Budapesti Műszaki és Gazdaságtudományi Egyetem
Mikrorendszerek tervezése (VIMIMA14)

Snake LOGSYS Spartan6 FPGA-ra

HÁZI FELADAT

–
–

2017. december

Tartalomjegyzék

Ábrajegyzék

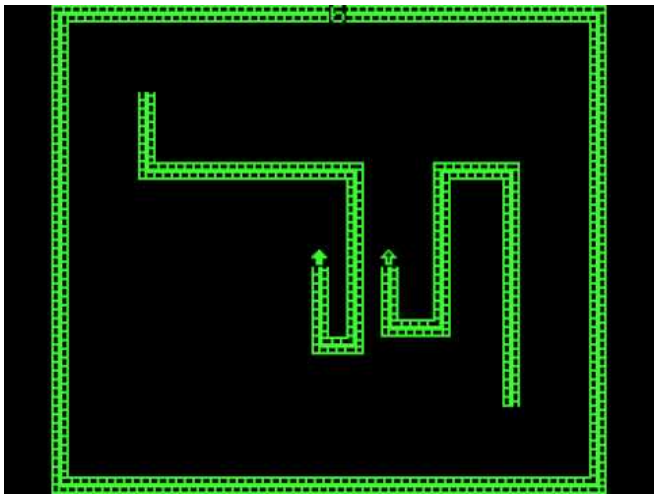
1. fejezet

A Snake videójáték

A házi feladat keretében egy kígyós játékot (Snake-et) kellett készítenünk egy LOGSYS Spartan6 FPGA-ra. Ehhez az FPGA logikából egy MicroBlaze alapú mikroprocesszoros rendszert építettünk meg, amely azért volt felelős, hogy végrehajtsa az általunk kitalál játék algoritmusát, ellássa adatokkal a fejlesztői kártya LCD kijelzőjét, valamint beolvassa a játék paraméterét a kártya DIP8-as kapcsolója segítségével. De még mielőtt rátérnénk az általunk kitalált algoritmus és a megtervezett rendszer részletes bemutatására, azelőtt egy rövid történelmi áttekintést szeretnénk adni a méltán híres kígyós játék kialakulásáról és fejlődéséről.

Amit valójában tudni kell a Snake-ről az az, hogy valójában nem egyetlen egy játékról van szó, hanem egy egyedi videójáték ötletről, koncepcióról, amelyben a játékosnak egy fokozatosan növekvő „vonallal" kell ügyeskednie a számára kijelölt területen. A játék legfőbb nehézségét a vonal hossza jelenti.

Ez az ötlet roppant sok formát öltött az első megjelenésétől számított néhány évtizedben. A legelső megvalósítása 1976-ban született, amely *Blockade* nevet kapta a Gremlin fejlesztőitől (a Gremlin árkád játékok fejlesztésével foglalkozó cég volt 1973 és 1984 között). Az 1.1. ábrán egy pillantfelvétel látható az említett játékról.



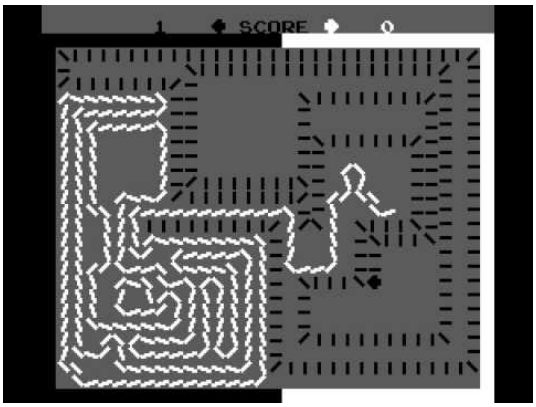
1.1. Ábra – *A **Blockade** nevű játék*

A kép alapján elsőre nehéz észrevenni, de valójában két játékost igényelt és a roppant egyszerű „ki bírja tovább" alapon működött. Úgy indul a játék, hogy az algoritmus választ véletlenszerűen két pozíciót a képernyőn és ezekből kezdik a játékosok a saját „folyamatosan növekedő kígyójuk" irányítását. Kígyó helyett inkább a gyors csiga lenne találó kifejezés erre, mivel ahogy bejárják a képernyőt, úgy húzzák maguk után a csíkot. Az a játékos nyer, amelyik tovább bírja anélkül, hogy eltalálná a másikat, önmagát, a csíkokat és a képernyő szélein található falakat. Természetesen, a cég, az akkori trendeknek megfelelően, pénzérmével működő játékgépek formájában forgalmazta ezt a játékot.

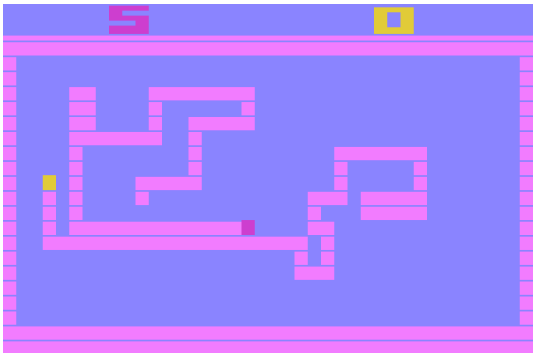
Még ugyanezen évben született belőle egy klón *Bigfoot Bonkers* néven a Meadows Games fejlesztői csapattól az 1.2. ábrán. Annyit tettek hozzá, hogy a játék kezdetén három lábnyomot is kisorsolt a gép a képernyőre nehezítésképpen, amelyeket szintén ki kellett kerülniük a játékosoknak.

1.2. Ábra – *A **Bigfoot Bonkers** játékgép*

A rákövetkező évben két újabb Blockade-klón jelent meg a híres *Atari* cégtől: a *Dominos* és a *Surround*. Az 1.3. ábrán látható róluk egy-egy kép.



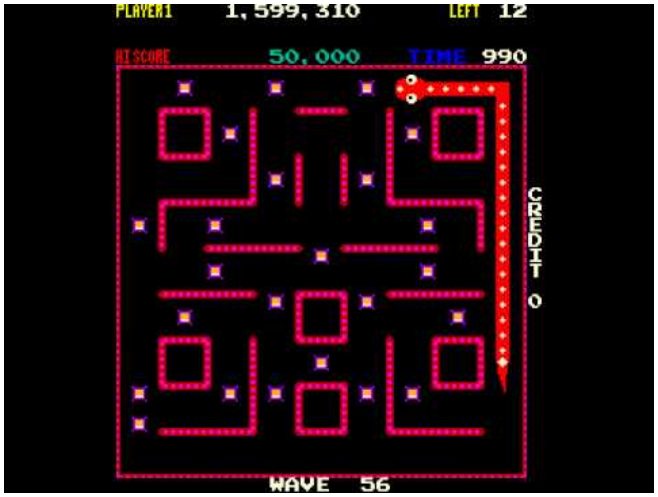
(a) *A **Dominos** játék*



(b) *A **Surround** játék*

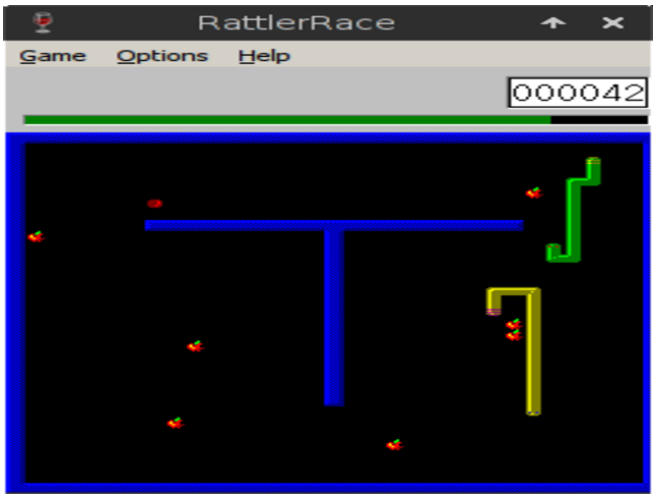
1.3. Ábra – *A két Atari klón*

A következő nagyobb mérföldkövet az 1982-ben kiadott a *Nibbler* nevű játék jelentette. Szakítottak a többjátékos megoldással és helyette a játékosnak egy labirintus-szerű pályán kellett ügyeskednie egy tényleges kígyóval (1.4. ábra).



1.4. Ábra – *A **Nibbler** játék*

1991-ben még az MS-DOS-ban is helyet kapott, viszont ’92-ben a Microsoft kiadta a saját kígyós játékát *Rattler Race* néven. Ezt mutatja be az 1.5. ábra. Az újdonsága az volt, hogy a játékoshoz tartozó kígyó mellett megjelentek az ellenséges kígyók, valamint a különböző színű és formájú almák is.



1.5. Ábra – *A **Rattler Race** játék*

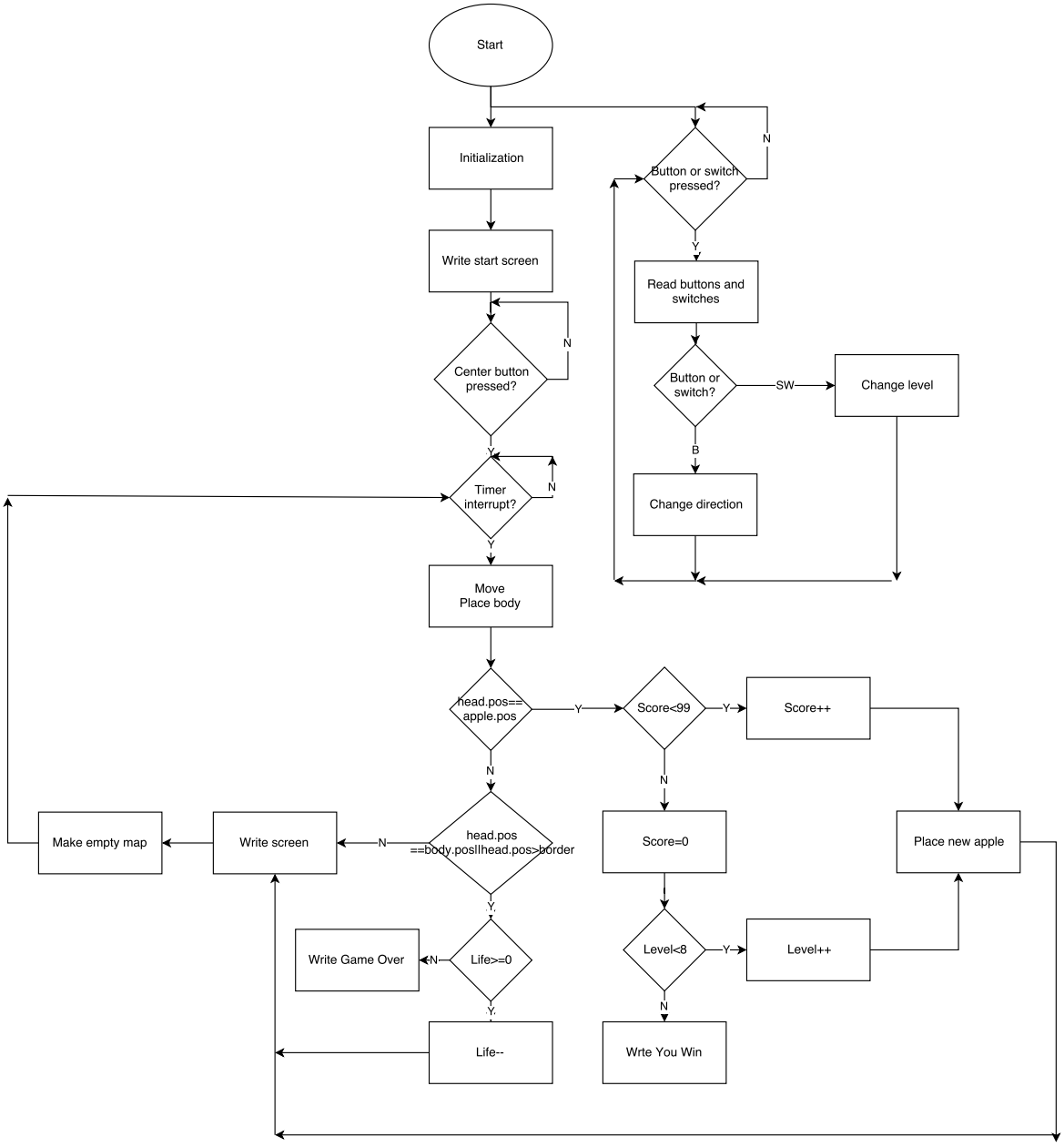
2016-ban bemutatták az első MMO-alapú Snake játékot, amely *Slither.io* néven fut és ezen az oldalon is érhető el (1.6. ábra).

1.6. Ábra – *A **Slither.io** játék*

2. fejezet

A Snake algoritmusa

Ebben a fejezetben az általunk kitalált algoritmus kerül bemutatásra, egy egyszerűsített folyamatábra segítségével. Ez a 2.1. ábrán látható.

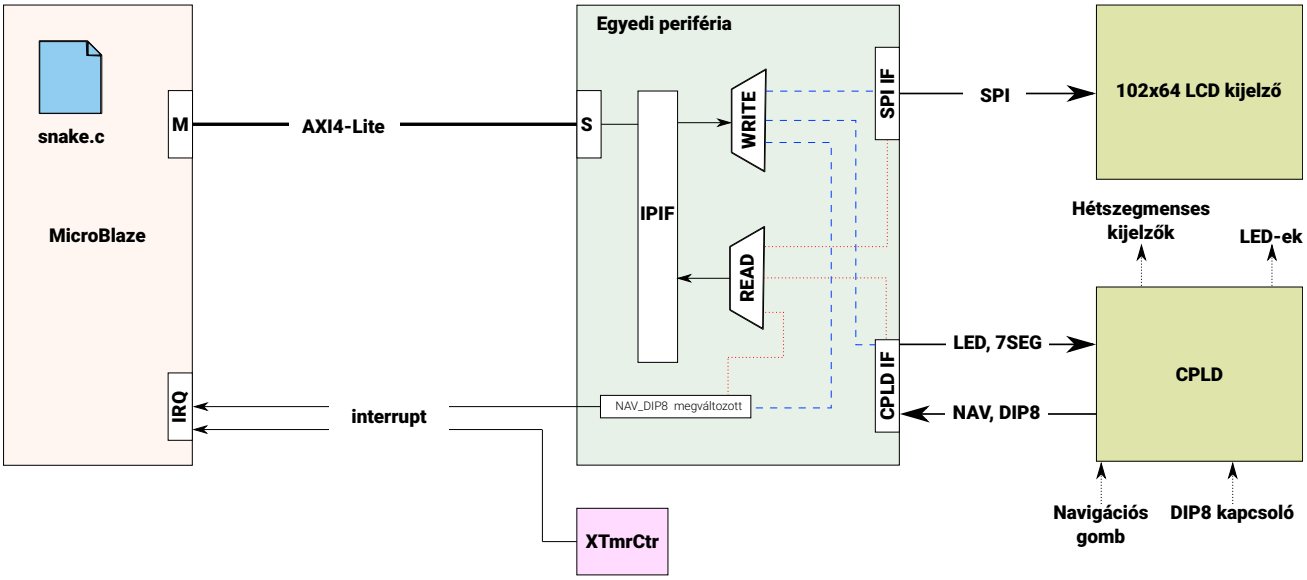


2.1. Ábra – *Az általunk létrehozott Snake működésének nagyvonalú algoritmusa*

3. fejezet

Rendszerterv

A játék algoritmusának vázolása után a beágyazott mikroprocesszoros rendszer felépítéséről lesz szó. A megvalósításhoz egy LOGSYS fejlesztői kártyát használtunk, amely tartalmazott egy Xilinx Spartan6 FPGA chipet és egy CPLD-t is. A rendszer nagy része az FPGA konfigurálható logikáját használva lett létrehozva (pl. a beágyazott processzoros környezet és az egyedi periféria), míg a maradék az előre felprogramozott CPLD logikájára támaszkodott (LED-ek és a különböző kapcsolók). A rendszer nagyvonalú terve a 3.1. ábrán látható.



3.1. Ábra – *A beágyazott processzoros rendszer rendszerterve*

A rendszer lelke a MicroBlaze processzor. Ez hajtja végre a játék algoritmusát leíró C forráskódot, reagál a felhasználói beavatkozásokra, valamint fogadja az időzítő/számláló (az ábrán XTmrCtr-rel jelölve) által küldött periodikus megszakításokat. Továbbá, ellátja adatokkal az LCD kijelzőt, a hétszegmenses kijelzőket és a LED-eket. Ezt egy köztes modulon keresztül teszi, amit itt *Egyedi periféria* névvel illettünk.

A processzor AXI4-Lite interfészen keresztül kommunikál az előbb említett modullal. Annak érdekében, hogy ne kelljen foglalkoznunk az AXI4-Lite Slave interfész megvalósításával, egy IPIF illesztőt kapcsoltunk a Slave portra, ami az AXI4-Lite kommunikációt szimpla memória írás/olvasás műveletté egyszerűsítette le. Erre az IPIF illesztőre kapcsolódik az SPI IF, a CPLD IF és a NAV_DIP8. Az utóbbi a navigációs gomb és a DIP8 kapcsoló aktuális értékét tárolja. Az SPI IF-en keresztül lehet elérni az LCD kijelző vezérlőjét, míg a CPLD IF-en magát a CPLD-ben megvalósított logikát.

A processzor két helyről kaphat megszakítást: az időzítő/számláló modultól, valamint az egyedi perifériában megvalósított logikától, amely figyeli a felhasználói beavatkozásokat a navigációs gombon és a DIP8 kapcsolón, majd megszakítást kér, ha változást észlel ezeken.

4. fejezet

A megvalósított rendszer

Ebben a fejezetben a rendszerterv alapján megvalósított FPGA környezetről lesz szó. Az itt szereplő egyes részeket próbáltuk úgy sorba rendezni, ahogy azok előjöttek a fejlesztés során. Ennek megfelelően először az SPI és a CPLD interfészek megvalósításáról lesz szó.

Az itt szereplő modulok leírásai a dokumentáció mellékleteként elérhetők, valamint a Függelékben (a 7. fejezetben) is szerepeltetve vannak.

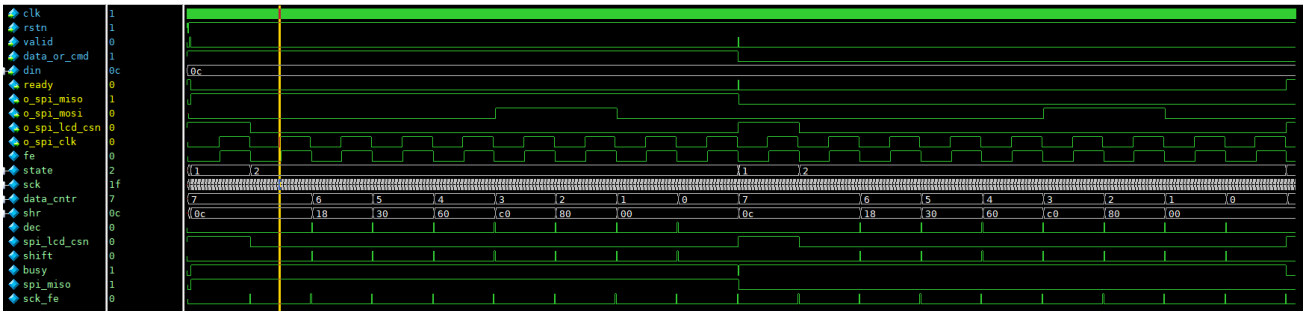
Az RTL modellek funkcionális verifikációjához a Mentor Graphics cég Questa Sim 10.4e HDL szimulátorát használtuk.

4.1. Az SPI interfész

A LOGSYS fejlesztői kártyán lévő LCD kijelzőt használtuk a Snake játék megjelenítésére. Az LCD-n való adatok megjelenítéséhez a kártya felhasználói útmutatójában lévő SPI időzítési diagramot kellett „lemásolnunk”. Azaz olyan logikát kellett készítenünk, amely az ott szereplő protokollt megvalósítja. Ehhez egy véges állapotú automatát használtunk (Finite State Machine, FSM), amivel ez könnyen megtehető volt.

Az SPI interfészre természetesen más perifériák (Flash, MicroSD kártya) is csatlakoztak, ám ezeket a rendszerünkben nem használtuk, így az azokhoz tartozó kiválasztó jeleket aktív magas szintre kötöttük. Továbbá, az SPI interfészt csak egyirányú adatkapcsolatra terveztük meg, hiszen a feladat nem igényelte azt, hogy az LCD-től adatot kérjünk le. Egyedül azt kellett számon tartatunk, hogy az adatátvitelt megvalósító FSM hol tart. Ehhez egy *ready* kimenetet hoztunk létre, amely azt reprezentálta, hogy az FSM képes egy újabb adat átvitelére.

Az FSM egy léptető regiszter segítségével hajtja meg az *o_spi_mosi* kimenetet. Az *o_spi_miso* az LCD vezérlése szempontjából kimenet lett, hiszen ezzel jeleztük az LCD felé, hogy a küldött információ adat vagy parancs. Az *o_spi_clk* órajelet az FPGA órajeléből egy 6 bites szabadon futó felfelé számlálóval állítottuk elő, nagyjából 0,7 MHz-esre. Az interfész viselkedésének szimulációja a 4.1. hullámformán látható.



4.1. Ábra – *Az SPI interfész hullámformája*

A modul RTL modellje a 7.1. szakaszban szerepel.

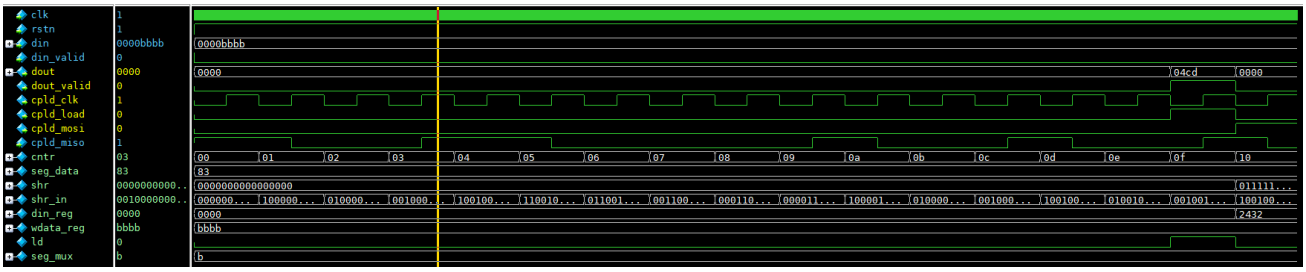
4.2. A CPLD interfészelése

A felhasználói beavatkozás (a kígyó mozgatása, kezdeti nehézségi szint beállítása) és a játék aktuális állapota (nehézségi szint, szerzett pontok) a fejlesztői kártyára integrált CPLD segítségével tehető meg. A CPLD eléréséhez az FPGA-ban egy olyan áramkört kellett megterveznünk, ami képes volt kommunikálni a CPLD-ben előre felkonfigurált logikával. Ehhez a LOGSYS felhasználói útmutatóban az ide illő időzítési diagramot vettük alapul.

A *cpld_miso* bemenetet egy 16 bites léptető regiszter (shr_in) MSB bitjére kötöttük, amelyet akkor léptettünk (ce), ha az FPGA órajelére működő 12 bites szabadon futó felfelé számláló (clk_div) MSB bitjén jött egy lefutó él. Ezzel egy nagyjából 12 kHz-es engedélyező jelet tudtunk létrehozni. Ha beérkezett mind a 16 adatbit a CPLD-től (ld), ezt is egy számlálóval mértük, akkor a léptetőregiszter tartalmát átvettük egy átmeneti regiszterbe (din_reg), ahonnan az adat a modul olvasásra szánt portjain elérhetővé válik.

A *cpld_mosi* kimenetet is egy 16 bites léptető regiszterrel hajtottuk meg (shr), amelynek tartalmának egy részét (a hétszegmenses kijelzőre kerülő adatot) az 5 bites számlálónk (cntr) MSB bitje határozta meg. Hiszen a CPLD felváltva várta az egyik majd a másik hétszegmenses kijelzőre kerülő adatot. Az alsó 4 bit elegendő volt a 16 kimenő adatbit számlálására.

A *cpld_clk*-t egy, az FPGA órajelére működőd, 12 bites szabadon futó felfelé számláló MSB bitjével állítottuk elő. Így, ahogy azt már említettük, kb. 12 kHz-es órajelet tudtunk biztosítani a CPLD-vel történő szinkron kommunikáció megvalósításához. Az interfész viselkedésének szimulációja a 4.2. ábrán látható.



4.2. Ábra – *A CPLD interfész hullámformája*

A modul RTL modellje a 7.3. szakaszban szerepel.

4.3. Az Egyedi periféria

Ahhoz, hogy a MicroBlaze processzor kommunikálni tudjon a CPLD-vel és az LCD kijelzővel, egy köztes, perifériaként működő modult kellett létrehoznunk, amely tartalmazta az előbb bemutatott két interfész áramkört, valamint additív logikát az IPIF illesztőtől érkező és az arra felé menő adatok kezelésére, és a felhasználói beavatkozás lévén elálló megszakítás kérést.

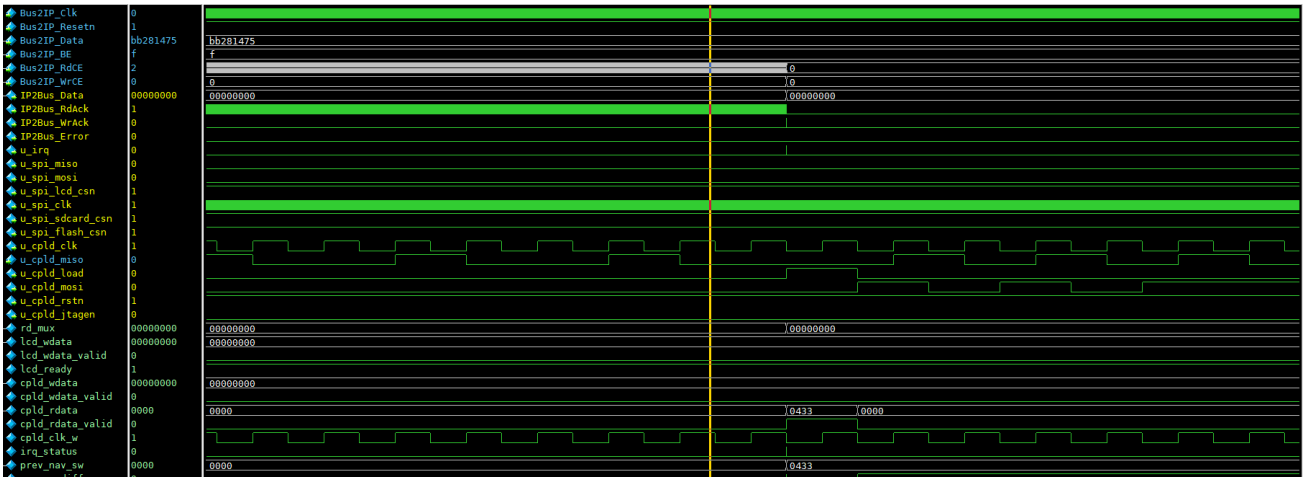
Az IPIF, e három fő egységet (LCD, CPLD és a megszakítás generálás), regiszter írással és olvasással tudta elérni. De persze az egyetlen, tényleges regiszter a megszakítás kérést generáló regiszter volt (irq_status), az LCD-nél és a CPLD-nél a megfelelő interfész-modulok megfelelő regiszterei lettek rákapcsolva az IPIP vezetékeire.

A modul regisztertérképe a 4.3. ábrán látható. Észrevehető, hogy az megszakítás generálásért felelős regiszter (irq_status) írásához Don't care-ek kerültek megadásra. Ez azért van, mert teljesen minden, hogy milyen adat érkezik erre a regiszterre, mivel a regiszter törléséhez (ami a megszakítás kérés nyugtázását jelenti) elegendő az, hogy az IPIF-től érkező regiszter írást jelző porton logikai 1 legyen.

	Cím	Írás	Olvasás
LCD	*BASE_ADDR + 0*	*0x{LCD_DATA_OR_CMD,0b000}000000{LCD_DATA[7:4]}{LCD_DATA[3:0]}*	*0x0000000{0b000,LCD_READY}*
CPLD	*BASE_ADDR + 4*	*0x0000{DISP2[15:12]}{DISP1[11:8]}{LED8[7:4]}{LED8[3:0]}*	*0x0000{0b00,RDATA_VALID,NAV_SW[12]}{NAV_SW[11:8]}{DIP8[7:4]}{DIP8[3:0]}*
INTR	*BASE_ADDR + 8*	*0x--------*	*0x0000000{0b000,IRQ_STATUS}*

4.3. Ábra – *Az Egyedi periféria (user_logic.v) regisztertérképe*

Az Egyedi perifériát modellező RTL modell szimulációja a 4.4. hullámformán látható. Itt egyszerre szerepelnek a CPLD és az SPI interfészek, valamint az IPIF illesztői jelei is. Az ábra elejétől a közepéig a Bus2IP_RdCE-n azért van rengeteg jelváltás, mert folyamatosan (az FPGA órajelével megegyező sebességgel) lekérdezzük az SPI interfészben lévő FSM állapotát.



4.4. Ábra – *Az Egyedi periféria hullámformája*

A modul RTL modellje a 7.4. szakaszban szerepel.

5. fejezet

A rendszeren futó szoftver

A megvalósított rendszeren futó szoftvert a Xilinx EDK fejlesztői környezet segítségével hoztuk létre, amely figyelembe vette az XPS-en megvalósított rendszer paramétereit.

Az egyszerű Snake játékunk C-ben került lekódolásra, a 2.1. folyamatábra alapján. Minden timer interrupt-ra mozdul egyet a kígyó a gombok által meghatározott irányba és a teste beíródik a 102×64-es pályára. Ezután ellenőrizzük, hogy a feje érinti-e a testét vagy elhagyta-e a pályát. Ha ezek a feltételek teljesülnek, akkor vagy csökken az életek száma vagy vége a játéknak.

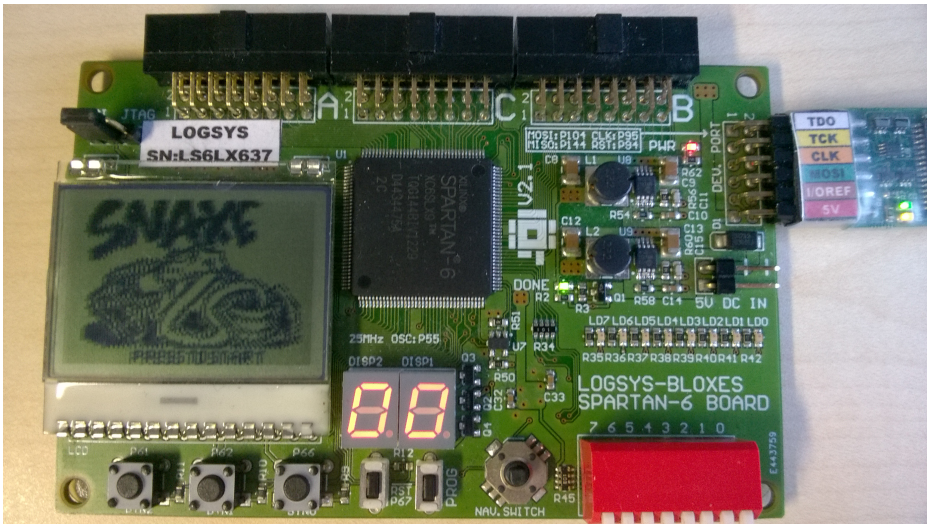
Ezután, ha nincs vége a játéknak, akkor ellenőrizzük, hogy a kígyó megette-e az almát, ha igen akkor a pontot, illetve a szintet a specifikációnak megfelelően növeljük. Ha elérjük a 8. szint 99-es pontját, akkor a játék befejeződik.

Ha a játék folytatódik, akkor kiírjuk az LCD-re a pályát, majd új, üres pályát hozunk létre a következő mozgatás előtt.

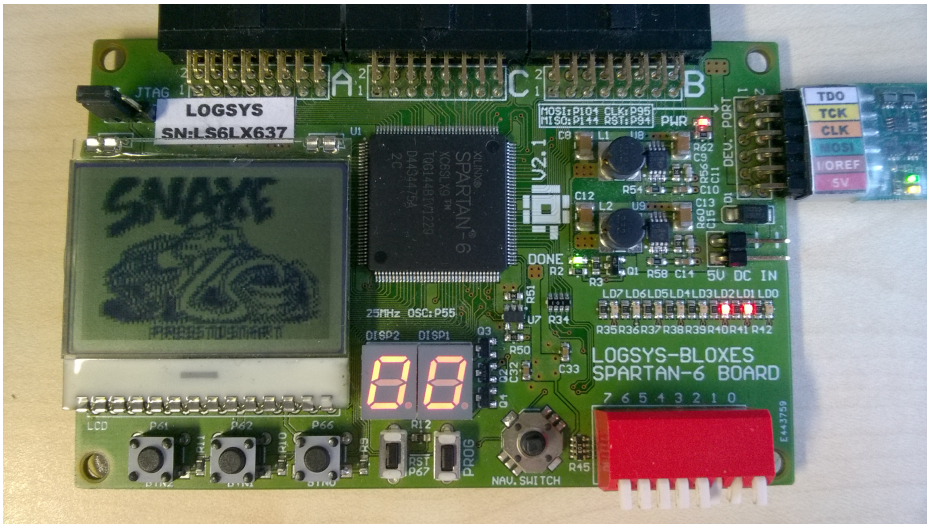
Minden navigációs gomb változásra ellenőrizzük, hogy az irányváltás lehetséges-e, ha igen, akkor megváltoztatjuk a fej mozgásának irányát. Ha a kapcsolók értéke változik, vagy benyomjuk a középső gombot, akkor az aktuális szint értéke a kapcsoló értékének megfelelően változik.

A szoftver forráskódja a 7.4. szakaszban szerepel.

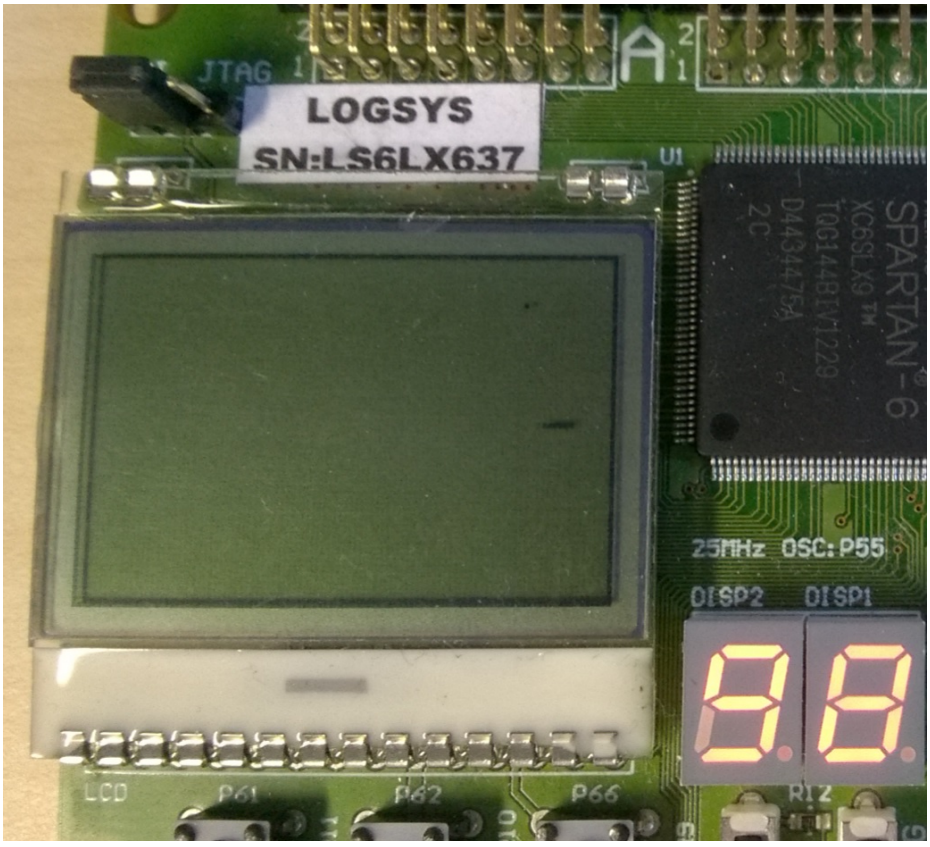
5.1. Néhány kép az FPGA-n kialakított beágyazott processzoros rendszeren futó szoftverről



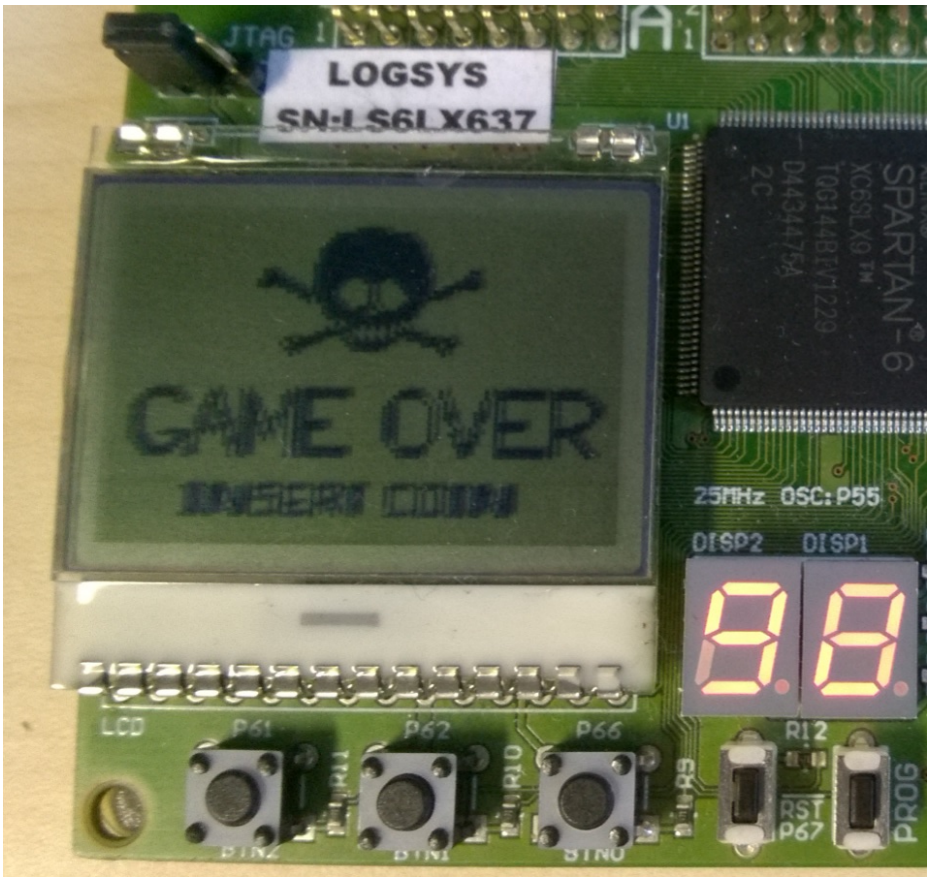
5.1. Ábra – *Az FPGA felkonfigurálása, valamint a szoftver feltöltése utáni állapotok.*



5.2. Ábra – *A DIP8 kapcsolóval a kezdeti nehézségi szint, bináris értékének beállítása lehetséges, valamit annak megjelenítése a LED-eken a történik.*



5.3. Ábra – *A Snake és a generált alma. A játék alapból 98 pontról indul, hogy gyorsan szemléltetni lehessen a 99 pont után történő szintlépést.*

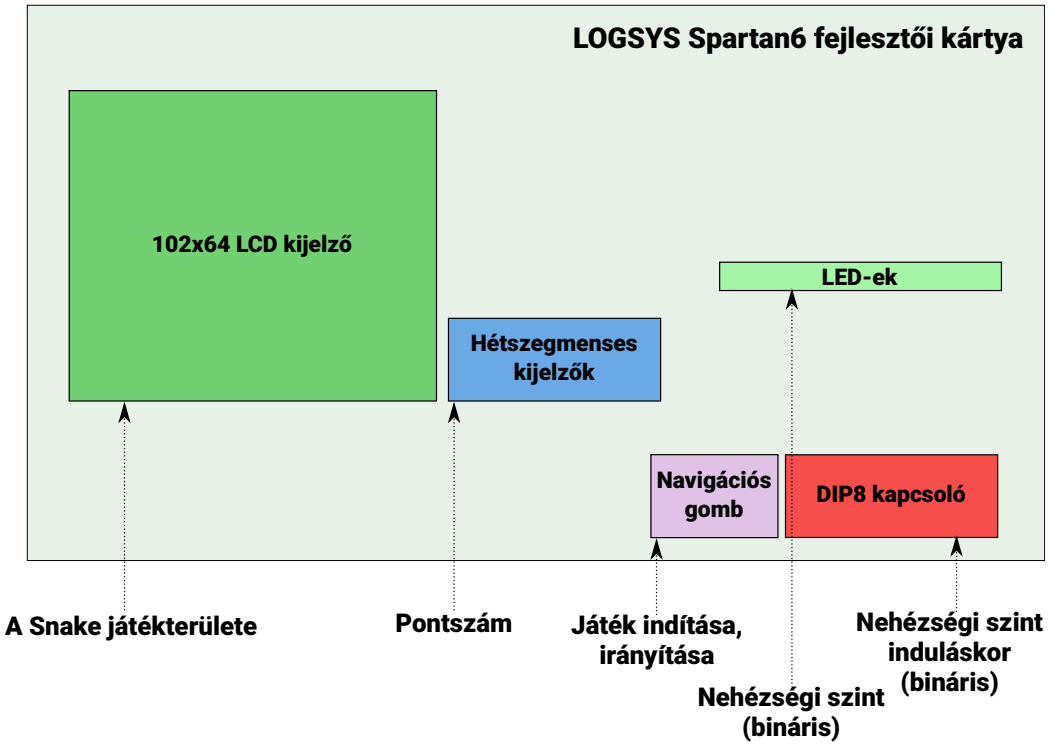


5.4. Ábra – *Alapból négy élete van a kigyónak, ezután a játék befejeződik, és az ábrán látható kép fogad. A játék újraindítása a P67-es nyomógomb segítségével tehető meg (ez a DISP2 kijelző alatt található).*

6. fejezet

Felhasználói útmutató

Az elkészült projektről egy rövid útmutató szeretnénk adni a felhasználáshoz. A 6.1. ábrán vázoltuk az útmutatóval kapcsolatos elemeket a fejlesztői kártyáról. Az ajánlott lépések a következők:



6.1. Ábra – *A fejlesztői kártya vázlatos képe a használat szempontjából*

1. Töltsük fel a projekthez tartozó konfigurációt az FPGA konfigurációs SRAM-jába.
2. Válasszuk ki a játék kezdeti nehézségi szintjét DIP8 kapcsolóval (a nehézség bináris értékét) és indítsuk el a játékot a Navigációs gomb lenyomásával (középső állásban kell benyomni). Ha a DIP8 kapcsolót nem állítjuk be, akkor a játék az 7-es (legnehezebb) nehézségi szintről fog indulni.
2. A játék elindult és a 102x64-es LCD kijelzőn tudjuk követni a játékban szereplő kígyó mozgását. Ennek irányítása a Navigációs gomb segítségével tehető meg.

7. fejezet

Függelék

7.1. Az SPI interfészt modellező Verilog modul (spi_if.v)

```
//=================================================================================================
// Description:     ~   This module implements a simple SPI-like interface for the LOGSYS
//                      SPARTAN6 FPGA's LCD module.
//
//                  ~   The module can accept new transfer, if the 'ready' output port is HIGH.
//                      Otherwise, it means that an going transfer is taking place by the inner FSM.
//
//                  ~   The module samples its input ports (data_or_cmd, din[7:0]) only, when
//                      the 'valid' input is asserted for one 'clk' period. This makes the new
//                      data for the LCD sampled, and it also makes the FSM to start its operation.
//
//=================================================================================================

module spi_lcd(
    input   wire            clk             ,   //  Clock.
    input   wire            rstn            ,   //  Active LOW reset.
    input   wire            valid           ,   //  Indicating that the 'data_or_cmd' and 'din[7:0]' can be sampled.
    input   wire            data_or_cmd     ,   //  Signaling towards the LCD the type of the payload.
    input   wire [7:0]      din             ,   //  The LCD payload.
    output  wire            ready           ,   //  During an ongoing transfer it is asserted (HIGH), and after it's completed it is de-asserted, meaning it can
          receive new transfer.
    output  wire            o_spi_miso      ,   //  This is an OUTPUT, the 'data_or_cmd' will be driven on this signal.
    output  wire            o_spi_mosi      ,   //  The serial line from the FPGA fabric to the LCD.
    output  wire            o_spi_lcd_csn   ,   //  Chip-select to the LCD:
    output  wire            o_spi_clk           //  The timing signal, "clock", for the serial line.
);


// ================  Internals BEGIN =====================
    reg             fe          ;   // Flop for falling edge detection.
    reg     [1:0]   state       ;   // State reg.
    reg     [5:0]   sck         ;   // From 50MHz about 0.7MHz "clock" (50MHz / 64 = ~0.7MHz).
    reg     [2:0]   data_cntr   ;   // How many bits have to be shifted out.
    reg     [7:0]   shr         ;   // Shift register for driving the 'o_spi_mosi' serial output line.

    // ==========  FSM control signals ==========
    reg             dec         ;   // The counter control signal, if it is asserted for one cycle, then the 'data_cntr' is decreased by 1.
    reg             spi_lcd_csn ;   // The chip-select control signal
    reg             shift       ;   // The shift register control signal, if it is asserted for one cycle, then the 'shr[7:0]' is shifted to the left by one.
                                    // Eventually, the shift reg. will be empty, because it's going to empty itself.
    reg             busy        ;   // The FSM is dealing with a transfer, and cannot accept new transfer, yet.
    // ==========  FSM control signals ==========

    reg             spi_miso    ;   // The 'data_or_cmd' buffering register, and its value will be updated by the FSM.

    wire            sck_fe      ;   // Combo logic for the falling-edge detector.
// ================  Internals END   =====================


// ================  Combinatorial BEGIN =====================
    assign          sck_fe          = (fe & ~sck[5])            ;   // Falling-edge on the 'sck[5]' signal.
    assign          ready           = ~busy                     ;   // Busy flag for indicating an ongoing transfer.
    assign          o_spi_miso      = spi_miso                  ;   // SPI MISO, in THIS case, this is an OUTPUT.
    assign          o_spi_mosi      = shr[7]                    ;   // SPI MOSI.
    assign          o_spi_lcd_csn   = spi_lcd_csn               ;   // Chip-select.
    assign          o_spi_clk       = sck[5]                    ;   // Driving the "clock" for the SPI IF.
// ================  Combinatorial END   =====================



//=========================================
    //SPI "clock" generating
    always @(posedge clk or negedge rstn)    begin
        if(~rstn)   begin
            sck <= 6'b000000;
        end
        else    begin
            sck <= sck + 6'b000001;             // Free running counter
        end
    end
//=========================================


//=========================================
    // For controlling the LCD (data or command)
    always @(posedge clk or negedge rstn)    begin
        if(~rstn)   begin
            spi_miso    <= 1'b0;
        end
        else    begin
            if(valid)
                spi_miso    <= data_or_cmd;
        end
    end
//=========================================
```

```
//=========================================
    // Flop chain for the falling edge detector
    always @(posedge clk or negedge rstn)    begin
        if(~rstn)   begin
            fe  <= 1'b0;
        end
        else    begin
            fe  <= sck[5];                      // Sampling the SPI clock
        end
    end
//=========================================


//=========================================
    //How much data is left in the shift reg?
    always @(posedge clk or negedge rstn)    begin
        if(~rstn)   begin
            data_cntr   <= 3'b111;
        end
        else    begin
            if(dec == 1'b1)    begin            // If one bit is shifted out
                data_cntr <= data_cntr - 3'b001;
            end
            if(state == 0)   begin              // In 'idle' state
                data_cntr <= 3'b111;
            end
        end
    end
//=========================================


//=========================================
    //Shift register
    always @(posedge clk)    begin
        if(~rstn)   begin
            shr <= 8'h0;
        end
        else    begin
            if(valid)  begin                        // If there is a new data to transmit
                shr <= din;
            end

            if(shift)   begin
                shr <= {shr[6:0],1'b0};         // Shifting into MSB
            end
        end
    end
//=========================================


//=========================================
    // FSM: 1 always  block, which means the control signals will be buffered (registered)!
    //
    // state:       0   => idle, waiting for an enable pulse
    //              1   => start, on sck falling edge
    //              2   => shifting data out

    always @(posedge clk or negedge rstn)  begin
        if(~rstn)   begin
            state   <= 0;                       // idle

            dec         <= 1'b0;
            spi_lcd_csn <= 1'b1;                //  Should be HIGH during reset state, lest it is indicating a transfer beginning
            shift       <= 1'b0;
            busy        <= 1'b0;
        end
        else    begin
            case(state)
            //=======================================
                0:  begin
                    dec         <= 1'b0;

                    spi_lcd_csn <= 1'b1;
                    shift       <= 1'b0;
                    busy        <= 1'b0;
                    if(valid)  begin
                        state       <= 1;       // start
                        busy        <= 1'b1;    // A transfer will take place.
                    end
                end
            //=======================================
                1:  begin
                    if(sck_fe)  begin
                        spi_lcd_csn <= 1'b0;    // Selecting the SLAVE.
                        state       <= 2;
                    end
                end
            //=======================================
                2:  begin                       // The MSB bit is already on the 'mosi'.
                    shift   <= 1'b0;
                    dec     <= 1'b0;

                    if(data_cntr != 3'b000) begin
                        if(sck_fe)  begin
                            shift   <= 1'b1;
                            dec     <= 1'b1;
                        end
                    end
                    else    begin              // If all the bits were shifted out.
                        if(sck_fe)  begin
                            spi_lcd_csn <= 1'b1;    // Back to high.
                            dec         <= 1'b0;
                            busy        <= 1'b0;    // No longer busy.
                            state       <= 0;       // The transfer has been completed.
                        end
                    end
                end
            //=======================================
                default:    begin
                    state   <= 2'bxx;           // Leaving some freedom for the synthesizer to optimize the FSM.
                    dec         <= 1'b0;

                    spi_lcd_csn <= 1'b1;
                    shift       <= 1'b0;
                    busy        <= 1'b0;
                end
            //=======================================
                endcase
        end
    end
//=========================================
endmodule
```

7.2. A CPLD interfészelését modellező Verilog modul (cpld_if.v)

```
module cpld_if(
   input               clk         ,
   input               rstn        ,
   input       [31:0]  din         ,   // [31:16] - Unused | [15:12] - DISP2 | [11:8] - DISP1 | [7:0] LED8
   input               din_valid   ,
   output      [12:0]  dout        ,
   output              dout_valid  ,
   output reg          cpld_clk    ,
   output reg          cpld_load   ,
   output reg          cpld_mosi   ,
   input               cpld_miso
);

//=============================================================
    reg     [11:0]  clk_div ;
    reg             clk_div_msb ;
    reg             ce          ;
    reg     [4:0]   cntr     ;
    reg     [7:0]   seg_data    ;
    reg     [15:0]  shr      ;
    reg     [15:0]  shr_in    ;
    reg     [15:0]  din_reg     ;

    reg     [15:0]  wdata_reg   ;

    wire            ld          ;
    wire    [3:0]   seg_mux     ;
//=============================================================


//=============================================================

    assign          dout    =   (cntr[3:0]==15) ? {shr_in[12:8],~shr_in[7:0]}: {13{1'b0}};
    assign          dout_valid= (cntr[3:0]==15) ;
    assign          ld      =   (cntr[3:0]==15) ;
    assign          seg_mux =   (cntr[4]) ? wdata_reg[15:12] : wdata_reg[11:8];
//=============================================================


//=============================================================
    always @(posedge clk or negedge rstn)
    begin
        if(~rstn)
            wdata_reg   <= {16{1'b0}};
        else if(din_valid)
            wdata_reg   <= {din[15:12],din[11:8],din[7:0]};
    end
//=============================================================


//=============================================================
    always  @ (posedge clk or negedge rstn)
    begin
        if(~rstn)
            clk_div <= {12{1'b0}};
        else
            clk_div <= clk_div + 1;
    end
//=============================================================


//=============================================================
    always  @ (posedge clk or negedge rstn)
    begin
        if(~rstn)   begin
            clk_div_msb <= 1'b0;
            ce          <= 1'b0;
        end
        else    begin
            clk_div_msb <= clk_div[11]              ;
            ce          <= clk_div_msb & ~clk_div[11];
        end
    end
//=============================================================



//=============================================================
    always  @ (posedge clk or negedge rstn)
    begin
        if(~rstn)   begin
            cntr <= {5{1'b0}};
        end
        else
            if (ce)
                cntr <= cntr + 1;
    end
//=============================================================


//=============================================================
    always  @(seg_mux)
        case (seg_mux)
            4'b0001 : seg_data = 8'b11111001;   // 1
            4'b0010 : seg_data = 8'b10100100;   // 2
            4'b0011 : seg_data = 8'b10110000;   // 3
            4'b0100 : seg_data = 8'b10011001;   // 4
            4'b0101 : seg_data = 8'b10010010;   // 5
            4'b0110 : seg_data = 8'b10000010;   // 6
            4'b0111 : seg_data = 8'b11111000;   // 7
            4'b1000 : seg_data = 8'b10000000;   // 8
            4'b1001 : seg_data = 8'b10010000;   // 9
            4'b1010 : seg_data = 8'b10001000;   // A
            4'b1011 : seg_data = 8'b10000011;   // b
            4'b1100 : seg_data = 8'b11000110;   // C
            4'b1101 : seg_data = 8'b10100001;   // d
            4'b1110 : seg_data = 8'b10000110;   // E
            4'b1111 : seg_data = 8'b10001110;   // F
            default : seg_data = 8'b11000000;   // 0
        endcase
//=============================================================


//=============================================================
    always @(posedge clk or negedge rstn)
    begin
        if(~rstn)   begin
            shr <= {16{1'b0}};
```

```
        end
        else    begin
            if (ce)
                if (ld)
                    shr <= {~seg_data, wdata_reg[7:0]};
                else
                    shr <= {1'b0, shr[15:1]};
        end
    end
//=============================================================


//=============================================================
    always @(posedge clk or negedge rstn)
    begin
        if(~rstn)   begin
           cpld_clk  <= 1'b0;
           cpld_load <= 1'b0;
           cpld_mosi <= 1'b0;
        end
        begin
            cpld_clk  <= clk_div[11];
            cpld_load <= ld         ;
            cpld_mosi <= shr[0]     ;
        end
    end
//=============================================================


//=============================================================
    always @(posedge clk or negedge rstn)
    begin
        if(~rstn)
            shr_in  <= {16{1'b0}};
        else
            if (ce)
                shr_in  <= {cpld_miso, shr_in[15:1]};
    end
//=============================================================


//=============================================================
    always @(posedge clk or negedge rstn)
    begin
        if(~rstn)   begin
            din_reg <= {16{1'b0}};
        end
        else    begin
            if(ce & ld)
                din_reg <= shr_in;
        end
    end
//=============================================================


endmodule
```

7.3. Az Egyedi perifériát modellező Verilog modul (user_logic.v)

```
//============================================================================================
// Description:
//
//              ~ LCD is at     (BASE_ADDR + 0)
//                  RD:     0x0000000{0b000,LCD_READY}
//                  WR:     0x{LCD_DATA_OR_CMD,0b000}00000{LCD_DATA[7:4]}{LCD_DATA[3:0]}
//
//              ~ CPLD is at    (BASE_ADDR + 4)
//                  RD:     0x0000{0b00,RDATA_VALID,NAV_SW[12]}{NAV_SW[11:8]}{DIP8[7:4]}{DIP8[3:0]}
//                  WR:     0x0000{DISP2[15:12]}{DISP1[11:8]}{LED8[7:4]}{LED8[3:0]}
//
//              ~ INTR is at    (BASE_ADDR + 8)
//                  RD:     0x0000000{0b000,IRQ_STATUS}
//                  WR:     0x-------- : The wdata can be anything, it only needs a write to this address to clear the interrupt register
//
//============================================================================================
`uselib lib=unisims_ver
`uselib lib=proc_common_v3_00_a


module user_logic #(
    parameter                               C_NUM_REG    = 3,           //   Number of registers
    parameter                               C_SLV_DWIDTH = 32           //   Data width
)(
    input   wire                            Bus2IP_Clk      ,           //   Clock
    input   wire                            Bus2IP_Resetn   ,           //   Active LOW reset
    input   wire    [C_SLV_DWIDTH-1:0]      Bus2IP_Data     ,           //   Read data bus
    input   wire    [C_SLV_DWIDTH/8-1:0]    Bus2IP_BE       ,       //   Byte enables (only valid at writing).
    input   wire    [C_NUM_REG-1:0]         Bus2IP_RdCE     ,           //   Reading CE for registers (OHE-HOT vectors)
    input   wire    [C_NUM_REG-1:0]         Bus2IP_WrCE     ,           //   Writing CE for registers (OHE-HOT vectors)
    output  wire    [C_SLV_DWIDTH-1:0]      IP2Bus_Data     ,           //   Write data bus
    output  wire                            IP2Bus_RdAck    ,           //   Reading ack.
    output  wire                            IP2Bus_WrAck    ,           //   Writing ack
    output  wire                            IP2Bus_Error    ,           //   Error signaling
    output      wire                                        u_irq           ,       //   The only interrupt line
    output  wire                            u_spi_miso      ,       //**
    output  wire                            u_spi_mosi      ,       //** IF for the LCD
    output  wire                            u_spi_lcd_csn   ,       //**
    output  wire                            u_spi_clk       ,       //**
    output  wire                            u_spi_sdcard_csn,       //** Unused: Driven HIGH
    output  wire                            u_spi_flash_csn ,       //** Unused: Driven HIGH
    output  wire                            u_cpld_clk      ,       //++
    input   wire                            u_cpld_miso     ,       //++ IF for the CPLD
    output  wire                            u_cpld_load     ,       //++
    output  wire                            u_cpld_mosi     ,       //++
    output  wire                            u_cpld_rstn     ,       //++
    output  wire                            u_cpld_jtagen           //++ Should be driven LOW!
);

// ================  Internals BEGIN ======================
    wire    [31:0]          rd_mux          ;

    wire    [31:0]          lcd_wdata       ;                       //Write data for the LCD
    wire                    lcd_wdata_valid ;                       //Valid signal
    wire                    lcd_ready       ;                       //Ready signal

    wire    [31:0]          cpld_wdata      ;                       //Write data for the CPLD (LED8+7SEGs)
    wire                    cpld_wdata_valid;
    wire    [12:0]          cpld_rdata      ;
```

```
    wire                    cpld_rdata_valid;
        wire                                    cpld_clk_w              ;


    reg                     irq_status      ;
    reg     [12:0]           prev_nav_sw     ;
    wire                    nav_sw_differs  ;
// ================  Internals END   =====================


// ================  Combinatorial BEGIN =====================
    assign      IP2Bus_Data     =   rd_mux;
    assign      IP2Bus_WrAck    =   |(Bus2IP_WrCE);
    assign      IP2Bus_RdAck    =   |(Bus2IP_RdCE);
    assign      IP2Bus_Error    =   1'b0;

    assign      u_cpld_jtagen   =   1'b0;
    assign      u_cpld_rstn     =   Bus2IP_Resetn;
        assign          u_cpld_clk              =       cpld_clk_w;

    assign      u_spi_sdcard_csn =   1'b1;
    assign      u_spi_flash_csn  =   1'b1;

    assign      u_irq           =   irq_status;
    assign      nav_sw_differs  =   |(prev_nav_sw ^ cpld_rdata[12:0]);                          // Are they any different?


    //=============================================================================================
    assign      lcd_wdata       =   (Bus2IP_BE == 4'hf && Bus2IP_WrCE == 3'b100) ? Bus2IP_Data   :   //base + 0 (LCD)
                                    32'h00000000;
    assign      lcd_wdata_valid =   (Bus2IP_BE == 4'hf && Bus2IP_WrCE == 3'b100) ?   1'b1: 1'b0  ;
    //=============================================================================================
    assign      cpld_wdata      =   (Bus2IP_BE == 4'hf && Bus2IP_WrCE == 3'b010) ? Bus2IP_Data   :   //base + 4 (CPLD)
                                    32'h00000000;
    assign      cpld_wdata_valid=   (Bus2IP_BE == 4'hf && Bus2IP_WrCE == 3'b010) ?   1'b1: 1'b0  ;
    //=============================================================================================


    //=============================================================================================
    assign      rd_mux          =       (Bus2IP_RdCE == 3'b100) ? {(Bus2IP_RdCE == 3'b100) ? {31{1'b0}} ,  lcd_ready                     }    :   //base + 0 (LCD)
                                    (Bus2IP_RdCE == 3'b010) ? {(Bus2IP_RdCE == 3'b010) ? {18{1'b0}} ,  cpld_rdata_valid, cpld_rdata  }    :   //base + 4 (CPLD)
                                    (Bus2IP_RdCE == 3'b001) ? {(Bus2IP_RdCE == 3'b001) ? {31{1'b0}} ,  irq_status                    }    :   //base + 8 (INTR)
                                    32'h00000000;
    //=============================================================================================


// ================  Combinatorial END   =====================



//=========================================
    //base + 8 (INTR)
    always  @(posedge Bus2IP_Clk or negedge Bus2IP_Resetn)    begin
        if(~Bus2IP_Resetn)
            irq_status  <= 1'b0;
        else
            if(cpld_rdata_valid && nav_sw_differs)              // Setting the interrupt line
                irq_status  <= 1'b1;
            else
                if(Bus2IP_BE == 4'hf && Bus2IP_WrCE == 3'b001)  // Clearing the interrupt status
                    irq_status  <= 1'b0;
    end
//=========================================

//=========================================
    always  @(posedge Bus2IP_Clk or negedge Bus2IP_Resetn)    begin
        if(~Bus2IP_Resetn)  begin
            prev_nav_sw <=  12'b000000000000;
        end
        else    begin
            if(cpld_rdata_valid)
                prev_nav_sw <= cpld_rdata[12:0];
        end
    end
//=========================================


// ================  Instantiations BEGIN =====================
    // CPLD is at     (BASE_ADDR + 4)
    cpld_if i_cpld_if(
        .clk            (Bus2IP_Clk         ),      //input
        .rstn           (Bus2IP_Resetn      ),      //input
        .din_valid      (cpld_wdata_valid   ),      //input
        .din            (cpld_wdata         ),      //input
        .dout           (cpld_rdata         ),      //output
        .dout_valid     (cpld_rdata_valid   ),      //output
        .cpld_clk       (cpld_clk_w         ),      //output
        .cpld_miso      (u_cpld_miso        ),      //input
        .cpld_load      (u_cpld_load        ),      //output
        .cpld_mosi      (u_cpld_mosi        )       //output
    );


    // LCD is at      (BASE_ADDR + 0)
    spi_lcd i_spi_lcd(
        .clk            (Bus2IP_Clk         ),      //input
        .rstn           (Bus2IP_Resetn      ),      //input
        .valid          (lcd_wdata_valid    ),      //input
        .data_or_cmd    (lcd_wdata[31]      ),      //input
        .din            (lcd_wdata[7:0]     ),      //input
        .ready          (lcd_ready          ),      //output
        .o_spi_miso     (u_spi_miso         ),      //output
        .o_spi_mosi     (u_spi_mosi         ),      //output
        .o_spi_lcd_csn  (u_spi_lcd_csn      ),      //output
        .o_spi_clk      (u_spi_clk          )       //output
    );
// ================  Instantiations END   =====================

endmodule
```

7.4. A játék algoritmusát megvalósító C kód (main.c)

```c
#include <stdio.h>
#include <xparameters.h>    // The global hardware definer
#include <xintc_l.h>        // Interrupt related defines
#include <mb_interface.h>   // MicroBlaze interface

#include <xtmrctr_l.h>
#include <xtmrctr.h>
#include <time.h>
#include <stdlib.h>
#include "header.h"



//========= Macros for memory writes and reads =========

    #define MEM8(addr)   (*(volatile unsigned char*     )(addr))
    #define MEM16(addr)  (*(volatile unsigned short*    )(addr))
    #define MEM32(addr)  (*(volatile unsigned long*     )(addr))

//======================================================




//========= Registers in 'snake_0' =====================

    #define LCD_REG     0x00
    #define CPLD_REG    0x04
    #define INTR_REG    0x08

//======================================================

//======================================================

    #define UP 0x01
    #define DOWN 0x02
    #define LEFT 0x04
    #define RIGHT 0x08
    #define CENT 0x10

//======================================================


//======================================================

    #define NAV_SW_IRQ  (1 << 0)                                // The NAV_SW's value is changed, and this caused an interrupt

//======================================================


//================== Global variables ==================

    volatile unsigned   char    user_points         =   0x00;   //  User points
    volatile unsigned   char    user_difficulty     =   0x01;   //  User difficulty, by default it is 1.
    volatile unsigned   char    difficulty_to_led[] = {         //  The proper difficulty value for the LEDS (one-hot vector)
            0x01,
            0x02,
            0x04,
            0x08,
            0x10,
            0x20,
            0x40,
            0x80
        };

    volatile unsigned char to_lcd[102][8];                      //  LCD data separated in pages
//======================================================


//======================================================
    int length;                                                 //  Length of the snake
    unsigned char dir;                                          //  Direction of the head of the snake
    unsigned char prev_dir;                                     //  Previous direction of the head of the snake
    int life;                                                   //  Number of extra lives
    unsigned char LCD[102][64];                                 //  Array of the map
    unsigned char score;                                        //  Number of apples eaten
    unsigned char level;                                        //  Difficulty level
    unsigned char win=0;                                        //  Check if the user has won the game

    void WritePixel(int x,int y, char pixel);                   //  Putting pixels into the 102*64 LCD array
    void InitializeBody();                                      //  If starting the game or the snake died, initialize snake body
    void Move();                                                //  Moving the snake
    void ChangeDirection();                                     //  Check if the direction has to change
    void Apple();                                               //  Generating and eating apple
    void MakeMap();                                             //  Making empty map
    void CheckDeath();                                          //  Check if the snake died
    void WriteScreen();                                         //  Converting the 102*64 array into 8 pages for LCD

    struct coordinate{
        int x;
        int y;
        int direction;
    };

    typedef struct coordinate coordinate;                       //  Structure for the snake body and apple

    coordinate apple,body[105];


    void nav_sw_int_handler(void *instancePtr);                 //  Interrupt handler of the navigation buttons and switches
    void timer_int_handler(void *instancePtr);                  //  Interrupt handler of the timer

    void lcd_init();                                            //  Sending the initialization command to the LCD
    void write_lcd();                                           //  Writing the screen to the LCD
    void read_input();                                          //  Read input from the navigation buttons and switches
    void write_score_and_level();                               //  Writing the score and level to the 7 segment displays and the LEDs

    void start();                                               //  Write to LCD, wait for button
    void gameover();                                            //  Decide if you win or game over, write to LCD
    void write_start();                                         //  Write start image to the LCD
    void write_gameover();                                      //  Write game over image to the LCD
    void write_win();                                           //  Write you win image to the LCD
//======================================================

//======================================================
    void WritePixel(int x, int y, char pixel)
    {
        LCD[x][y]=pixel;                                        //Write pixel to the array
```

```
    }
//================================================

//================================================
    void InitializeBody()
    {
        body[0].x   =   51;                 //Initial coordinates and direction
        body[0].y   =   31;
        body[0].direction   =   RIGHT;

        int i;
        for(i=0; i<length; i++)         //Every part of the body moves the same way
        {
            body[i].direction=body[0].direction;
        }

        for(i=1; i<length; i++)         //Making the body
        {
            body[i].x   =   body[i-1].x-1;
            body[i].y   =   body[i-1].y;
        }
    }
//================================================

//================================================================
    void Move()
    {
        int i;

        if(body[0].direction    ==  RIGHT)   //Moving the head according to the direction
        {
            body[0].x++;
            WritePixel(body[0].x,body[0].y,1);
        }

        if(body[0].direction    ==  LEFT)
        {
            body[0].x--;
            WritePixel(body[0].x,body[0].y,1);
        }

        if(body[0].direction    ==  UP)
        {
            body[0].y--;
            WritePixel(body[0].x,body[0].y,1);
        }

        if(body[0].direction    ==  DOWN)
        {
            body[0].y++;
            WritePixel(body[0].x,body[0].y,1);
        }

        for(i=length-1; i>0; i--)      //Every part of the body follows the direction of the previous part
        {
            if(body[i].direction == RIGHT)
                body[i].x++;

            if(body[i].direction == LEFT)
                body[i].x--;

            if(body[i].direction == UP)
                body[i].y--;

            if(body[i].direction == DOWN)
                body[i].y++;

            body[i].direction   =   body[i-1].direction;
        }

        for(i=1; i<length; i++)             //Writing the body to the array
            WritePixel(body[i].x,body[i].y,1);

        Apple();

        if(win == 0)                    //If the user didn't win the game
            WriteScreen();              //Write the map to the screen

        CheckDeath();                   //Check if the snake dies

        MakeMap();                      //Make empty map

    }
//================================================================

//================================================================
    void ChangeDirection()
    {
        //Change the direction according the button pressed
        if (    (dir == RIGHT   &&  body[0].direction != LEFT   &&  body[0].direction != RIGHT  )   ||
                (dir == LEFT    &&  body[0].direction != RIGHT  &&  body[0].direction != LEFT   )   ||
                (dir == UP      &&  body[0].direction != DOWN   &&  body[0].direction != UP     )   ||
                (dir == DOWN    &&  body[0].direction != UP     &&  body[0].direction != DOWN   )
           )
        {
            body[0].direction = dir;
        }
    }
//================================================================

//================================================================
    void Apple()
    {
        if(body[0].x == apple.x && body[0].y == apple.y)    //If the snake ate the apple, attach a new body part
        {
            body[length].direction  =   body[length-1].direction;

            if(body[length].direction   ==  UP)
            {
                body[length].x  = body[length-1].x;
                body[length].y  = body[length-1].y+1;
            }

            if(body[length].direction   ==  DOWN)
            {
                body[length].x  = body[length-1].x;
                body[length].y  = body[length-1].y-1;
            }

            if(body[length].direction   ==  RIGHT)
            {
                body[length].x  =   body[length-1].x-1;
```

```
            body[length].y  =   body[length-1].y;
        }

        if(body[length].direction   ==  LEFT)
        {
            body[length].x  =   body[length-1].x+1;
            body[length].y  =   body[length-1].y;
        }
        if(score < 99)
        {
            length++;
            score++;
        }
        else              //If the score is over 99, and the level is under 8, add score
        {
            score   = 0;
            length  = 5;

            if(level < 7)
                level++;
            else          //If the score is over 99 and the level is over 9, the user has won the game
            {
                win = 1;
                gameover();
                //level=0;
            }
        }

        apple.x =   rand() % 101;   //Generating new apple

        if(apple.x == 0)
            apple.x = 1;

        apple.y =   rand() % 63;

        if(apple.y == 0)
            apple.y = 1;
    }
    else if(apple.x == 0)       //Creating apple for the first time because global variable are initialized with 0
    {
        apple.x = rand() % 101;
        if(apple.x <= 0)
            apple.x += 1;

        apple.y = rand() % 63;
        if(apple.y <= 0)
            apple.y += 1;
    }
    WritePixel(apple.x,apple.y,1);  //placing apple
}
//=================================================================

//=================================================================
void MakeMap()
{
    int row,col,i;

    for(row = 0; row<64; row++)
    {
        for(col=0; col<102; col++)   //Making empty map
        {
             LCD[col][row] = 0;
        }
    }
    for(i=0; i<102; i++)             //Writing borders
    {
       WritePixel(i,0,1);
       WritePixel(i,63,1);
    }
    for(i=0; i<64; i++)
    {
        WritePixel(0,i,1);
        WritePixel(101,i,1);
    }
}
//=================================================================

//=================================================================
void CheckDeath()
{
    int i,check=0;
    for(i = 1 ; i < length ; i++)
    {
        if(body[0].x == body[i].x  &&  body[0].y == body[i].y)
        {
            check++;                //The head hit the body
        }
        if(check != 0)
            break;
    }
    if( body[0].x <= 0  ||
        body[0].x >= 101||
        body[0].y <= 1  ||
        body[0].y >= 63 ||
        check != 0)
    {
        life--;
        if(life >= 0)           //If there is more extra life, start again
        {
            InitializeBody();
            Move();
        }
        else
        {
            gameover();
        }
    }
}
//=================================================================


//=================================================================
void WriteScreen()
{

    int page, col;
    for(page = 0; page<8; page++)
    {
        for(col=0; col<102; col++)      //Dividing the map into pages for the LCD
        {
            to_lcd[col][page] = ((LCD[col][page*8 + 7]) << 7)    |
                                ((LCD[col][page*8 + 6]) << 6)    |
```

```
                                    ((LCD[col][page*8 + 5]) << 5)   |
                                    ((LCD[col][page*8 + 4]) << 4)   |
                                    ((LCD[col][page*8 + 3]) << 3)   |
                                    ((LCD[col][page*8 + 2]) << 2)   |
                                    ((LCD[col][page*8 + 1]) << 1)   |
                                    ((LCD[col][page*8 + 0])
                                    );
            }
        }
        write_lcd();
    }
//=================================================================


    //======= NAV_SW generated interrupt routine ===========
    void nav_sw_int_handler(void *instancePtr)
    {
        unsigned long   irq_status;          // The INTR_REG value

        irq_status = MEM32(XPAR_SNAKE_0_BASEADDR  + INTR_REG);  // Reading out
        MEM32(XPAR_SNAKE_0_BASEADDR  + INTR_REG) = irq_status;  // Acknowledging, clearing the status, simply writing back the read out value of it
        //=============================================
        //=============================================
        //=============================================
        read_input();                                           //Read which button or switch was pressed
        ChangeDirection();                                      //Check if changing the direction is necessary
        write_score_and_level();                                //Write score and level to the 7 segment displays and LEDs
    }




    //======= Timer generated interrupt routine ===========
    void timer_int_handler(void *instancePtr)
    {
        unsigned long csr;

        write_score_and_level();
        Move();

        //For every interrupt, update the timer interval for interrupt
        XTmrCtr_SetLoadReg(XPAR_AXI_TIMER_0_BASEADDR, 0, (XPAR_PROC_BUS_0_FREQ_HZ/(level+8)));
        //A megszakitas jelzes torlese.
        csr = XTmrCtr_GetControlStatusReg(XPAR_AXI_TIMER_0_BASEADDR, 0);
        XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_0_BASEADDR, 0, csr);
    }

//=============================================================================================

//========== Initialize LCD ============================
    void lcd_init()
    {
        unsigned long lcd_data              = 0x00000000;
        unsigned char lcd_init_commands[]   ={        // The array of the initializing commands
            0x40,
            0xA0,
            0xC8,
            0xA4,
            0xA6,
            0xA2,
            0x2F,
            0x27,
            0x81,
            0x10,
            0xFA,
            0x90,
            0xAF,
            0x0E,   // Column address 0x1E
            0x11,   // Column address 0x1E
            0xB0,   // Page addres: 0.
        };
        int i;
        for( i=0 ; i<16 ; i++ )       // Going through the commands
        {
            while(MEM32(XPAR_SNAKE_0_BASEADDR  + LCD_REG) == 0) {} //  Wait while the LCD is NOT ready to recieve the data.
            MEM32(XPAR_SNAKE_0_BASEADDR  + LCD_REG) = ( lcd_data | lcd_init_commands[i] ); // Writing in one of the commands.
        }
    }
//=================================================================




//=================================================================
    void write_lcd()
    {
        unsigned long lcd_data              = 0x00000000;
        unsigned long page_cmd              = 0x000000B0;

        unsigned long page;
        unsigned long col;

        for(page=0; page<8; page++)
        {
            while(MEM32(XPAR_SNAKE_0_BASEADDR  + LCD_REG) == 0) {}
            MEM32(XPAR_SNAKE_0_BASEADDR  + LCD_REG) = ( lcd_data | 0x0E );          //Setting the column number to the left side of the LCD

            while(MEM32(XPAR_SNAKE_0_BASEADDR  + LCD_REG) == 0) {}
            MEM32(XPAR_SNAKE_0_BASEADDR  + LCD_REG) = ( lcd_data | 0x11 );


            while(MEM32(XPAR_SNAKE_0_BASEADDR  + LCD_REG) == 0) {}
            MEM32(XPAR_SNAKE_0_BASEADDR  + LCD_REG) = ( lcd_data | page_cmd | page );       //Setting the number of the page to be written

            for(col=0; col<102; col++)
            {
                while(MEM32(XPAR_SNAKE_0_BASEADDR  + LCD_REG) == 0) {}
                MEM32(XPAR_SNAKE_0_BASEADDR  + LCD_REG) = ( 0x80000000 | to_lcd[col][page] );   //Writing to LCD
            }
        }
    }
//=================================================================


//=================================================================
    void read_input()                                                                   //If an interrupt was received from the buttons or switches, read the
           values
    {
        while((MEM32(XPAR_SNAKE_0_BASEADDR  + CPLD_REG)&0x00002000) == 0) {}
```

```
        prev_dir = dir;                                                         //Previous value of the direction, to check if we should update the level
               or not

        dir =   ((0x00001F00)   &   MEM32(XPAR_SNAKE_0_BASEADDR  + CPLD_REG))   >> 8;   //Reading the value of the navigation button

        if( (dir == 0 && prev_dir == 0) || (dir == 16)  )                       //Check if the interrupt was received, because the value of the switches
              has changed, or we want to update it via the center button
        {
            if( ((0x000000FF) & MEM32(XPAR_SNAKE_0_BASEADDR  + CPLD_REG)) < 8)          //The level must be under 8
            {
                level =( (0x000000FF) & MEM32(XPAR_SNAKE_0_BASEADDR  + CPLD_REG) );
            }
            else
            {
                level = 7;
            }
        }
    }
//=====================================================================



//======== Print points and difficulty to the LED ======================
    void write_score_and_level()                    // Simultaneously writing the User's POINT and the DIFFICULTY
    {
        unsigned long data_to_cpld = 0x00000000;
        char score_ones;
        char score_tens;
        unsigned char level_onehot[] = {
            0x01,
            0x02,
            0x04,
            0x08,
            0x10,
            0x20,
            0x40,
            0x80
        };



        int ii = (int) level;







        score_tens  =   (score / 10) & 0xf;       //Converting to BCD
        score_ones  =   (score-score_tens*10) & 0xf;

        data_to_cpld = ((score_tens << 12) | (score_ones << 8) | level); //Writing level and score



        MEM32(XPAR_SNAKE_0_BASEADDR  + CPLD_REG) = data_to_cpld;        // Writing out the points and the difficulty, as well

    }
//=====================================================================

//=====================================================================
    void start()
    {
        write_start();        //Writing the start image to the LCD
        while(dir != 16) {}   //Waiting for center button hit
    }
//=====================================================================



//=====================================================================
    void gameover()
    {
        XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_0_BASEADDR,0,(!XTC_CSR_ENABLE_TMR_MASK )  |
                                                                 (!XTC_CSR_ENABLE_INT_MASK )  |
                                                                 (!XTC_CSR_AUTO_RELOAD_MASK)  |
                                                                   XTC_CSR_DOWN_COUNT_MASK )  ; //If the game is over, we have to stop the counter
        if(win  ==  1)                  //If the user won
            write_win();                //Write you win image
        else
            write_gameover();           //Else write game over image
    }
//=====================================================================



//=====================================================================
    void write_start()
    {
        int page, col;
        for(page = 0; page<8; page++)
        {
            for(col=0; col<102; col++) //Writing start image
            {
                to_lcd[col][page]   =   (   ( ((Start_Image[col+(page*8 + 7)*102]) & (0x01)) << 7) |
                                            ( ((Start_Image[col+(page*8 + 6)*102]) & (0x01)) << 6) |
                                            ( ((Start_Image[col+(page*8 + 5)*102]) & (0x01)) << 5) |
                                            ( ((Start_Image[col+(page*8 + 4)*102]) & (0x01)) << 4) |
                                            ( ((Start_Image[col+(page*8 + 3)*102]) & (0x01)) << 3) |
                                            ( ((Start_Image[col+(page*8 + 2)*102]) & (0x01)) << 2) |
                                            ( ((Start_Image[col+(page*8 + 1)*102]) & (0x01)) << 1) |
                                            ( ((Start_Image[col+(page*8    )*102]) & (0x01))     )
                                        );
            }
        }
        write_lcd();
    }
//=====================================================================


//=====================================================================
    void write_gameover()
    {
        int page, col;
        for(page = 0; page<8; page++)
        {
            for(col=0; col<102; col++)  //Writing game over image
            {
```

```
                to_lcd[col][page] = (   ( ((GameOver_Image[col+(page*8 + 7)*102]) &   (0x01)) << 7)  |
                                        ( ((GameOver_Image[col+(page*8 + 6)*102]) &   (0x01)) << 6)  |
                                        ( ((GameOver_Image[col+(page*8 + 5)*102]) &   (0x01)) << 5)  |
                                        ( ((GameOver_Image[col+(page*8 + 4)*102]) &   (0x01)) << 4)  |
                                        ( ((GameOver_Image[col+(page*8 + 3)*102]) &   (0x01)) << 3)  |
                                        ( ((GameOver_Image[col+(page*8 + 2)*102]) &   (0x01)) << 2)  |
                                        ( ((GameOver_Image[col+(page*8 + 1)*102]) &   (0x01)) << 1)  |
                                        ( ((GameOver_Image[col+(page*8    )*102]) &   (0x01))     )
                                    );
            }
        }
        write_lcd();
    }
//=================================================================


//=================================================================
    void write_win()                            //Writing you win image
    {
        int page, col;
        for(page=0; page<8; page++)
        {
            for(col=0; col<102; col++)
            {
                to_lcd[col][page] = (   ( ((Win_Image[col+(page*8 + 7)*102])  &   (0x01)) << 7)   |
                                        ( ((Win_Image[col+(page*8 + 6)*102])  &   (0x01)) << 6)   |
                                        ( ((Win_Image[col+(page*8 + 5)*102])  &   (0x01)) << 5)   |
                                        ( ((Win_Image[col+(page*8 + 4)*102])  &   (0x01)) << 4)   |
                                        ( ((Win_Image[col+(page*8 + 3)*102])  &   (0x01)) << 3)   |
                                        ( ((Win_Image[col+(page*8 + 2)*102])  &   (0x01)) << 2)   |
                                        ( ((Win_Image[col+(page*8 + 1)*102])  &   (0x01)) << 1)   |
                                        ( ((Win_Image[col+(page*8    )*102])  &   (0x01))     )
                                    );
            }
        }
        write_lcd();
    }
//=================================================================


int main()
{

//=============================================================================
// Registering the 'nav_sw_int_handler' function as an interrupt service routine (ISR).
// The ISR will be called when an interrupt occurs on the.

    XIntc_RegisterHandler(
        XPAR_INTC_0_BASEADDR                    ,       // Base address of 'AXI_INTC_0' interrupt controller peripheral, whose vector table will be modified.
        XPAR_AXI_INTC_0_SNAKE_0_IRQ_INTR        ,       // Interrupt ID on the MicroBlaze's portlist
        (XInterruptHandler) nav_sw_int_handler  ,       // The name of the routine (nav_sw_int_handler), a function pointer that will be added for the Interrupt ID
        NULL                                            // No additional parameters will be passed for the 'nav_sw_int_handler'
    );

    XIntc_RegisterHandler(
    XPAR_INTC_0_BASEADDR   , //INTC baziscime
    XPAR_AXI_INTC_0_AXI_TIMER_0_INTERRUPT_INTR, //Megszakitas azonosito
    (XInterruptHandler) timer_int_handler, //Megszakitaskezelo rutin
    NULL //Megsz. kezelo rutin parametere
    );



//=============================================================================
// Enables all the interrupts in the 'AXI_INTC_0' interrupt controller.

    XIntc_MasterEnable(XPAR_INTC_0_BASEADDR);


//=============================================================================
// Individual interrupts

    XIntc_EnableIntr(
        XPAR_INTC_0_BASEADDR ,(XPAR_SNAKE_0_IRQ_MASK | XPAR_AXI_TIMER_0_INTERRUPT_MASK)
    );

//=================================================
// Enabling the interrupts on the MicroBlaze processor

    microblaze_enable_interrupts();



//=================================================
    level = 7   ;       //Initial level is 8 for testing
    lcd_init()  ;       //Initializing LCD
    start()     ;       //Writing start image and waiting for button
    write_score_and_level();


    //=============================================================
    //A timer LOAD regiszterenek beallitasa (megszakitas masodpercenkent/(szint+8))
    XTmrCtr_SetLoadReg(XPAR_AXI_TIMER_0_BASEADDR, 0, (XPAR_PROC_BUS_0_FREQ_HZ / (level + 8)));

    //A timer alapallapotba allitasa.
    XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_0_BASEADDR,0,XTC_CSR_INT_OCCURED_MASK | XTC_CSR_LOAD_MASK);

    //A timer elinditasa.
    XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_0_BASEADDR,0,XTC_CSR_ENABLE_TMR_MASK | XTC_CSR_ENABLE_INT_MASK |XTC_CSR_AUTO_RELOAD_MASK | XTC_CSR_DOWN_COUNT_MASK);
    //=============================================================

    length  =   5   ;       //Initial length is 5
    score   =   98  ;       //Initial score is 98 for testing
    life    =   3   ;       //Initial number of extra lives is 3

    InitializeBody();   //Initializing body

    MakeMap();          //Making empty map

    Apple();            //Generating apple coordinates initially

    while(1) {}

    return 0;
}
```