SNAKE

Feladat felosztás:

    SW: Limbay Bence

    HW: Cseh Péter

Perifériák:

    LCD(27mmx38mm)

    Navigációs gomb(regiszterbe beleírja az utolsó gombnyomás értékét, nem generál interruptot, timer interruptnál kérdezzük le)

    LED: szint kijelzése

    7szeg: pontszám (0-99)

    Gombok: előző szint(BTN2), szint újrakezdése(BTN1), következő szint(BTN0), (regiszterbe beleírja az utolsó gombnyomás értékét, nem generál interruptot, a timer interruptnál kérdezzük le)

    Timer nehézségi szintek beállítása

    UART(debugolásra)

Buszrendszer:

    AXI4-Lite

Szabályok:

Pálya szélének megy a kígyó, akkor meghal.

Ha megeszi a bogyót, akkor 1-el nő.

Ha saját magába belemegy, akkor is meghal

Objektumok:

    Kígyó fej: 3x3 teli

    Kaja: 3x3 -> (0,1; 1,0; 1,2; 2,1) -> lásd ábra a kiadott lapon

    Keret -> úgy vágjuk le, hogy belül 3-al osztható legyen



**Budapesti Műszaki és Gazdaságtudományi Egyetem**

**Villamosmérnöki és Informatikai Kar**

**Programozható logikai eszközök alkalmazástechnikája mellékspecializáció** **(MIT)**

**Mikrorendszerek tervezése**

**Snake játék LOGSYS Spartan 6 FPGA kártáyra**

**Konzulens: Wacha Gábor**

**Készítették: Limbay Bence (E2JT1E)**

**Cseh Péter (DM5HMB)**

**2017.10.06.**

Tartalomjegyzék

[1. Játék 4](#_Toc498854829)

[1.1. Játékleírás, szabályok 4](#_Toc498854830)

[1.2. Megjelenítés, játéktér 4](#_Toc498854831)

[2. Hardver 6](#_Toc498854832)

[2.1. Specifikáció 6](#_Toc498854833)

[2.2. MicroBlaze 6](#_Toc498854834)

[2.3. Simple IO 6](#_Toc498854835)

[2.4. LCD 7](#_Toc498854836)

[2.4.1. LCD regiszter 7](#_Toc498854837)

[2.4.2. LCD blokkvázlata 8](#_Toc498854838)

[2.5. Tesztelés 12](#_Toc498854839)

[3. Szoftver 13](#_Toc498854840)

[3.1. Tervezés 13](#_Toc498854841)

[3.2. Implementáció 13](#_Toc498854842)

[3.3. Tesztelés 13](#_Toc498854843)

[Függelék - LCD periféria IP kódok: 14](#_Toc498854844)

[LCD – Top modul (user logic.v) 14](#_Toc498854845)

[LCD – Bus Interface (bus\_if.vhd) 15](#_Toc498854846)

[LCD – FIFO (fifo.vhd) 16](#_Toc498854847)

[LCD – SPI (spi.vhd) 17](#_Toc498854848)

[LCD – Top modul szimuláció (lcd\_sim.v) 19](#_Toc498854849)

[LCD – FIFO szimuláció (fifo\_sim.v) 20](#_Toc498854850)

[Irodalomjegyzék 20](#_Toc498854851)

# 1. Játék

## 1.1. Játékleírás, szabályok

A Snake játék célja, hogy egy irányított kígyóval meg kell enni a játéktéren megjelenő ételeket.

A kígyó 4 irányban haladhat a pályán: fölfelé, lefelé, balra és jobbra. Az aktuális haladási irányhoz képest a kígyó egy lépésben nem fordulhat az átellenes irányba, tehát például ha fölfelé megy a kígyó, akkor csak balra vagy jobbra fordulhat, lefelé nem.

A kígyó blokkokból áll. Haladáskor a blokk felveszi az előtte lévő blokk pozícióját, a fej új pozícióba kerül.

A játék a kígyó számára generál ételeket. Ha a kígyó áthalad az ételen (megeszi), akkor a kígyó mérete eggyel növekszik, és a játékos pontja is eggyel nő. Amikor a kígyó megeszi az ételt, a játék új ételt generál.

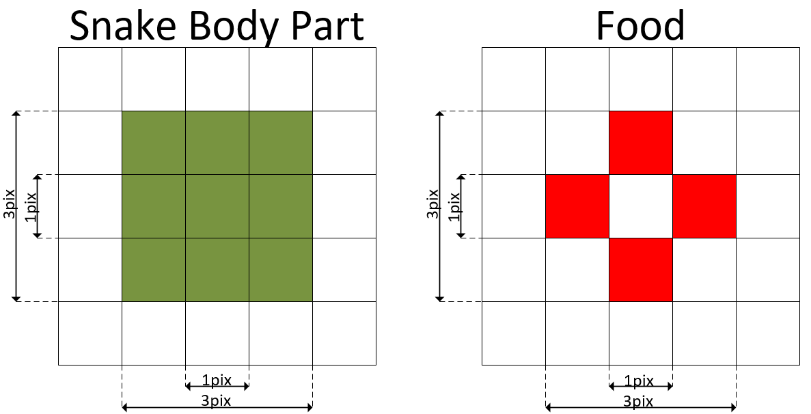
A játékban 8 szint van, az 1. szint a legkönnyebb, a 8. szint a legnehezebb. A nehézség a kígyó gyorsaságában mutatkozik meg, nehezebb pályákon a kígyó gyorsabban halad.

Ahhoz, hogy a játékos teljesítse a szintet, ahhoz el kell érnie 99 pontot az adott szinten.

Ha a kígyó belemegy a saját farkába, vagy ha neki megy a falnak, akkor meghal, a szintet újra kell kezdeni.

## 1.2. Megjelenítés, játéktér

A megjelenítés során 3x3-as pixel blokkokból építkezünk, mert ezzel jól szemléltethetőek a játéktérben megjelenő objektumok. Két féle 3x3-as objektum létezik: kígyó test és étel. Ezek az **1.2.a. ábrán** láthatók (természetesen az LCD-n nem ilyen színekben lesznek).



1.2.a. ábra: 3x3 objektumok

A játéktér kialakításához szükséges még egy keret is. A kerettel ellátott játéktér az **1.2.b. ábrán** látható.

Mivel egy objektum 3x3 pixel méretű, és a játék során csak objektumokat akarunk a játéktérbe rajzolni, ezért a játéktér fel van osztva 3x3 blokkokra.

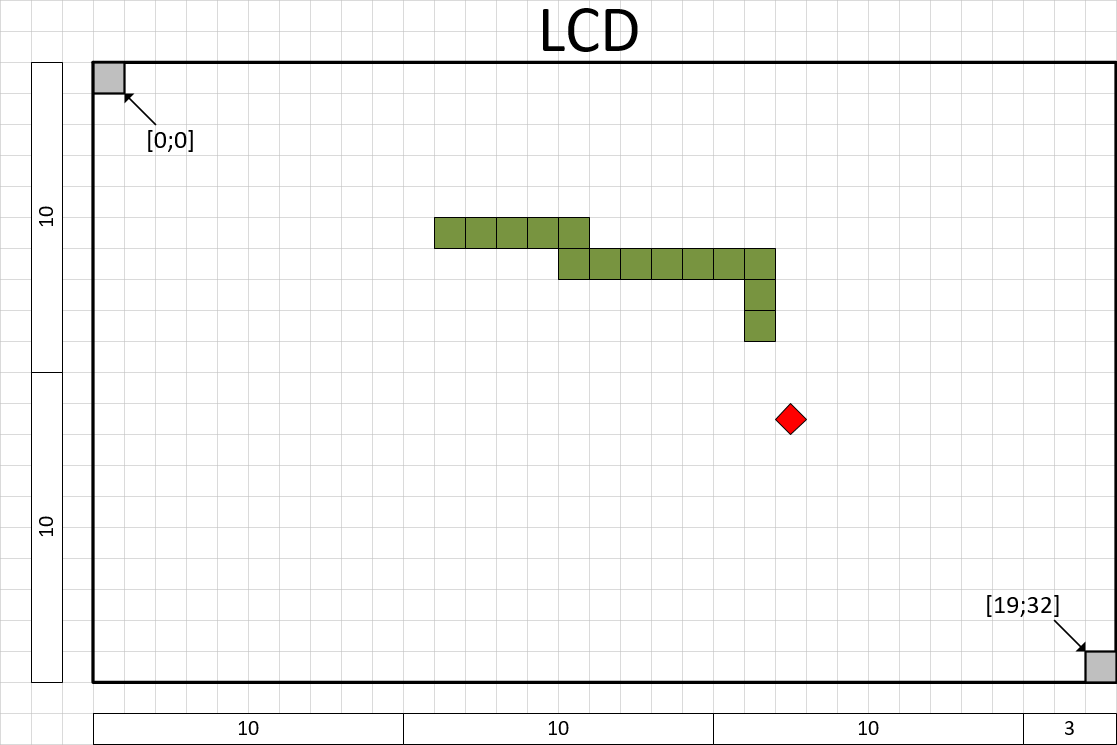
A teljes szélessége az LCD-nek 102 pixel. A keret 2 pixelt foglal el ebből, ezért ha a megmaradó 100 pixelt felbontjuk 3-as egységekre, akkor az 33db 3-as blokkot eredményez, és egy megmaradó pixelt, amit a játék során nem használunk.

(102 - 2) / 3 = 33blokk + 1 pixel

A teljes magassága az LCD-nek 64 pixel. A blokkok száma itt 20, a megmaradó pixelek száma: 2.

(64 – 2) / 3 = 20 blokk + 2 pixel

Az LCD tehát 33x20 blokkot tartalmaz, ennyi objektum lehet legfeljebb a játéktérben.



# 2. Hardver

## 2.1. Specifikáció

A Snake játékot a Logsys Spartan-6-os board-ján kell implementálni. A játék a 102x64 pixeles LCD kijelzőn jelenik meg. A játékos pontszámát a hétszegmenses kijelző mutatja, a pályaszintet a LED-ek. A kígyó irányítását a navigációs kapcsolóval lehet megtenni. A három gombbal (BTN0, BTN1, BTN2) lehet pályát váltani, illetve újraindítani a szintet.

## 2.2. MicroBlaze

Az FPGA-n lévő hardver lelke a MicroBlaze soft processzoros rendszer. A projekt kiinduló forrása a gyakorlatokon összerakott processzoros rendszer, amely komponensei AXI4 buszon keresztül kapcsolódnak a MicroBlaze processzorhoz. A rendszerben megtalálható „interrupt controller”, amely kezeli am megszakításokat. Megtalálható „timer”, az időzítési feladatok kezeléséhez, és egy UART modul is debugolási célok miatt.

## 2.3. Simple IO

Az egyszerű perifériák kezeléséhez a már meglévő „logsys\_axi\_sp6\_simpleio” IP-t használtuk fel, amely segítségével elérhetőek a gombok, a navigációs gomb, a LED-ek és a két hétszegmenses kijelző.

A szimple IO IP regiszterei a **2.3. ábrán** láthatók. A regiszterek 8 bitesek.

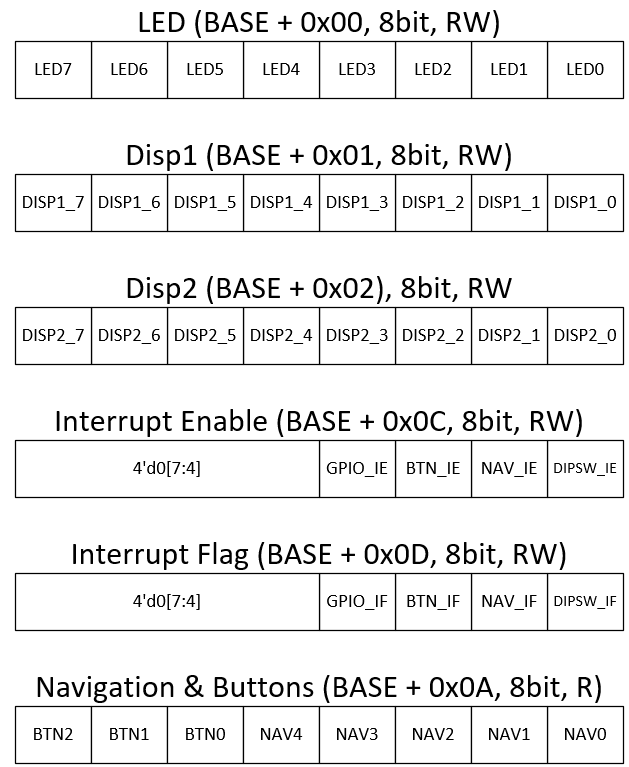
A LED-eket tartalmazó regiszter az IP-hez tartozó báziscímen van, írható és olvasható regiszter.

A „Disp1” és „Disp2” elnevezés a hétszegmenses kijelzőkre utal. Ezek a regiszterek szintén írható és olvasható regiszterek, a báziscímhez képest „0x01” és „0x02” offsettel rendelkeznek.

A báziscímhez képest „0x0C” offseten lévő regiszter az írható, olvasható Interrupt Enable regiszter, ahol engedélyezhetők a GPIO, gomb, navigációs gomb, és kapcsolók megszakításai. A projekt szempontjából csak a gombok és a navigációs gomb a lényeges.

A következő regiszter az Interrupt Flag regiszter, amely a megszakításkérést a perifériák felől. Ez a regiszter szintén írható és olvasható, a címe a báziscímhez képest „0x0D”-vel el van tolva.

Az utolsó lényeges regiszter a Simple IO IP-ban a navigációs gombhoz és a gombokhoz rendelt regiszter. Ez egy csak olvasható regiszter, ahol a felső három bit tartalmazza a gombok értékét, és az alsó 5 bit tartalmazza a navigációs gomb értékét. Ez a báziscím után van „0x0A”-val.



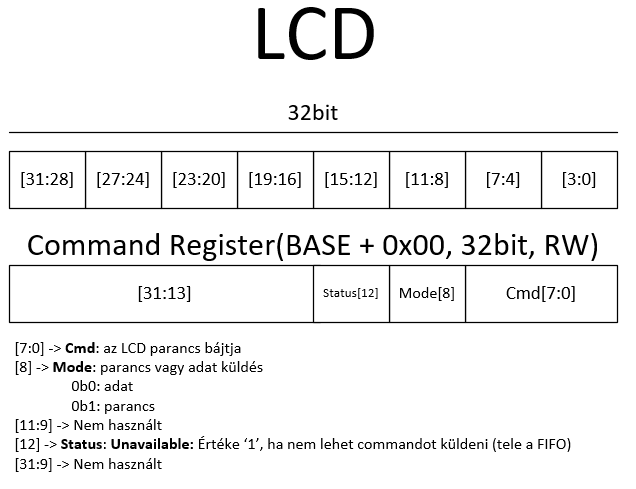
2.3. ábra: Simple IO IP regiszterei

## 2.4. LCD

Az LCD periféria kezelést egy a MicroBlaze processzorhoz elkészített IP végzi.

### 2.4.1. LCD regiszter

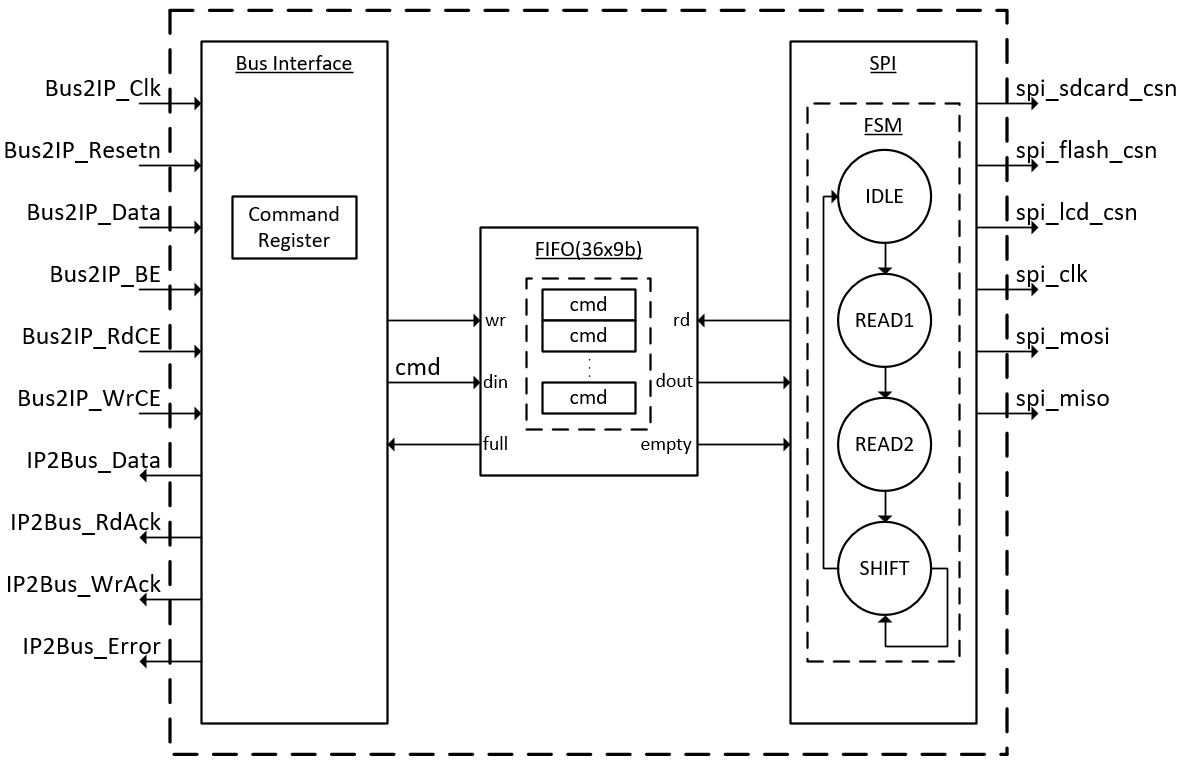
Az LCD perifériának egyetlen parancs regisztere van (Command Register), amely a **2.4.1. ábrán** látható, a bitek magyarázatával együtt. A regiszter 32 bites, írható és olvasható regiszter. A regiszter tartalmazza, hogy az elküldeni kívánt adatot az LCD parancsként vagy adatként értelmezze, tartalmazza magát a 8 bites parancsot is, valamint egy státusz bitet, amely jelzi, hogy lehet-e új parancsot küldeni az LCD-nek.



2.4.1. ábra: LCD regiszter

### 2.4.2. LCD blokkvázlata

Az LCD blokkvázlata a **2.4.2. ábrán** látható.



2.4.2. ábra LCD blokkvázlata

A blokkvázlat három fő modulra bontható fel, amelyet a top modul fog össze, amely csak wrapper-ként szolgál ebben az alkalmazásban. A top modul felel meg az IP fejlesztésben a „user logic”-nak, tehát a végső IP-ban e fölött a modul fölött még van egy AXI4 busz wrapper, de a fejlesztés során a „user logic” elkészítése volt a cél, az AXI4 busz wrapperben csak apró módosításokat kell elvégezni (külső IP portok deklarációja, összekötése). Ezért a továbbiakban a top modul a „user logic”-ra vonatkozik.

A top modul Verilog nyelven íródott, az almodulok VHDL nyelven.

#### 2.4.2.1. Bus Interface

A Bus Interface felelős a MicroBlaze felől érkező periféria hozzáférés kezeléséért. A modul tartalmazza a **2.4.1.** fejezetben ismertetett LCD regisztert. A külső busz jelek:

* Bus2IP\_Clk: Az 50Mhz-es rendszer órajel, amely a buszon történő kommunikációt is ütemezi.
* Bus2IP\_Resetn: A periféria reset jele. Negált reset.
* Bus2IP\_Data: A processzor felől érkező 32 bites adat íráskor.
* Bus2IP\_BE: Byte enable jel. A byte enable segítségével a 32 biten belül 4db 8-as csoportot lehet kiválasztani. az LCD IP esetén csak 32 bites hozzáférés engedélyezett, tehát a periféria csak akkor szólítható meg, ha a Bus2IP\_BE értéke csupa ’1’-es.
* Bus2IP\_RdCE: Jelzi, ha a processzor olvasás műveletet kezdeményez
* Bus2IP\_WrCE: Jelzi, ha a processzor írás műveletet kezdeményez.
* IP2Bus\_Data: A processzor felé küldendő 32 bites adat olvasáskor.
* IP2Bus\_RdAck: A periféria jelzése a processzor felé, ha sikeres volt az olvasás művelet. Jelen esetben ez a jel akkor logikai magas szintű, ha az összes byte enable jel ’1’-es és a processzor olvasás műveletet kezdeményezett.
* IP2Bus\_WrAck: A periféria jelzése a processzor felé, ha sikeres volt az írás művelet. Jelen esetben ez a jel akkor logikai magas szintű, ha az összes byte enable jel ’1’-es és a processzor írás műveletet kezdeményezett.
* IP2Bus\_Error: A periféria jelzése a processzor felé, ha sikertelen volt a busz kommunikáció. Ez a jel konstans alacsony logikai szinttel van meghajtva, nincs metódus a busz kommunikációs hibák kezelésre.

A Bus Interface modulnak vannak belső jelei is, amelyek a következő modult, a FIFO-t vezérli.

A busz kommunikációs kezeléséről a **2.4.2.1. táblázatban** látható egy-egy szimulációs kép. A felső kép egy busz írási ciklus, az alsó egy olvasás. A szimuláció egy a BFM (Bus Functional Model) szerint megírt írási task és olvasási task segítségével készült.

|  |
| --- |
| C:\Users\Crimson\Pictures\Screenpresso\2017-11-19_01h30_55.png |
| C:\Users\Crimson\Pictures\Screenpresso\2017-11-19_01h31_48.png |

2.4.2.1. táblázat: Bus Interface szimuláció. Fent írási ciklus, alul olvasási ciklus

#### 2.4.2.2. FIFO

A FIFO modul felelős a processzor felől érkező, a Bus Interface által kezelt LCD parancsok átmeneti tárolásáért.

Ezek a parancsok tartalmazzák a 8 bites „cmd” részt és az 1 bites „mode” részt (lásd **2.4.1.** fejezet), így a FIFO szélessége 9 bites, ahol az MSB a „mode” és az alsó 8 bit a „cmd”.

Az LCD periféria használatakor a Snake játékból adódóan általában 3x3 pixel blokkokat akarunk kirajzolni. Az LCD dokumentációja szerint ehhez az alábbi parancsok szükségesek:

1. SRAM lapcím beállítás
2. Oszlopcím beállítás
3. Adat kiírás

Egy adat kiírásakor az oszlopcím inkrementálisan növekszik, ezért a teljes 3x3-as blokk frissítés a következőképpen néz ki a felsorolásban lévő betűk felhasználásával:

***A, B, C, C, C -- A, B, C, C, C***

A lapcímet nem lenne szükséges minden esetben módosítani, de előfordulhat, hogy egy 3x3-as blokk kirajzolásához 2 memória lapot is írni kell. A 3x3 blokk parancssorozata tehát összesen 10 parancsból áll.

Azonban, általában a játék során olyan események történnek, amelyhez két darab 3x3-as blokk frissítése szükséges. Például: a kígyó mozog, tehát a fejét odébb kell mozgatni, és a farkának utolsó szegmensét letörölni. Ha a kígyó megeszik egy ételt, akkor a fejét odébb kell mozgatni, és új ételt kell generálni.

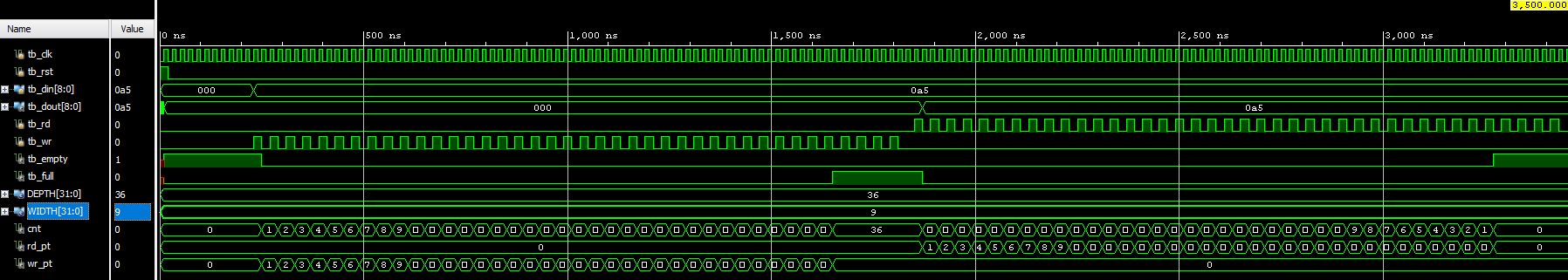
Ezek miatt a megfontolások miatt lett a FIFO mélysége 20, amely tehát 2 darab 3x3-as blokk parancsainak egyidejű tárolására alkalmas.

A FIFO modul külső jelei, a szokásos órajel és reset mellett:

* din: Data In, azaz a beérkező 9 bites adat.
* dout: Data Out, azaz a kimenő 9 bites adat.
* rd: Read, azaz olvasás műveletet jelző egy bites jel.
* wr: Write, azaz írás műveletet jelző egy bites jel.
* empty: Jelzi, ha üres a FIFO
* full: Jelzi, ha a FIFO tele van

A FIFO-t lehet írni, olvasni, vagy mindkét műveletet egyszerre elvégezni. Kialakítását tekintve nem shift regiszteres, olvasás és írás pointert használ.

A FIFO modul szimulációja a **2.4.2.2. ábrán** látható. A szimulációban egymás után 40 írás művelet történik, amely teleírja a FIFO-t, majd ez után 40 olvasás, amely teljesen kiüríti.



2.4.2.2. ábra FIFO szimuláció

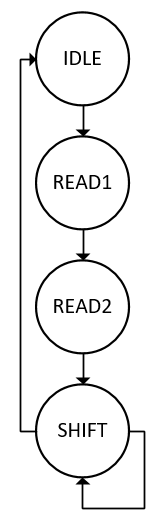
#### 2.4.2.3. SPI

Az LCD kijelző a Logsys panel SPI buszán keresztül vezérelhető. Az LCD kijelző mellett a buszra kapcsolódik még a soros Flash memória és a MicroSD kártya. A projektben csak az LCD-t használjuk az SPI-ra kapcsolódó perifériák közül.

Az SPI modul külső portjai a következők:

* spi\_sdcard\_csn: Az SD kártya kiválasztó jele. Aktív alacsony, tehát konstans ’1’-el van meghajtva, mert a projektben nem használjuk ezt a perifériát.
* spi\_flash\_csn: A Flash memória kiválasztó jele. Aktív alacsony, tehát konstans ’1’-el van meghajtva, mert a projektben nem használjuk ezt a perifériát.
* spi\_lcd\_csn: Az LCD kiválasztó jele, aktív alacsony logika.
* spi\_sck: Az SPI kommunikáció órajele. LCD esetén a maximálisan megengedett órajel 33Mhz. A rendszer órajele 50Mhz. Az sck jelet ezért 25Mhz-re választottuk, mert az egy egyszerű órajel osztóval előállítható, nem szükséges hozzá PLL FPGA primitív, és ezzel a sebességgel is megfelelő játékélmény biztosítható. A dokumentáció szerint az SPI eszközök az órajel felfutó élére mintavételeznek, tehát az adatot a lefutó élkor kell elküldeni.
* spi\_mosi: Az LCD parancs „cmd” részét ezen a vonalon kell kiléptetni az eszköz számára. Az MSB az első kiléptetett bit az átvitel során.
* spi\_miso: Általában ez a vonal a kommunikációban a master eszköz (FPGA) bemenete, amit a slave eszköz hajt meg. Az LCD periféria esetén ezt a vonalat szintén a master hajtja meg, a kommunikáció során az utolsó „cmd” bittel egy időben ezt a vonalat meg kell hajtani a „mode” bit értékével. Tehát ez a port is kimenet lesz.

Az SPI modul megfelelő működését egy FSM vezérli, amely a **2.4.2.3.a. ábrán** látható.

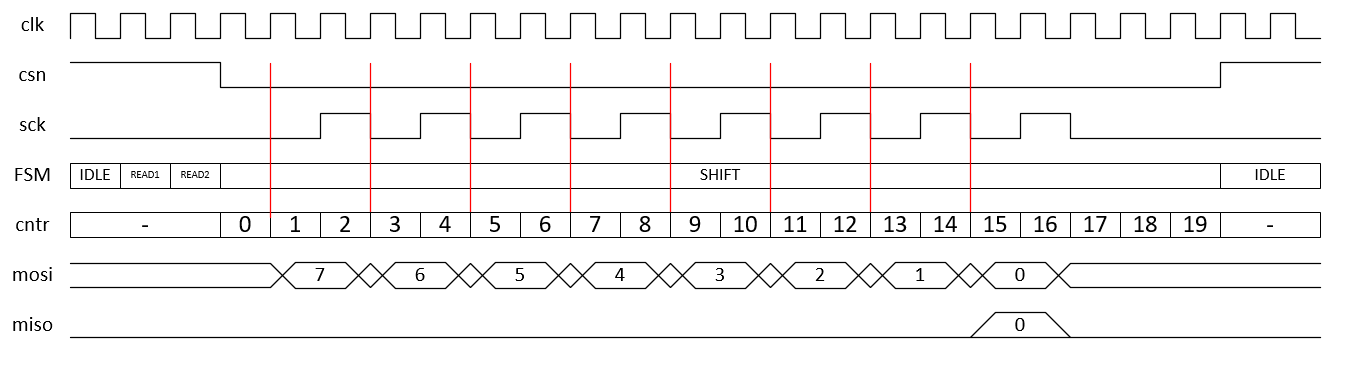


2.4.2.3.a. ábra: SPI FSM

Az állapotgép leírása a következő:

* IDLE állapot: Kezdetben (reset után) ebben az állapotban van. A vezérlő jeleket a passzív logikai szinten hajtja meg. Ha a FIFO „emtpy” jele logikai ’0’ szinten van, azaz van adat a FIFO-ban amit el kell küldeni, akkor meghajtja a FIFO olvasást kezdeményező vezetékét és a READ1 állapotba lép.
* READ1 állapot: A vezérlő jeleket szintén passzív állapotban tartja, erre az állapotra csak a FIFO olvasás miatt van szükség, feltétel nélkül a READ2 állapotba lép.
* READ2 állapot: Beolvassa a FIFO-ból kiolvasott adatokat és eltárolja shift regiszterben, majd feltétel nékül a SHIFT állapotba lép.
* SHIFT: Ebben az állapotban történik az adatok kishiftelése. Ebben egy számláló nyújt segítséget, amellyel meghatározható, hogy az sck felfutó vagy lefutó élénél tartunk, illetve egyéb időzítési feladatokban is segít. SHIFT állapotban ha a FIFO-ban még mindig van adat, akkor azt kiolvassa és az PI kommunikációs ciklust újra indítja és újból elkezdi shiftelni a beolvasott értéket a spi\_lcd\_csn jel magasba állítása nélkül. Ha a FIFO-ban nincs adat, akkor az IDLE állapotba lép.

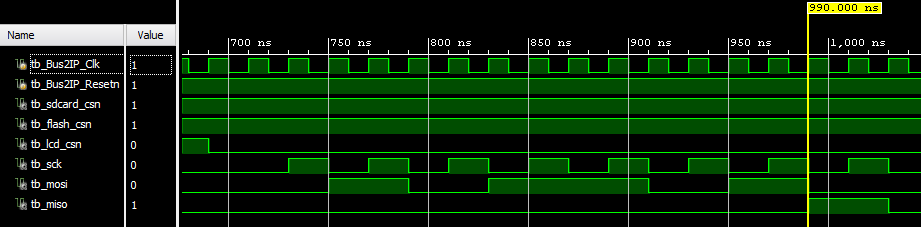
A tervezett SPI kommunikációs ciklus a **2.4.2.3.b. ábrán**  látható.



2.4.2.3.b. ábra: SPI kommunikációs ciklus

Az ábrán pirossal vannak jelezve azok az órajelek, amikor shiftelni kell az adatot, hogy az sck felfutó élére stabil legyen az LCD számára. Külön figyelmet érdemel még az, hogy a spi\_lcd\_scn jel alacsonyba állítása után az sck generálásáig szokott lenni egy rövid várakozás, az adatlapból ez az érték nem derül ki, én 2 órajelre választottam.

Az SPI szimulációja a **2.4.2.3.c. ábrán** látható.



2.4.2.3.c. ábra SPI szimuláció

Érdemes megfigyelni, hogy az sck felfutó élekor a mosi adat már stabil. Az ábrán küldött érték „0x5A”.

## 2.5. Tesztelés

Az összerakott hardver tesztelése az xmd parancssoros felület segítségével történt. A „0x50000000” báziscímen lévő LCD IP periféria teszteléséhez az alábbi parancsok szükségesek:

* connect mb mdm: Csatlakozás a MicroBlaze Debug Module-hoz
* stop: processzor megállítása
* mwr 0x50000000 0x2F: tápellátás bekapcsolása
* mwr 0x50000000 0xAF: kijelző engedélyezés, megjelenítés bekapcsolása

A kiadott parancsok következtében a kijelző megjeleníti az SRAM tartalmát.

# 3. Szoftver

## 3.1. Tervezés

## 3.2. Implementáció

## 3.3. Tesztelés

# Függelék - LCD periféria IP kódok:

## LCD – Top modul (user logic.v)

|  |  |
| --- | --- |
| `uselib lib=unisims\_ver  `uselib lib=proc\_common\_v3\_00\_a  module user\_logic #(  parameter C\_NUM\_REG = 1,  parameter C\_SLV\_DWIDTH = 32  ) (  input wire Bus2IP\_Clk,  input wire Bus2IP\_Resetn,  //input wire [31:0] Bus2IP\_Addr,  //input wire [0:0] Bus2IP\_CS,  //input wire Bus2IP\_RNW,  input wire [C\_SLV\_DWIDTH-1:0] Bus2IP\_Data,  input wire [C\_SLV\_DWIDTH/8-1:0] Bus2IP\_BE,  input wire [C\_NUM\_REG-1:0] Bus2IP\_RdCE,  input wire [C\_NUM\_REG-1:0] Bus2IP\_WrCE,  output wire [C\_SLV\_DWIDTH-1:0] IP2Bus\_Data,  output wire IP2Bus\_RdAck,  output wire IP2Bus\_WrAck,  output wire IP2Bus\_Error,    output wire sdcard\_csn,  output wire flash\_csn,  output wire lcd\_csn,  output wire sck,  output wire mosi,  output wire miso  );  wire rst = ~Bus2IP\_Resetn;  wire full;  wire empty;  wire rd;  wire wr;  wire [8:0] cmd2fifo;  wire [8:0] cmd2spi;  wire sh;  wire stat\_empty = empty;  wire stat\_spi\_busy = sh;  reg stat\_fifo\_neg;  reg stat\_spi\_neg;  always @ (posedge Bus2IP\_Clk)  begin  if(rst == 1) begin  stat\_fifo\_neg <= 0;  stat\_spi\_neg <= 0;  end else begin  if(rd == 1)  stat\_fifo\_neg <= ~stat\_fifo\_neg;  if(sh == 1)  stat\_spi\_neg <= 1;  end  end | bus\_if bus\_if\_inst(  .Bus2IP\_Clk(Bus2IP\_Clk),  .Bus2IP\_Resetn(Bus2IP\_Resetn),  .Bus2IP\_Data(Bus2IP\_Data),  .Bus2IP\_BE(Bus2IP\_BE),  .Bus2IP\_RdCE(Bus2IP\_RdCE[0]),  .Bus2IP\_WrCE(Bus2IP\_WrCE[0]),  .IP2Bus\_Data(IP2Bus\_Data),  .IP2Bus\_RdAck(IP2Bus\_RdAck),  .IP2Bus\_WrAck(IP2Bus\_WrAck),  .IP2Bus\_Error(IP2Bus\_Error),  .cmd(cmd2fifo),  .wr(wr),  .full(full),  .stat\_empty(stat\_empty),  .stat\_spi\_busy(stat\_spi\_busy),  .stat\_fifo\_neg(stat\_fifo\_neg),  .stat\_spi\_neg(stat\_spi\_neg)  );  fifo fifo\_inst(  .clk(Bus2IP\_Clk),  .rst(rst),  .din(cmd2fifo),  .dout(cmd2spi),  .rd(rd),  .wr(wr),  .empty(empty),  .full(full)  );    spi spi\_inst(  .clk(Bus2IP\_Clk),  .rst(rst),  .sdcard\_csn(sdcard\_csn),  .flash\_csn(flash\_csn),  .lcd\_csn(lcd\_csn),  .sck(sck),  .mosi(mosi),  .miso(miso),  .empty(empty),  .rd(rd),  .din(cmd2spi),  .sh(sh)  );  endmodule |

## LCD – Bus Interface (bus\_if.vhd)

|  |  |
| --- | --- |
| ---------------------------------------------------------------------------------------  -- Company: BME  -- Engineer: Cseh Peter (DM5HMB), Limbay Bence (E2JT1E)  --  -- Create Date: 2017.10.29  -- Design Name: lcd  -- Module Name: bus\_if  ---------------------------------------------------------------------------------------  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.NUMERIC\_STD.ALL;  --library UNISIM;  --use UNISIM.VComponents.all;  entity bus\_if is  generic  (  C\_NUM\_REG : integer := 1;  C\_SLV\_DWIDTH : integer := 32  );  port  (  --Bus signals  Bus2IP\_Clk : in std\_logic;  Bus2IP\_Resetn : in std\_logic;  Bus2IP\_Data : in std\_logic\_vector((C\_SLV\_DWIDTH - 1) downto 0);  Bus2IP\_BE : in std\_logic\_vector((C\_SLV\_DWIDTH / 8 - 1) downto 0);  Bus2IP\_RdCE : in std\_logic;  Bus2IP\_WrCE : in std\_logic;  IP2Bus\_Data : out std\_logic\_vector((C\_SLV\_DWIDTH - 1) downto 0);  IP2Bus\_RdAck : out std\_logic;  IP2Bus\_WrAck : out std\_logic;  IP2Bus\_Error : out std\_logic;  --Inner signals  cmd : out std\_logic\_vector(8 downto 0);  wr : out std\_logic;  full : in std\_logic;    --Status signals  stat\_empty : in std\_logic;  stat\_spi\_busy : in std\_logic;  stat\_fifo\_neg : in std\_logic;  stat\_spi\_neg : in Std\_logic  );  end bus\_if;  architecture rtl of bus\_if is  signal clk : std\_logic := '0';  signal rst : std\_logic := '0';  signal cmd\_reg : std\_logic\_vector((C\_SLV\_DWIDTH - 1) downto 0) := (others => '0');  begin  ---------------------------------------------------------------------------------------  -- Clock and Reset signals  ---------------------------------------------------------------------------------------  clk <= Bus2IP\_Clk;  rst <= not(Bus2IP\_Resetn); | ---------------------------------------------------------------------------------------  -- Bus READ  ---------------------------------------------------------------------------------------  IP2Bus\_Data <= cmd\_reg when (Bus2IP\_RdCE = '1' and Bus2IP\_BE = "1111") else (others => '0');  ---------------------------------------------------------------------------------------  -- Bus WRITE  ---------------------------------------------------------------------------------------  proc\_write : process(clk)  begin  if(rising\_edge(clk)) then  if(rst = '1') then  cmd\_reg <= (others => '0');  else  --WRITE BUS  if(Bus2IP\_WrCE = '1' and Bus2IP\_BE = "1111") then  cmd\_reg <= Bus2IP\_Data;  if(full = '1') then  wr <= '0';  else  wr<= '1';  end if;  --STATUS BITS  else  if(full = '1') then --Command Register Status: Unavaible  cmd\_reg(12) <= '0';  else  cmd\_reg(12) <= '1';  end if;  cmd\_reg(31) <= stat\_empty;  cmd\_reg(30) <= stat\_spi\_busy;  cmd\_reg(29) <= stat\_fifo\_neg;  cmd\_reg(28) <= stat\_spi\_neg;  wr <= '0';  end if;  end if;  end if;  end process proc\_write;  cmd <= cmd\_reg(8 downto 0);  ---------------------------------------------------------------------------------------  -- Bus ACKnowledge and ERROR signals  ---------------------------------------------------------------------------------------  IP2Bus\_RdAck <= '1' when (Bus2IP\_RdCE = '1' and Bus2IP\_BE = "1111") else '0';  IP2Bus\_WrAck <= '1' when (Bus2IP\_WrCE = '1' and Bus2IP\_BE = "1111") else '0';  IP2Bus\_Error <= '0';  end rtl; |

## LCD – FIFO (fifo.vhd)

|  |  |
| --- | --- |
| ---------------------------------------------------------------------------------------  -- Company: BME  -- Engineer: Cseh PÃ©ter (DM5HMB), Limbay Bence (E2JT1E)  --  -- Create Date: 2017.10.29  -- Design Name: lcd  -- Module Name: fifo  ---------------------------------------------------------------------------------------  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.NUMERIC\_STD.ALL;  library UNISIM;  use UNISIM.VComponents.all;  entity fifo is  generic  (  DEPTH : integer := 20;  WIDTH : integer := 9  );  port  (  clk : in std\_logic;  rst : in std\_logic;  din : in std\_logic\_vector((WIDTH - 1) downto 0);  dout : out std\_logic\_vector((WIDTH - 1) downto 0);  rd : in std\_logic;  wr : in std\_logic;  empty : out std\_logic;  full : out std\_logic  );  end fifo;  architecture rtl of fifo is  type mem\_type is array ((DEPTH - 1) downto 0) of std\_logic\_vector((WIDTH - 1) downto 0);  signal mem : mem\_type := (others => (others => '0'));  signal cnt : integer range 0 to DEPTH := 0;  signal rd\_pt : integer range 0 to (DEPTH - 1) := 0;  signal wr\_pt : integer range 0 to (DEPTH - 1) := 0;  begin  proc\_fifo : process(clk)  begin  if(rising\_edge(clk)) then  if(rst = '1') then  dout <= (others => '0');  empty <= '1';  full <= '0';  rd\_pt <= 0;  wr\_pt <= 0;  cnt <= 0;  else | --ONLY READ  if(rd = '1' and wr = '0') then  if(cnt = 1) then --EMPTY  empty <= '1';  end if;  full <= '0'; --FULL  if(cnt > 0) then  cnt <= cnt - 1; --COUNT  if(rd\_pt = (DEPTH - 1)) then --READ POINTER rd\_pt <= 0;  else  rd\_pt <= rd\_pt + 1;  end if;  end if;  dout <= mem(rd\_pt); --READ DATA    end if;    --ONLY WRITE  if(wr = '1' and rd = '0') then  if(cnt = (DEPTH - 1)) then --FULL  full <= '1';  end if;  empty <= '0'; --EMPTY  if(cnt < DEPTH) then  cnt <= cnt + 1; --COUNT  if(wr\_pt = (DEPTH - 1)) then --WRITE POINTER  wr\_pt <= 0;  else  wr\_pt <= wr\_pt + 1;  end if;  mem(wr\_pt) <= din; --WRITE DATA  end if;  end if;  --BOTH READ AND WRITE  if(wr = '1' and rd = '1') then  mem(wr\_pt) <= din; --WRITE DATA  dout <= mem(rd\_pt); --READ DATA  if(rd\_pt = (DEPTH - 1)) then --READ POINTER  rd\_pt <= 0;  else  rd\_pt <= rd\_pt + 1;  end if;  if(wr\_pt = (DEPTH - 1)) then --WRITE POINTER  wr\_pt <= 0;  else  wr\_pt <= wr\_pt + 1;  end if;    end if;  end if;  end if;  end process proc\_fifo;  end rtl; |

## LCD – SPI (spi.vhd)

|  |  |
| --- | --- |
| ---------------------------------------------------------------------------------------  -- Company: BME  -- Engineer: Cseh Peter (DM5HMB), Limbay Bence (E2JT1E)  --  -- Create Date: 2017.10.29  -- Design Name: lcd  -- Module Name: spi  ---------------------------------------------------------------------------------------  library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.NUMERIC\_STD.ALL;  --library UNISIM;  --use UNISIM.VComponents.all;  entity spi is  port  (  clk : in std\_logic;  rst : in std\_logic;  --SPI signals  sdcard\_csn : out std\_logic;  flash\_csn : out std\_logic;  lcd\_csn : out std\_logic;  sck : out std\_logic;  mosi : out std\_logic;  miso : out std\_logic;  --Inner signals  empty : in std\_logic;  rd : out std\_logic;  din : in std\_logic\_vector(8 downto 0);    --Stat signal  sh : out std\_logic  );  end spi;  architecture rtl of spi is  signal fsm\_cntr : integer range 0 to 7 := 0;  type spi\_fsm\_type is (IDLE, READ1, READ2, SHIFT);  signal fsm : spi\_fsm\_type := IDLE;  signal cmd\_reg : std\_logic\_vector(7 downto 0) := (others => '0');  signal mode\_reg : std\_logic := '0';  signal sck\_wire : std\_logic := '0';  signal cont\_send : std\_logic := '0';  signal shift\_end : std\_logic := '0';  signal send\_cntr : std\_logic\_vector(4 downto 0) := (others => '0'); --0 to 20  signal rise : std\_logic := '0';  signal fall : std\_logic := '0';  begin  ---------------------------------------------------------------------------------------  -- Drive unused spi channel with HIGH  ---------------------------------------------------------------------------------------  sdcard\_csn <= '1';  flash\_csn <= '1';  ---------------------------------------------------------------------------------------  -- Clock divider for sck, and send signals  ---------------------------------------------------------------------------------------  proc\_clk\_div : process(clk)  begin  if(rising\_edge(clk)) then  if(rst = '1') then  send\_cntr <= (others => '0');  else  case fsm is  when SHIFT =>  if(cont\_send = '1' and to\_integer(unsigned(send\_cntr)) = 18) then  send\_cntr <= "00010";  else  if(to\_integer(unsigned(send\_cntr)) = 20) then  send\_cntr <= (others => '0');  else  send\_cntr <= std\_logic\_vector(unsigned(send\_cntr) + 1);  end if;  end if;  when others => send\_cntr <= (others => '0');  end case;  end if;  end if;  end process proc\_clk\_div;  rise <= '1' when send\_cntr(0) = '0' else '0';  fall <= '1' when send\_cntr(0) = '1' else '0';  ---------------------------------------------------------------------------------------  -- Finite State Machine  ---------------------------------------------------------------------------------------  sck <= sck\_wire;  sh <= '1' when (fsm = SHIFT) else '0'; | proc\_fsm : process(clk)  begin  if(rising\_edge(clk)) then  if(rst = '1') then  fsm <= IDLE;  else  case fsm is  --IDLE STATE  when IDLE =>  if(empty = '0') then --IDLE -> SHIFT  rd <= '1';  fsm <= READ1;  else  rd <= '0';  end if;  lcd\_csn <= '1';  mosi <= '0';  miso <= '0';  sck\_wire <= '0';    when READ1 =>    lcd\_csn <= '1';  mosi <= '0';  miso <= '0';  sck\_wire <= '0';    rd <= '0';  fsm <= READ2;    when READ2 =>    lcd\_csn <= '1';  mosi <= '0';  miso <= '0';  sck\_wire <= '0';    rd <= '0';  cmd\_reg <= din(7 downto 0);  mode\_reg <= din(8);    fsm\_cntr <= 7;  fsm <= SHIFT;  --SHIFT STATE  when SHIFT =>  if(to\_integer(unsigned(send\_cntr)) < 1) then  lcd\_csn <= '0';  mosi <= '0';  miso <= '0';  sck\_wire <= '0';  else    --SCK vezérlés  if(to\_integer(unsigned(send\_cntr)) < 18) then  if(rise = '1') then  sck\_wire <= '1';  end if;  if(fall = '1') then  sck\_wire <= '0';  end if;  end if;    --MOSI & MISO  if(fall = '1' and to\_integer(unsigned(send\_cntr)) < 19) then  mosi <= cmd\_reg(7);  cmd\_reg <= cmd\_reg(6 downto 0) & '0';    if(to\_integer(unsigned(send\_cntr)) = 15) then  miso <= mode\_reg;  else  miso <= '0';  end if;  end if;    if(to\_integer(unsigned(send\_cntr)) = 16 and empty = '0') then  cont\_send <= '1';  rd <= '1';  end if;  if(to\_integer(unsigned(send\_cntr)) = 17 and cont\_send = '1') then  cont\_send <= '1';  rd <= '0';  end if;  if(to\_integer(unsigned(send\_cntr)) = 18 and cont\_send = '1') then  cont\_send <= '0';  rd <= '0';  cmd\_reg <= din(7 downto 0);  mode\_reg <= din(8);      end if;    if(to\_integer(unsigned(send\_cntr)) = 19) then  fsm <= IDLE;  end if;    end if;  when others => fsm <= IDLE;  end case;  end if;  end if;  end process proc\_fsm;  end rtl; |

## LCD – Top modul szimuláció (lcd\_sim.v)

|  |  |
| --- | --- |
| `timescale 1ns / 1ps  module lcd\_sim();  parameter C\_SLV\_DWIDTH = 32;  parameter C\_NUM\_REG = 1;  reg tb\_Bus2IP\_Clk;  reg tb\_Bus2IP\_Resetn;  reg [C\_SLV\_DWIDTH-1:0] tb\_Bus2IP\_Data;  reg [C\_SLV\_DWIDTH/8-1:0] tb\_Bus2IP\_BE;  reg [C\_NUM\_REG-1:0] tb\_Bus2IP\_RdCE;  reg [C\_NUM\_REG-1:0] tb\_Bus2IP\_WrCE;  wire [C\_SLV\_DWIDTH-1:0] tb\_IP2Bus\_Data;  wire tb\_IP2Bus\_RdAck;  wire tb\_IP2Bus\_WrAck;  wire tb\_IP2Bus\_Error;  wire tb\_sdcard\_csn;  wire tb\_flash\_csn;  wire tb\_lcd\_csn;  wire tb\_sck;  wire tb\_mosi;  wire tb\_miso;  user\_logic user\_logic\_inst(  .Bus2IP\_Clk(tb\_Bus2IP\_Clk),  .Bus2IP\_Resetn(tb\_Bus2IP\_Resetn),  .Bus2IP\_Data(tb\_Bus2IP\_Data),  .Bus2IP\_BE(tb\_Bus2IP\_BE),  .Bus2IP\_RdCE(tb\_Bus2IP\_RdCE),  .Bus2IP\_WrCE(tb\_Bus2IP\_WrCE),  .IP2Bus\_Data(tb\_IP2Bus\_Data),  .IP2Bus\_RdAck(tb\_IP2Bus\_RdAck),  .IP2Bus\_WrAck(tb\_IP2Bus\_WrAck),  .IP2Bus\_Error(tb\_IP2Bus\_Error),  .sdcard\_csn(tb\_sdcard\_csn),  .flash\_csn(tb\_flash\_csn),  .lcd\_csn(tb\_lcd\_csn),  .sck(tb\_sck),  .mosi(tb\_mosi),  .miso(tb\_miso)  );    initial begin  tb\_Bus2IP\_Clk <= 0;  tb\_Bus2IP\_Resetn <= 0;  tb\_Bus2IP\_Data <= 0;  tb\_Bus2IP\_BE <= 0;  tb\_Bus2IP\_RdCE <= 0;  tb\_Bus2IP\_WrCE <= 0; | #20  tb\_Bus2IP\_Resetn <= 1;    #20  //STIMULUS  write(32'h000001FF);  #500  repeat (40) begin  write(32'h0000015A);  end  read();    end  //GENERATE CLOCK (50MHz)  always #10 tb\_Bus2IP\_Clk = ~tb\_Bus2IP\_Clk;    //WRITE COMMAND  task write;  input [31:0] item;  begin  @ (posedge tb\_Bus2IP\_Clk)  tb\_Bus2IP\_WrCE <= 1'b1;  tb\_Bus2IP\_Data <= item;  tb\_Bus2IP\_BE <= 4'b1111;  @ (posedge tb\_Bus2IP\_Clk)  tb\_Bus2IP\_WrCE <= 1'b0;  tb\_Bus2IP\_Data <= 0;  tb\_Bus2IP\_BE <= 4'b0000;  end  endtask  //READ COMMAND  task read;  begin  @ (posedge tb\_Bus2IP\_Clk)  tb\_Bus2IP\_RdCE <= 1'b1;  tb\_Bus2IP\_BE <= 4'b1111;  @ (posedge tb\_Bus2IP\_Clk)  tb\_Bus2IP\_RdCE <= 1'b0;  tb\_Bus2IP\_BE <= 4'b0000;  end  endtask  endmodule |

## LCD – FIFO szimuláció (fifo\_sim.v)

|  |  |
| --- | --- |
| `timescale 1ns / 1ps  module fifo\_sim();  parameter DEPTH = 36;  parameter WIDTH = 9;  reg tb\_clk;  reg tb\_rst;  reg [(WIDTH - 1) : 0] tb\_din;  wire [(WIDTH - 1) : 0] tb\_dout;  reg tb\_rd;  reg tb\_wr;  wire tb\_empty;  wire tb\_full;  fifo fifo\_inst(  .clk(tb\_clk),  .rst(tb\_rst),  .din(tb\_din),  .dout(tb\_dout),  .rd(tb\_rd),  .wr(tb\_wr),  .empty(tb\_empty),  .full(tb\_full)  );    initial begin  tb\_clk <= 0;  tb\_rst <= 1;  tb\_din <= 0;  tb\_rd <= 0;  tb\_wr <= 0;    #20  tb\_rst <= 0; | #200  //STIMULUS    repeat (40) begin  write(9'h0A5);  end  #40  repeat (40) begin  read();  end    end  //GENERATE CLOCK (50MHz)  always #10 tb\_clk = ~tb\_clk;    //WRITE COMMAND  task write;  input [8:0] item;  begin  @ (posedge tb\_clk)  tb\_wr <= 1'b1;  tb\_din <= item;  @ (posedge tb\_clk)  tb\_wr <= 1'b0;  end  endtask  //READ COMMAND  task read;  begin  @ (posedge tb\_clk)  tb\_rd <= 1'b1;  @ (posedge tb\_clk)  tb\_rd <= 1'b0;  end  endtask  endmodule |

# Irodalomjegyzék