr_vga+i) = 0x1;
   }

   while(1)
   {

     ps2_status = *(ptr_ps2);
     while ((ps2_status & 0x1) != 1)
     {
         ascii = *(ptr_ps2+1);
         ps2_status = *(ptr_ps2);
     }

     sw = *(ptr_sw);
     if ((sw & 0x1) == 0)
         *(ptr_led) = k;
     else
         *(ptr_led) = ascii;

      pixel = (~pixel & 0x7);
      for (i=0;i<80;i++)
      {
         *(ptr_vga+i) = pixel;
         *(ptr_vga+128+i) = pixel;
      }

      for(i=0;i<=1000000;i++);

      if (k==255)
         k=0;
      else
         k=k+1;
      *(ptr_led) = k;

   }
   return 0;

}
```

Karakterek 8x8 méretű bittérképe

Minta a „0”-ás karakter értelmezése




```
const unsigned char FONT8x8[][8] = {
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00, // space                    ASCII 0x20
0x30,0x78,0x78,0x30,0x30,0x00,0x30,0x00, // !
0x6C,0x6C,0x6C,0x00,0x00,0x00,0x00,0x00, // "
0x6C,0x6C,0xFE,0x6C,0xFE,0x6C,0x6C,0x00, // #
0x18,0x3E,0x60,0x3C,0x06,0x7C,0x18,0x00, // $
0x00,0x63,0x66,0x0C,0x18,0x33,0x63,0x00, // %
0x1C,0x36,0x1C,0x3B,0x6E,0x66,0x3B,0x00, // &
0x30,0x30,0x60,0x00,0x00,0x00,0x00,0x00, // '
0x0C,0x18,0x30,0x30,0x30,0x18,0x0C,0x00, // (
0x30,0x18,0x0C,0x0C,0x0C,0x18,0x30,0x00, // )
0x00,0x66,0x3C,0xFF,0x3C,0x66,0x00,0x00, // *
0x00,0x30,0x30,0xFC,0x30,0x30,0x00,0x00, // +
0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00, // ,
0x00,0x00,0x00,0x7E,0x00,0x00,0x00,0x00, // -
0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00, // .
0x03,0x06,0x0C,0x18,0x30,0x60,0x40,0x00, // /

0x3E,0x63,0x63,0x6B,0x63,0x63,0x3E,0x00, // 0                        ASCII 0x30

0x18,0x38,0x58,0x18,0x18,0x18,0x7E,0x00, // 1
0x3C,0x66,0x06,0x1C,0x30,0x66,0x7E,0x00, // 2
0x3C,0x66,0x06,0x1C,0x06,0x66,0x3C,0x00, // 3
0x0E,0x1E,0x36,0x66,0x7F,0x06,0x0F,0x00, // 4
0x7E,0x60,0x7C,0x06,0x06,0x66,0x3C,0x00, // 5
0x1C,0x30,0x60,0x7C,0x66,0x66,0x3C,0x00, // 6
0x7E,0x66,0x06,0x0C,0x18,0x18,0x18,0x00, // 7
0x3C,0x66,0x66,0x3C,0x66,0x66,0x3C,0x00, // 8
0x3C,0x66,0x66,0x3E,0x06,0x0C,0x38,0x00, // 9
0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x00, // :
0x00,0x18,0x18,0x00,0x00,0x18,0x18,0x30, // ;
0x0C,0x18,0x30,0x60,0x30,0x18,0x0C,0x00, // <
0x00,0x00,0x7E,0x00,0x00,0x7E,0x00,0x00, // =
0x30,0x18,0x0C,0x06,0x0C,0x18,0x30,0x00, // >
0x3C,0x66,0x06,0x0C,0x18,0x00,0x18,0x00, // ?
};
```