Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Snake megvalósítása Logsys Spartan 6 fejlesztőpanelen

Mikrorendszerek házi feladat

Készítette

Gránicz Attila - GQEKMH

Tóth Tibor - HH6PXT

2017. december 13.

Tartalomjegyzék

Feladat

A feladat egy snake játék megvalósítása Logsys Spartan6 fejlesztői kártyán. A játékot a grafikus LCD kijelzőn kell megjeleníteni, a kezelés a jostick-al, illetve a nyomógombokkal történik. A pontszám a megjelenítésére a hétszegmenses kijelző használandó. A játék szintjének néhány pont szerzése után emelkednie kell.

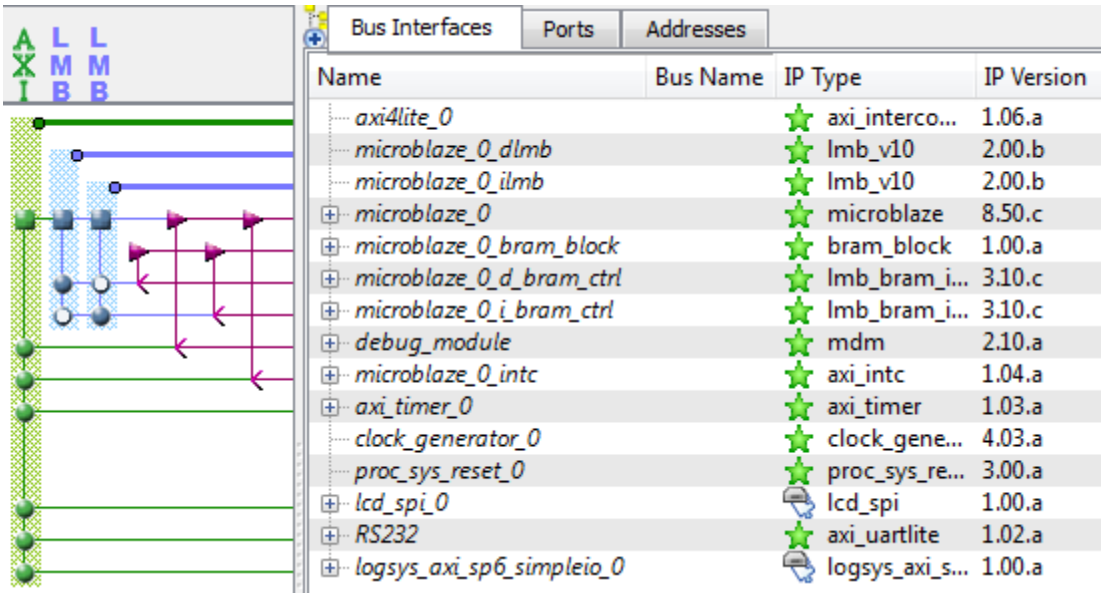
1. Architektúra

A megvalósításhoz microblaze szoftprocesszort használtunk, a következő paraméterekkel:

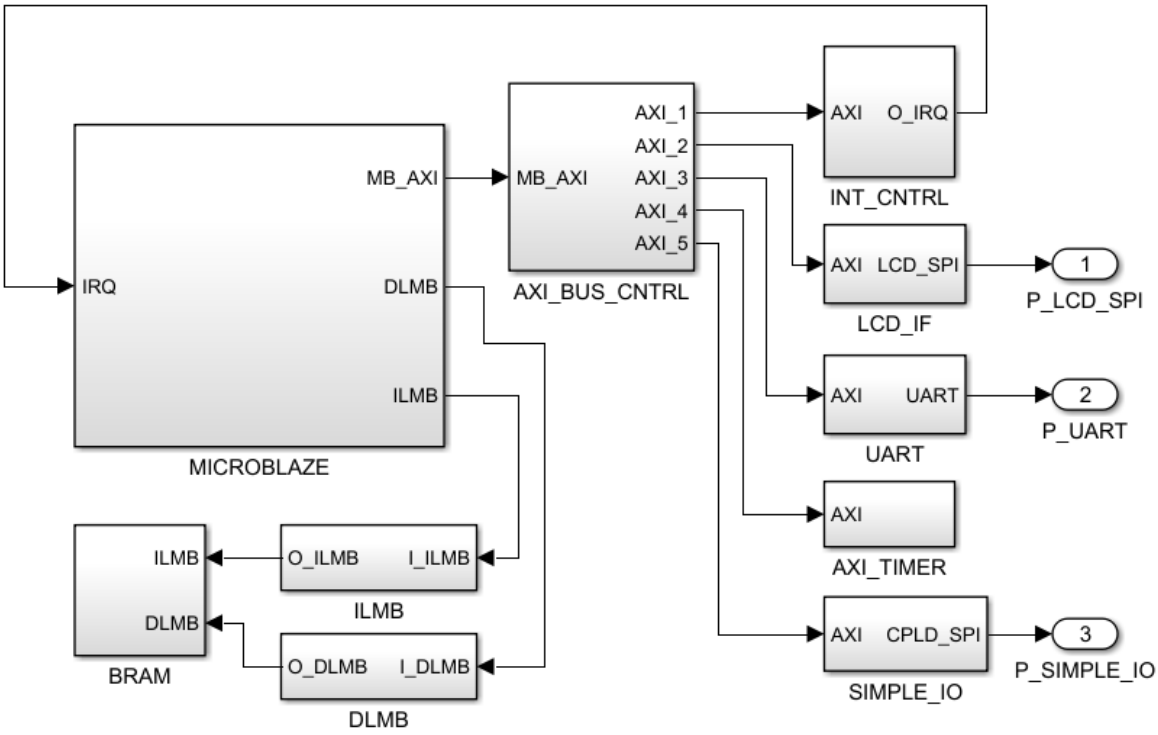
- Engedélyezett barrel shifter
- Hardveres integer szorzó
- Hardveres integer osztó
- Letiltott lebegőpontos egység
- Letiltott cache
- Letiltott memóriamenedzsment egység
- Engedélyezett debug interface
- 64 kB blokkram
- 50MHz órajel

A processzort ki kellett egészítenünk az általunk megírt LCD vezérlővel, illetve a gombok és ledek kezelésére használt simpleIO perifériával.
Ezeken kívül a megvalósításhoz felhasználtunk egy axi timer és egy interrupt controller IP-t.

A processzor és a hozzáadott perifériák:



1. ábra: Perifériák



2. ábra: Blokkvázlat

LCD periféria leírása:

A periféria konfigurálható órajelgenerátorral rendelkezik. A legnagyobb előállítható frekvencia a bemeneti órajel fele. Jelen esetben ez 25 MHz, ami pont ideális, mert a használt kijelző maximális interfész órajele 33 MHz lehet.

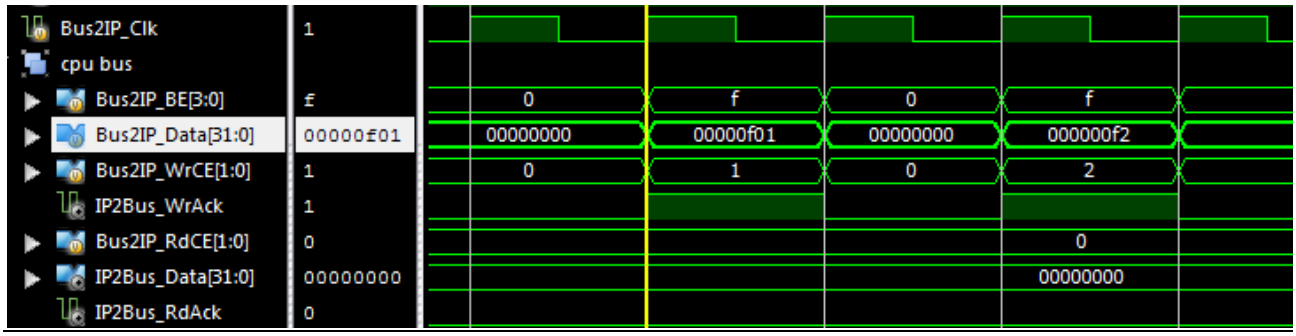
A periféria két belső regiszterrel rendelkezik:

Regiszter neve	**AXI cím**	**Bit[11]**	**Bit[10]**	**Bit[9]**	**Bit[8]**	**Bit[7:0]**
Control reg	0x00	Slave select	Global EN	Int EN	Int clr	Baudrate
Status reg	0x04	X	Irq reg	Busy reg	CMDn/Data	Data

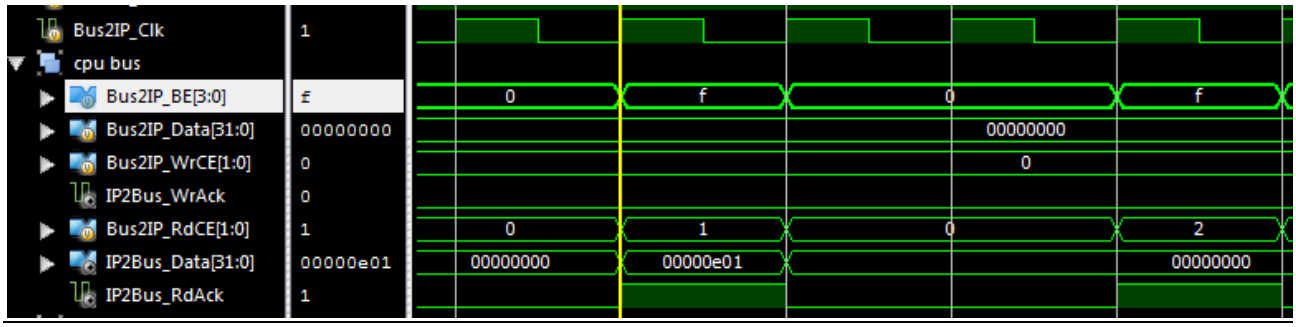
A preiféria konfigurálása után (baudrate beállítása, global enable, slave select) a kijelző írása automatikusan történik, az átküldeni kívánt biteket a Status regiszter [7:0] bitjeibe kell írni, továbbá a CMDn/Data bitet kell megfelelő értékre állítani.

A perifériák szimuciós eredményei:

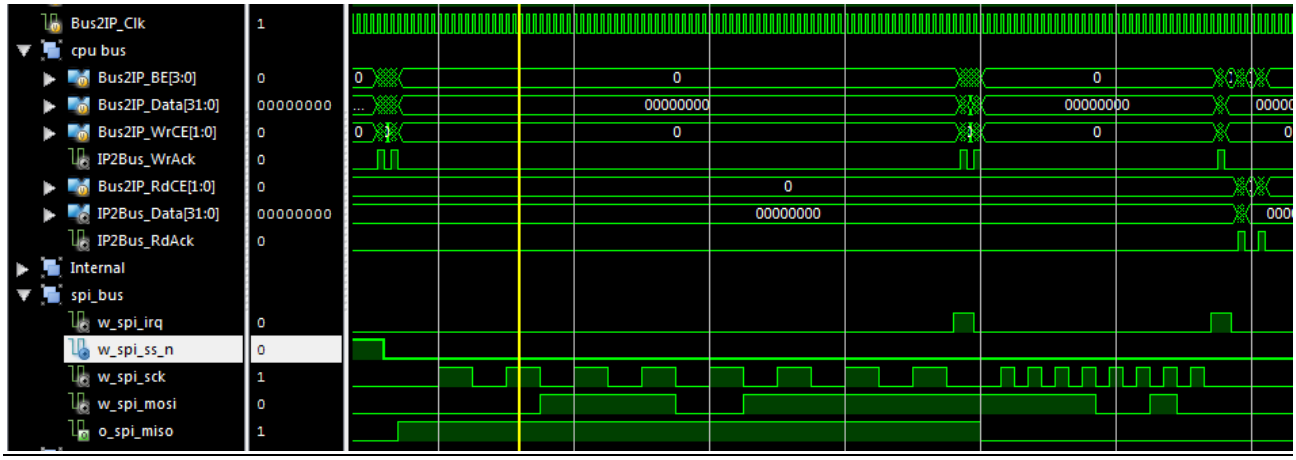
LCD vezérlő:



3. ábra: AXI LCD regiszter írási ciklus

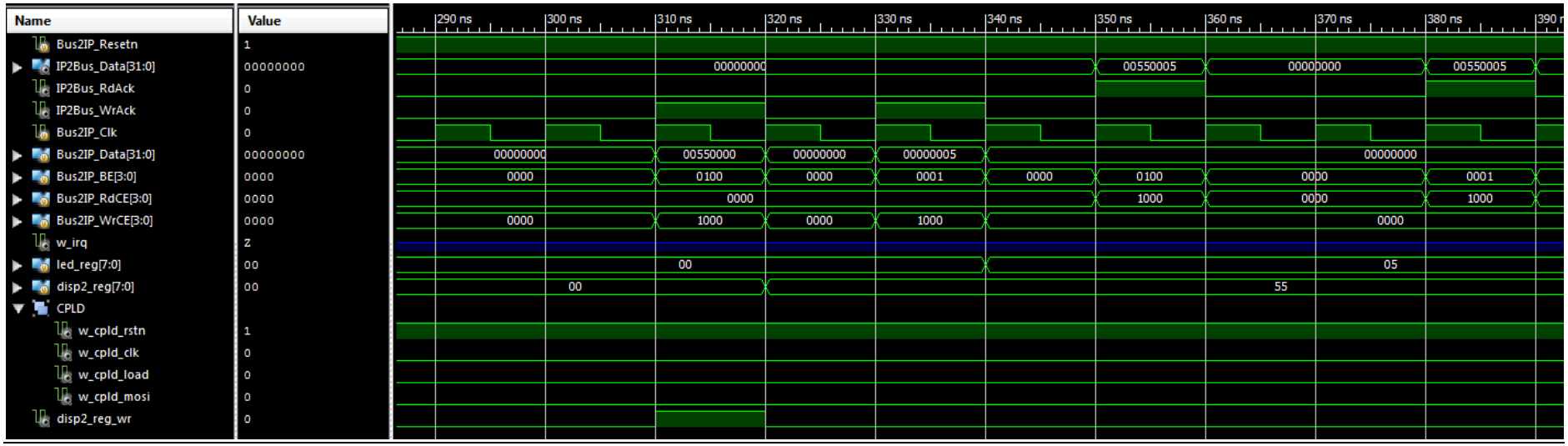


4. ábra: AXI LCD regiszter olvasási ciklus

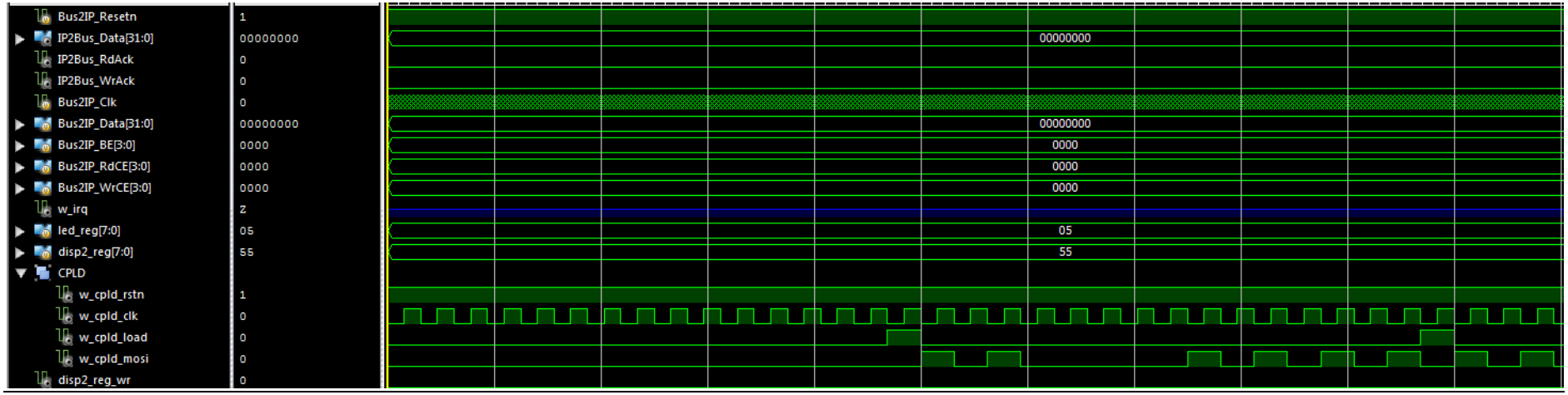


5. ábra: LCD soros adatátvitel

SimpleIO periféria:



6. ábra: SimpleIO regiszter írás/olvasás



7. ábra: SimpleIO CPLD interfész írás

2. Szoftver

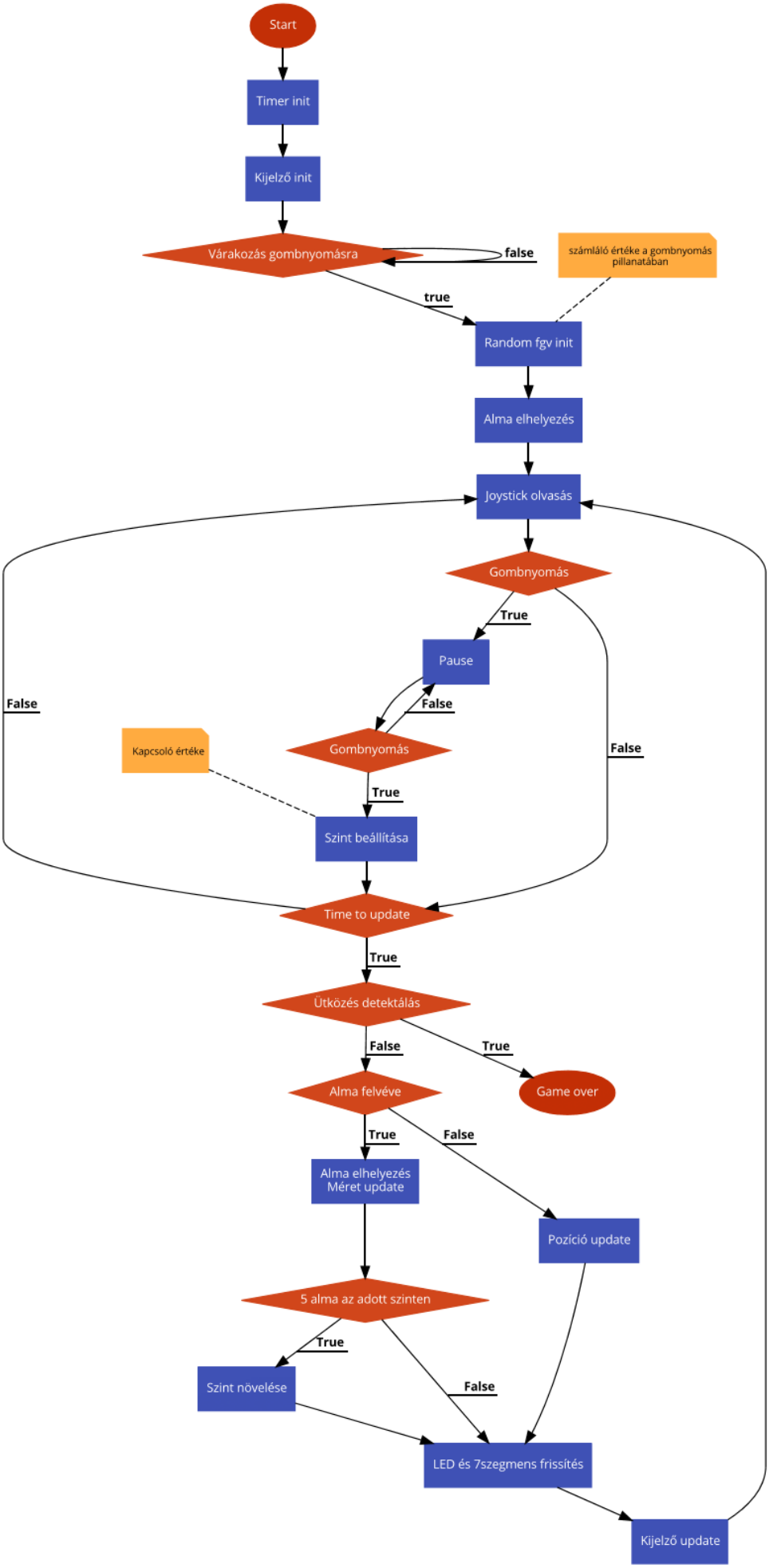
A kód folyamatábrája a következő oldalon látható. A játék a joystick megnyomásával veszi kezdetét, a megnyomás pillanatában a timer regisztert kiolvassuk, az "alma" elhelyezésére használt pszeudorandom generátor inicializálása ezzel az értékkel történik.

Az LCD kijelző pixeleinek értékét egy 102*64 méretű egydimenziós bájtos tömbben tároljuk. A memóriafoglalás malloc fügvénnyel történik. Ennek az az oka, hogy a fordító nem jelezte az elegendő memória hiányát a fix méretű tömbre.

A játék megvalósítására akövetkező módszert alkalmazzuk:

- A játéktér 3x3 pixeles négyzetekből áll, tehát a kígyó 3 pixelenként mozog
- Ezeknek a négyzeteknek a középpontjait egy játéktér méretű, snake nevű tömbben tároljuk (33*22 elem)
- A snake tömbben, a játéktér szélének megfelelő helyeken 0xFFFF értékkel helyezkednek el a keret elemei
- A kígyó pixeleit a snake tömb tárolja, a következő módon:
    - A fejhez tartozó érték megegyezik a kígyó hosszával
    - A farok felé közeledve a pixelek értéke csökken, a farok értéke 1
- A fej x-y koodinátáját, illetve a haladási irányt külön változokban tároljuk
- A játéktér frissítésekor (mapupdate függvény) a snake tömb minden elemét 1-gyel csökkentjük (kivéve az alma helyének megfelelő elemet illetve a keret elemeit), majd egy új fejet helyezünk a megfelelő koordinátára, a kígyó hosszának megfelelő értékkel
- Ütközésdetektáláshoz a fej új koordinátájának megfelelő elemet vizsgáljuk a snake tömbből, ha ez nagyobb mint 0, és nem esik egybe az alma pozíciójával, akkor ütközés történt

```
void mapupdate(uint16_t * map){// decrease all map element values by one (this
makes the tail disappear)
  int i;
  for (i=0; i < (MAPSIZE); i++ ){
        if (map[i] != 0xffff){ // if not part of the edge
              if (i != alma_y*MAP_WIDTH + alma_x){ // if does not match the food
position
                    if (map[i] != 0){
                          map[i] -= 1;
                    }
              }
        }
  }
  map[headpos_y*MAP_WIDTH + headpos_x] = snake_size;  // give maximum value to the
head
}
```


Start
Timer init
Kijelző init
Várakozás gombnyomásra
false
számláló értéke a gombnyomás pillanatában
true
Random fgv init
Alma elhelyezés
Joystick olvasás
Gombnyomás
True
Pause
False
False
Kapcsoló értéke
Gombnyomás
False
True
Szint beállítása
Time to update
True
Ütközés detektálás
False
True
Game over
Alma felvéve
True
False
Alma elhelyezés
Méret update
Pozíció update
5 alma az adott szinten
True
Szint növelése
False
LED és 7szegmens frissítés
Kijelző update

3. Felhasználói dokumentáció

A hardver felkonfigurálása után a játék a jostick lenyomására vár. A lenyomást követően megjelenik a kígyó és az "alma", a kép azonban "áll".

A jostick ismételt lenyomását követően kezdetét veszi a játék, a lenyomás pillanatában beállított szinten. A szintet a dipswitch segítségével állíthatjuk be.

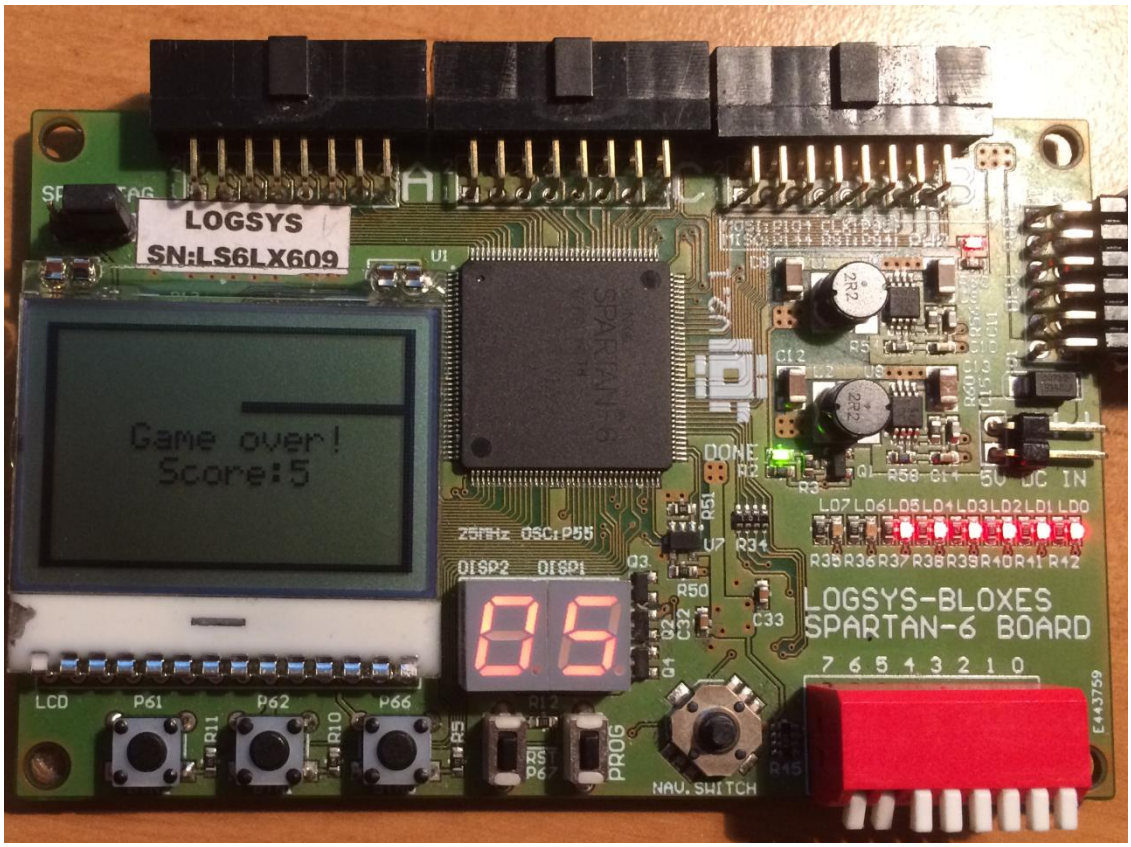
A játék bármikor megállítható, ezt a jostick lenyomásával tehetjük meg. A következő lenyomással folytatható. A szintet játék közben megállításkor állíthatjuk, az indítás pillanatában beállított szintet veszi fel.

Adott szinten felvett 5 alma után a játék automatikusan a következő szintre lép. Ezt megállítással illetve újraindítással felülírhatjuk.

A pontszám a hétszegmenses kijelzőn, a szint a ledeken olvasható le.

A játék véget ér, ha a kígyó falnak, vagy saját magának ütközik. A játék végén a "Game over" szöveg, illetve a pontszám kiírásra kerül a kijelzőre.

Új játékot a jostick kétszeri lenyomásával indíthatunk.



8. ábra: Game over

Szoftver forráskódok:

snake.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <xparameters.h>
#include <xintc_l.h>
#include <xtmrctr_l.h>
#include <mb_interface.h>
#include "platform.h"
#include "lcd_lib/lcd_lib.h"
#include "io_lib/io_lib.h"
#include "font/font.h"
#define MAP_WIDTH 34
#define MAP_HEIGHT 22
#define MAPSIZE MAP_WIDTH*MAP_HEIGHT
#define SNAKESIZE_START 10
#define LVLTHR 5

volatile uint8_t counter = 0;
volatile uint8_t update = 0;

int8_t headpos_x = 0;
int8_t headpos_y = 0;
int8_t direction = RIGHT;
int8_t direction_x = 0;
int8_t direction_y = 0;
int8_t activedirection_x = 0;
int8_t activedirection_y = 0;
int16_t snake_size = 0;
uint8_t running = 0;
uint8_t alma_x;
uint8_t alma_y;
uint8_t snake_grow = 0;

void TimerInit(void);

void drawframe(uint16_t width, uint16_t height, uint16_t * array);
void drawframe2(uint16_t width, uint16_t height, uint8_t * array);
void mapupdate(uint16_t * map);

int8_t snakecheck(uint16_t * map);
uint8_t almagen(uint16_t * map);
void placealma(uint16_t * map);
void maplcdconv(uint16_t width, uint16_t height, uint16_t * map,
        uint16_t lcdwidth, uint16_t lcdheight, uint8_t * framebuffer);

inline void drawpixel(uint16_t loc_x, uint16_t loc_y, uint16_t value, uint8_t *
framebuffer);

void printchar(uint8_t row, uint8_t col, uint8_t * array, char ch);
void printstring(uint8_t row, uint8_t col, uint8_t * array, char * string);
void printnum(uint8_t row, uint8_t col, uint8_t * array, char * num);
void num2string(char num, char * string);
uint8_t getmsb(uint8_t num);

void timer_int_handler(void *instance_Ptr)
{
  if (update != 0){
  update--;
```

```
  }
  // clear interrupt flag
  unsigned long csr;
  csr = XTmrCtr_GetControlStatusReg(XPAR_AXI_TIMER_0_BASEADDR, 0);
  XTmrCtr_SetControlStatusReg(XPAR_AXI_TIMER_0_BASEADDR, 0, csr);
}

int main()
{
  init_platform();
  TimerInit();
  microblaze_enable_interrupts();
  LcdInit();

  uint8_t * framebuffer = NULL;
  framebuffer = malloc(LCD_SIZE * sizeof(uint8_t));

  if (framebuffer == NULL){
    print ("not enough free memory\r\n");
    return 0;
  }
  else print("framebuffer OK\r\n");

  uint16_t * snake = NULL;
  snake = malloc(MAPSIZE * sizeof(uint16_t));

  if (snake == NULL){
    print ("not enough free memory\r\n");
    return 0;
  }
  else print("map OK\r\n");

while(1){

  drawframe2(LCD_WIDTH, LCD_HEIGHT, framebuffer);
  drawframe(MAP_WIDTH, MAP_HEIGHT, snake);

  headpos_x = MAP_WIDTH / 2;
  headpos_y = MAP_HEIGHT / 2;
  direction_x = 1;
  direction_y = 0;
  snake_size = SNAKESIZE_START;
  uint8_t running = 1;
  uint8_t midbutton = 0;
  uint8_t level;
  int64_t dummy64 = 0;
  uint8_t dummy;

  // initial snake
  for (dummy = 0; dummy < snake_size; dummy++){
    snake[headpos_y*MAP_WIDTH + headpos_x - dummy] = snake_size - dummy;
  }
  while(NavswR() != PUSH); // wait for buttonpress
  dummy64 = XTmrCtr_GetTimerCounterReg(XPAR_AXI_TIMER_0_BASEADDR,0);
  srand(dummy64); // random init
  level = getmsb(DipswR());
  update = 8 - level;
```

```
// place food
  uint8_t almacheck;
  do {
  almacheck = almagen(snake);
  } while (almacheck);
  placealma(snake);
  maplcdconv(MAP_WIDTH, MAP_HEIGHT, snake, LCD_WIDTH, LCD_HEIGHT, framebuffer);
  LcdArrayConv(framebuffer);
  // game cycle
    while (running){
    // btn read
    switch(NavswR()){
      case 0x01:   if (activedirection_y != 1){
                direction_x = 0;
                direction_y = -1;
            } break;

      case 0x02:   if (activedirection_y != -1){
                direction_x = 0;
                direction_y = 1;
            } break;

      case RIGHT: if (activedirection_x != -1){
                direction_x = 1;
                direction_y = 0;
            } break;

      case LEFT:   if (activedirection_x != 1){
                direction_x = -1;
                direction_y = 0;
            } break;

      case PUSH:   midbutton = 1; break;
    }

    if (midbutton){
      update = 4;
      while(update); //button debounce
      while (NavswR() != PUSH ); //paused until next push
      update = 4;
      while(update);
      midbutton = 0;
      level = getmsb(DipswR()); //read the dipswitch
      update = 0;
    }

      if(update == 0 && midbutton == 0) {
        if (snakecheck(snake) != 0) { //collision detected, game over
          running = 0;
          printstring(26, 24, framebuffer, "Game over!");
          uint8_t printpos = 33;
          char scorebuf[5];
          int16_t score = snake_size - SNAKESIZE_START;
          // print the score

          if(score > 99) {
            num2string(score / 100 , &scorebuf[0]);
            score = score % 100;
            num2string(score / 10 , &scorebuf[1]);
            num2string(score % 10 , &scorebuf[2]);
            scorebuf[3] = 0;
```

```
            printpos = 26;
          }
          else if(score > 9){
            num2string(score / 10 , &scorebuf[0]);
            num2string(score % 10 , &scorebuf[1]);
            scorebuf[2] = 0;
            printpos = 29;
          }
          else {
            num2string(score, &scorebuf[0]);
            scorebuf[1] = 0;
            printpos = 32;
          }

          printstring(35, printpos, framebuffer, "Score:");
          printstring(35, printpos + 36, framebuffer, scorebuf);
          LcdArrayConv(framebuffer);
          while (NavswR() != PUSH );
          break;
        }
        if (!snake_grow) {  //food not picked up
          mapupdate(snake);
        }
        else { //food picked up
          do {
            almacheck = almagen(snake);
            } while (almacheck);
            placealma(snake);
            if ((snake_size - SNAKESIZE_START) % LVLTHR == 0 && (snake_size -
SNAKESIZE_START) > 0){ //increase level after LVLTHR points
              if (level < 8) {
                level++;
              }
            }
        }
        maplcdconv(MAP_WIDTH, MAP_HEIGHT, snake, LCD_WIDTH, LCD_HEIGHT, framebuffer);
        LcdArrayConv(framebuffer);
        DispW(snake_size - SNAKESIZE_START);
        LedW(level);
        update = 8 - level;  //set delay
      }
    }
  }
 return 0;
}
void drawframe(uint16_t width, uint16_t height, uint16_t * array){  // draw frame on
map (invisible, only for collision detection)
  uint16_t i;

  for (i=0; i < width * height;i++ ){
    if (i < width || (i % width) == 0 || i > width * height - width || (i % (width)) ==
width - 1){
      array[i] = 0xffff; // frame pixels have 0xff value for easy recognition (no value
decrement on map update)
    }
    else {
      array[i] = 0;
    }
  }
}
```

```
void drawframe2(uint16_t width, uint16_t height, uint8_t * array){  // draw frame on
the LCD
  uint16_t i;
  for (i=0; i < width * height; i++ ){
    if (i < 2*width - 1 || (i % width) < 3 || i > width * height - 2*width || (i %
(width)) >= width - 3){
      array[i] = 0xff;
    }

    else {
      array[i] = 0;
    }
  }
}


int8_t snakecheck(uint16_t * map){ // collision detection
  uint8_t ret=1;

  activedirection_x = direction_x;
  activedirection_y = direction_y;

  headpos_x = headpos_x + activedirection_x;
  headpos_y = headpos_y + activedirection_y;

  if (headpos_y == alma_y && headpos_x == alma_x){   // ate food
    snake_grow = 1;
    snake_size += 1;
    ret = 0;
  }
  else {                          // no collision
    snake_grow = 0;
    ret = 0;
  }

  if (map[headpos_y*MAP_WIDTH + headpos_x] && snake_grow == 0){  // collision detected
    running = 0;
    ret = -1;
  }

return ret;
}

void mapupdate(uint16_t * map){ // decrease all map element values by one (this makes
the tail disappear)
  int i;
  for (i=0; i< (MAPSIZE);i++ ){
      if (map[i] != 0xffff){             // if not part of the edge
        if (i != alma_y*MAP_WIDTH + alma_x){   // if does not match the food position
          if (map[i] != 0){
            map[i] -= 1;
          }
        }
      }
  }
  map[headpos_y*MAP_WIDTH + headpos_x] = snake_size;  // give maximum value to the head
}
```

```
void maplcdconv(uint16_t width, uint16_t height, uint16_t * map,
        uint16_t lcdwidth, uint16_t lcdheight, uint8_t * framebuffer){ // convert the
small map to the LCD resolution
  int x, y;
  for (y = 1; y < height - 1; y++){
    for (x = 1; x < width - 1; x++){
      //framebuffer[(y*3*lcdwidth + 1 + 3*x] = map[y*width + x]; // only draws the
middle points
      drawpixel(1+3*x, y*3, map[y*width + x], framebuffer);
    }
  }
}

inline void drawpixel(uint16_t loc_x, uint16_t loc_y, uint16_t value, uint8_t *
framebuffer){ // draw the other 8 pixels around the middle coordinate
  uint16_t x, y;
  for (y = loc_y - 1; y <= loc_y + 1; y++){
    for (x = loc_x - 1; x <= loc_x + 1; x++){
      framebuffer[y * LCD_WIDTH + x] = value;
    }
  }
}

uint8_t almagen(uint16_t * map){ // generate food
  alma_x = 1 + (rand() % (MAP_WIDTH-1));
  alma_y = 1 + (rand() % (MAP_HEIGHT-1));

  if (map[alma_y*MAP_WIDTH + alma_x]){
    return 1;
  }
  else return 0;
}

void placealma(uint16_t * map){ // place generated food
  map[alma_y*MAP_WIDTH + alma_x] = snake_size;
}

void printchar(uint8_t row, uint8_t col, uint8_t * array, char ch) { // print character
to framebuffer
  unsigned char buf[5];
  uint8_t i, b;

  for (i = 0; i < 5; i++)  {
    buf[i] = font5x8[5*ch + i]; // read from character font array
  }

  for (i = 0; i < 5; i++)  {
    for (b = 0; b < 8; b++) {
      array[row*LCD_WIDTH + b*LCD_WIDTH + col + i] = (buf[i] & 1<<b); // copy character
to framebuffer
    }
  }

  for (b = 0; b < 8; b++) {
    array[row*LCD_WIDTH + b*LCD_WIDTH + col + 5] = 0; // clear screen between
characters (1 vertical line)
  }
}
```

```
void printstring(uint8_t row, uint8_t col, uint8_t * array, char * string) { // print
string to framebuffer
  unsigned char buf;
  while(*string != 0) {
    buf = *string;
    printchar(row,col,array, buf - 32);
    col += 6; // 1 empty line between characters
    string++;
  }
}

void num2string(char num, char * string) {
  *string = num + 48 ;
}

uint8_t getmsb(uint8_t num) {
  uint8_t shift;
  for (shift = 8; shift > 0; shift--) {
    if(num & (0x01 << (shift-1))) {
      return shift;
    }
  }
  return 0;
}

void TimerInit(void){
  // register interrupt routines
  XIntc_RegisterHandler(
      XPAR_MICROBLAZE_0_INTC_BASEADDR,
      XPAR_MICROBLAZE_0_INTC_AXI_TIMER_0_INTERRUPT_INTR, // timer interrupt
      (XInterruptHandler)timer_int_handler,
      NULL
  );
  // enable interrupts
  // configure the handler
  XIntc_MasterEnable(XPAR_MICROBLAZE_0_INTC_BASEADDR);
  XIntc_EnableIntr(XPAR_MICROBLAZE_0_INTC_BASEADDR,
      XPAR_AXI_TIMER_0_INTERRUPT_MASK
  );
  XTmrCtr_SetLoadReg(
      XPAR_AXI_TIMER_0_BASEADDR,
      0,
      XPAR_AXI_TIMER_0_CLOCK_FREQ_HZ / 20 - 2 //50 ms interrupt
  );
  XTmrCtr_SetControlStatusReg(
      XPAR_AXI_TIMER_0_BASEADDR,
      0,
      XTC_CSR_INT_OCCURED_MASK |
      XTC_CSR_LOAD_MASK
  );
  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
  );
}
```

lcd_lib.h:

```
#ifndef __LCD_LIB_H_
#define __LCD_LIB_H_

#include <inttypes.h>
#include <xparameters.h>

#define MEM8(addr)   (*(volatile uint8_t *)(addr))
#define MEM16(addr)  (*(volatile unsigned short *)(addr))
#define MEM32(addr)  (*(volatile unsigned long *)(addr))
//     SPI CONTROL REGISTER
//     [ SS_reg [11] | Global EN [10] | Interrupt EN [9] | Interrupt Clear [8] |
Baudrate [7:0] ]
#define SPICR                   0x00
#define SS_REG(bit)       (bit << 11)
#define GLOBAL_EN(bit)    (bit << 10)
#define INT_EN(bit)       (bit << 9)
#define INT_CLR(bit)      (bit << 8)

//     SPI STATUS
// [ IRQreg [10] | BUSY reg [9] | LCD CMDn/DATA [8] | Data [7:0] ]
#define SPISR                   0x04
#define IRQ_REG(bit)      (bit << 10)
#define BUSY_REG(bit)     (bit << 9)
#define CMDn_DAT(bit)     (bit << 8)

#define LCD_CONTROLREG    MEM32(XPAR_LCD_SPI_0_BASEADDR + SPICR)
#define LCD_STATUSREG     MEM32(XPAR_LCD_SPI_0_BASEADDR + SPISR)

//2: display start line set (lower 6 bits select first line on lcd from 64 lines in
memory)
#define LCD_START_LINE        0x40

//3: Page address set (lower 4 bits select one of 8 pages)
#define LCD_PAGE_ADDRESS      0xB0

//4: column address (lower 4 bits are upper / lower nibble of column address)
#define LCD_COL_ADDRESS_MSB   0x10
#define LCD_COL_ADDRESS_LSB   0x00  //second part of column address

#define LCD_RESET_CMD         0xE2

#define LCD_WIDTH                 102
#define LCD_HEIGHT                64
#define LCD_SIZE                  LCD_WIDTH*LCD_HEIGHT
#define LCD_PAGENUM               8

void LcdBusy(void);
uint32_t LcdCntrl(uint32_t cntrl);
void LcdCmd(uint8_t cmd);
void LcdData(uint8_t data);
void LcdSelect(void);
void LcdDeSelect(void);
void LcdInit(void);
void LcdReset(void);
void LcdSetFirstLine(uint8_t line);
void LcdSetColumnAddress(uint8_t col);
void LcdSetPageAddress(uint8_t page);
void LcdGoToXY(uint8_t col, uint8_t page);
```

```
void LcdEnable(void);
void LcdDisable(void);
void LcdArrayOut(uint8_t *data);
void LcdArrayConv(uint8_t *data);
inline void LcdLineOut(uint8_t data, uint8_t page, uint8_t x);

#endif
```

lcd_lib.c:

```
#include "lcd_lib.h"

inline void LcdBusy(void){
  uint32_t reg;
  do{
    reg = LCD_STATUSREG;
  } while (reg & BUSY_REG(1));
}

inline uint32_t LcdCntrl(uint32_t cntrl){
  LcdBusy();
  if (cntrl != 0)
    LCD_CONTROLREG = (cntrl);
  return LCD_CONTROLREG;
}

inline void LcdCmd(uint8_t cmd){
  uint32_t reg;
  LcdBusy();
  reg = cmd & (~CMDn_DAT(1));
  LCD_STATUSREG = reg;
}

inline void LcdData(uint8_t data){
  uint32_t reg;
  LcdBusy();
  reg = data | CMDn_DAT(1);
  LCD_STATUSREG = reg;
}

inline void LcdSelect(void){
  uint32_t cfg;
  LcdBusy();
  cfg = LCD_CONTROLREG;
  cfg = cfg | SS_REG(1);
  LCD_CONTROLREG = cfg;
}

inline void LcdDeSelect(void){
  uint32_t cfg;
  LcdBusy();
  cfg = LCD_CONTROLREG;
  cfg = cfg & (~SS_REG(1));
  LCD_CONTROLREG = cfg;
}

inline void LcdEnable(void){
  uint32_t cfg;
  LcdBusy();
  cfg = LCD_CONTROLREG;
```

```
  cfg = cfg | GLOBAL_EN(1);
  LCD_CONTROLREG = cfg;
}

inline void LcdDisable(void){
  uint32_t cfg;
  LcdBusy();
  cfg = LCD_CONTROLREG;
  cfg = cfg & (~GLOBAL_EN(1));
  LCD_CONTROLREG = cfg;
}

void LcdInit(void)
{
  uint8_t init_seq [] =
          {  0x40,   // Fuggoleges gorgetes    Az elso megjelenitett sor a 0
            0xA0,   // SEG irany beallitas    Normal iranyu oszlopcimzes
            0xC8,  // COM irany beallitas    Forditott iranyu sorcimzes
            0xA4,  // Minden pixel be       Az SRAM tartalom megjelenitese
            0xA6,  // Inverz kijelzes       Az inverz megjelenites tiltasa
            0xA2,  // LCD bias beallitas    1/9 LCD bias
            0x2F,  // Tapellatas vezerles    A tapellatas bekapcsolasa
            0x24,  // Tapellatas vezerles    A kontraszt beallitasa
            0x81,  // VEV beallitas      A kontraszt beallitasa
            0x2C,  // VEV beallitas      A kontraszt beallitasa
            0xFA,  // APC0 regiszter irasa    Homerseklet kompenzacio
            0x90,  // APC0 regiszter irasa    Homerseklet kompenzacio
            0xAF,  // Kijelzo engedelyezes    A megjelenites bekapcsolasa
            0x00  };

  LcdEnable();
  LcdSelect();

  uint8_t i;
  for(i=0; init_seq[i] != 0; i++){
    LcdCmd(init_seq[i]);
  }
}

inline void LcdReset(void){
  LcdCmd(LCD_RESET_CMD);
}

inline void LcdSetFirstLine(uint8_t line){
  LcdCmd(LCD_START_LINE | ((line) & 0x3F));
}

inline void LcdSetColumnAddress(uint8_t col){
  LcdCmd(LCD_COL_ADDRESS_MSB | ((col>>4) & 0x0F));
  LcdCmd(LCD_COL_ADDRESS_LSB | ((col) & 0x0F));
}

inline void LcdSetPageAddress(uint8_t page){
   LcdCmd(LCD_PAGE_ADDRESS | ((page) & 0x0F));
}

inline void LcdGoToXY(uint8_t col, uint8_t page){
  LcdSetColumnAddress(0x1e + col);
  LcdSetPageAddress(page);
}
```

```
inline void LcdArrayOut(uint8_t *data){
  uint8_t page, col;

  for(page = 0; page < LCD_PAGENUM; page++){
    for(col = 0; col < LCD_WIDTH;col++){
      LcdGoToXY(col,page);
      LcdData(data[page*LCD_WIDTH + col]);
    }
  }
}

inline void LcdLineOut(uint8_t data, uint8_t page, uint8_t x){
  LcdGoToXY(x,page);
  LcdData(data);
}

void LcdArrayConv(uint8_t *data){
  uint8_t y, x, bit;
  uint8_t dataout = 0;

  for(y = 0; y < LCD_PAGENUM; y++){
    for (x = 0; x < LCD_WIDTH; x++){
      for (bit = 0; bit < 8; bit++){
        if (data[LCD_WIDTH*8*y+bit*LCD_WIDTH+x]){
          dataout |= (1 << bit);
        }
        else {
          dataout &= (~(1 << bit));
        }
      }
      LcdLineOut(dataout,y, x);
    }
  }
}
```

io_lib.h:

```
#ifndef __IO_LIB_H_
#define __IO_LIB_H_

#include <inttypes.h>
#include <xparameters.h>

#define MEM8(addr)   (*(volatile uint8_t *)(addr))
#define MEM16(addr)  (*(volatile unsigned short *)(addr))
#define MEM32(addr)  (*(volatile unsigned long *)(addr))

#define LEDREG                  0x00 // 8bit W
#define DISPREG1                0x01 // 8bit W
#define DISPREG2                0x02 // 8bit W
#define DIPSWITCH               0x06 // 8bit R
#define NAVSWITCH               0x0A // 8bit R

#define O_DISP1     MEM8(XPAR_LOGSYS_AXI_SP6_SIMPLEIO_0_BASEADDR + DISPREG1)
#define O_DISP2     MEM8(XPAR_LOGSYS_AXI_SP6_SIMPLEIO_0_BASEADDR + DISPREG2)
#define O_LED             MEM8(XPAR_LOGSYS_AXI_SP6_SIMPLEIO_0_BASEADDR + LEDREG)
#define I_DIPSW     MEM32(XPAR_LOGSYS_AXI_SP6_SIMPLEIO_0_BASEADDR + DIPSWITCH)
#define I_NAVSW     MEM32(XPAR_LOGSYS_AXI_SP6_SIMPLEIO_0_BASEADDR + NAVSWITCH)
```

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

const unsigned char bin2sevenseg[] = {
             0x3f,
             0x06,
             0x5b,
             0x4f,
             0x66,
             0x6d,
             0x7d,
             0x07,
             0x7f,
             0x6f
};

const unsigned char ledtable[] = {
             0x00,
             0x01,
             0x03,
             0x07,
             0x0f,
             0x1f,
             0x3f,
             0x7f,
             0xff
};
void Disp1W(uint8_t datd1);
void Disp2W(uint8_t datd2);
void DispW(uint8_t dat);
void LedW(uint8_t dat);
uint8_t DipswR(void);
uint8_t NavswR(void);
#endif
```

io_lib.c:

```
#include "io_lib.h"

inline void Disp1W(uint8_t dat){
       O_DISP1 = dat;
}

inline void Disp2W(uint8_t dat){
       O_DISP2 = dat;
}

void DispW(uint8_t dat){
       if (dat>99) dat = 66;
       Disp2W(bin2sevenseg[dat / 10]);
       Disp1W(bin2sevenseg[dat % 10]);
}

void LedW(uint8_t dat){
       if(dat > 8) dat = 8;
       O_LED = ledtable[dat];
}
```

```
uint8_t DipswR(void){
	uint8_t dat;
	dat = (I_DIPSW & 0x00ff0000) >> 16;
	return dat;
}

uint8_t NavswR(void){
	uint8_t dat;
	dat = (I_NAVSW & 0x00ff0000) >> 16;
	return dat;
}
```

font.h:

```
#ifndef __FONT_H_
#define __FONT_H_

#include <inttypes.h>

const uint8_t font5x8[] = {
0x00, 0x00, 0x00, 0x00, 0x00,// (space)
0x00, 0x00, 0x5F, 0x00, 0x00,// !
0x00, 0x07, 0x00, 0x07, 0x00,// "
0x14, 0x7F, 0x14, 0x7F, 0x14,// #
0x24, 0x2A, 0x7F, 0x2A, 0x12,// $
0x23, 0x13, 0x08, 0x64, 0x62,// %
0x36, 0x49, 0x55, 0x22, 0x50,// &
0x00, 0x05, 0x03, 0x00, 0x00,// '
0x00, 0x1C, 0x22, 0x41, 0x00,// (
0x00, 0x41, 0x22, 0x1C, 0x00,// )
0x08, 0x2A, 0x1C, 0x2A, 0x08,// *
0x08, 0x08, 0x3E, 0x08, 0x08,// +
0x00, 0x50, 0x30, 0x00, 0x00,// ,
0x08, 0x08, 0x08, 0x08, 0x08,// -
0x00, 0x30, 0x30, 0x00, 0x00,// .
0x20, 0x10, 0x08, 0x04, 0x02,// /
0x3E, 0x51, 0x49, 0x45, 0x3E,// 0
0x00, 0x42, 0x7F, 0x40, 0x00,// 1
0x42, 0x61, 0x51, 0x49, 0x46,// 2
0x21, 0x41, 0x45, 0x4B, 0x31,// 3
0x18, 0x14, 0x12, 0x7F, 0x10,// 4
0x27, 0x45, 0x45, 0x45, 0x39,// 5
0x3C, 0x4A, 0x49, 0x49, 0x30,// 6
0x01, 0x71, 0x09, 0x05, 0x03,// 7
0x36, 0x49, 0x49, 0x49, 0x36,// 8
0x06, 0x49, 0x49, 0x29, 0x1E,// 9
0x00, 0x36, 0x36, 0x00, 0x00,// :
0x00, 0x56, 0x36, 0x00, 0x00,// ;
0x00, 0x08, 0x14, 0x22, 0x41,// <
0x14, 0x14, 0x14, 0x14, 0x14,// =
0x41, 0x22, 0x14, 0x08, 0x00,// >
0x02, 0x01, 0x51, 0x09, 0x06,// ?
0x32, 0x49, 0x79, 0x41, 0x3E,// @
0x7E, 0x11, 0x11, 0x11, 0x7E,// A
0x7F, 0x49, 0x49, 0x49, 0x36,// B
0x3E, 0x41, 0x41, 0x41, 0x22,// C
0x7F, 0x41, 0x41, 0x22, 0x1C,// D
0x7F, 0x49, 0x49, 0x49, 0x41,// E
0x7F, 0x09, 0x09, 0x01, 0x01,// F
0x3E, 0x41, 0x41, 0x51, 0x32,// G
```

```
0x7F, 0x08, 0x08, 0x08, 0x7F,// H
0x00, 0x41, 0x7F, 0x41, 0x00,// I
0x20, 0x40, 0x41, 0x3F, 0x01,// J
0x7F, 0x08, 0x14, 0x22, 0x41,// K
0x7F, 0x40, 0x40, 0x40, 0x40,// L
0x7F, 0x02, 0x04, 0x02, 0x7F,// M
0x7F, 0x04, 0x08, 0x10, 0x7F,// N
0x3E, 0x41, 0x41, 0x41, 0x3E,// O
0x7F, 0x09, 0x09, 0x09, 0x06,// P
0x3E, 0x41, 0x51, 0x21, 0x5E,// Q
0x7F, 0x09, 0x19, 0x29, 0x46,// R
0x46, 0x49, 0x49, 0x49, 0x31,// S
0x01, 0x01, 0x7F, 0x01, 0x01,// T
0x3F, 0x40, 0x40, 0x40, 0x3F,// U
0x1F, 0x20, 0x40, 0x20, 0x1F,// V
0x7F, 0x20, 0x18, 0x20, 0x7F,// W
0x63, 0x14, 0x08, 0x14, 0x63,// X
0x03, 0x04, 0x78, 0x04, 0x03,// Y
0x61, 0x51, 0x49, 0x45, 0x43,// Z
0x00, 0x00, 0x7F, 0x41, 0x41,// [
0x02, 0x04, 0x08, 0x10, 0x20,// "\"
0x41, 0x41, 0x7F, 0x00, 0x00,// ]
0x04, 0x02, 0x01, 0x02, 0x04,// ^
0x40, 0x40, 0x40, 0x40, 0x40,// _
0x00, 0x01, 0x02, 0x04, 0x00,// `
0x20, 0x54, 0x54, 0x54, 0x78,// a
0x7F, 0x48, 0x44, 0x44, 0x38,// b
0x38, 0x44, 0x44, 0x44, 0x20,// c
0x38, 0x44, 0x44, 0x48, 0x7F,// d
0x38, 0x54, 0x54, 0x54, 0x18,// e
0x08, 0x7E, 0x09, 0x01, 0x02,// f
0x08, 0x14, 0x54, 0x54, 0x3C,// g
0x7F, 0x08, 0x04, 0x04, 0x78,// h
0x00, 0x44, 0x7D, 0x40, 0x00,// i
0x20, 0x40, 0x44, 0x3D, 0x00,// j
0x00, 0x7F, 0x10, 0x28, 0x44,// k
0x00, 0x41, 0x7F, 0x40, 0x00,// l
0x7C, 0x04, 0x18, 0x04, 0x78,// m
0x7C, 0x08, 0x04, 0x04, 0x78,// n
0x38, 0x44, 0x44, 0x44, 0x38,// o
0x7C, 0x14, 0x14, 0x14, 0x08,// p
0x08, 0x14, 0x14, 0x18, 0x7C,// q
0x7C, 0x08, 0x04, 0x04, 0x08,// r
0x48, 0x54, 0x54, 0x54, 0x20,// s
0x04, 0x3F, 0x44, 0x40, 0x20,// t
0x3C, 0x40, 0x40, 0x20, 0x7C,// u
0x1C, 0x20, 0x40, 0x20, 0x1C,// v
0x3C, 0x40, 0x30, 0x40, 0x3C,// w
0x44, 0x28, 0x10, 0x28, 0x44,// x
0x0C, 0x50, 0x50, 0x50, 0x3C,// y
0x44, 0x64, 0x54, 0x4C, 0x44,// z
0x00, 0x08, 0x36, 0x41, 0x00,// {
0x00, 0x00, 0x7F, 0x00, 0x00,// |
0x00, 0x41, 0x36, 0x08, 0x00,// }
0x08, 0x08, 0x2A, 0x1C, 0x08,// ->
0x08, 0x1C, 0x2A, 0x08, 0x08 // <-
};

#endif
```

Hardver forráskódok:

LCD vezérlő userlogic:

```
`uselib lib=unisims_ver
`uselib lib=proc_common_v3_00_a

module user_logic#(
        parameter C_NUM_REG                     = 2,          //Az IPIF által
dekódolt 32 bites regiszterek száma.
        parameter C_SLV_DWIDTH                  = 32              //Az
adatbusz szélessége bitekben.
)(
        // General Input Signals
        input  wire                         Bus2IP_Clk,           //Órajel.
        input  wire                         Bus2IP_Resetn,            //Aktív
alacsony reset jel.
        input  wire [C_SLV_DWIDTH-1:0]   Bus2IP_Data,             //Írási
adatbusz.
        input  wire [C_SLV_DWIDTH/8-1:0] Bus2IP_BE,               //Bájt
engedélyezõ jelek (csak írás esetén érvényesek).
        input  wire [C_NUM_REG-1:0]      Bus2IP_RdCE,             //A
regiszterek olvasás engedélyezõ jelei.
        input  wire [C_NUM_REG-1:0]      Bus2IP_WrCE,             //A
regiszterek írás engedélyezõ jelei.
        output reg [C_SLV_DWIDTH-1:0]   IP2Bus_Data,              //Olvasási
adatbusz.
        output wire                         IP2Bus_RdAck,             //Az
olvasási mûveletek nyugtázó jele.
        output wire                         IP2Bus_WrAck,             //Az írási
mûveletek nyugtázó jele.
        output wire                         IP2Bus_Error,             //Hibajelzés

        output wire                       o_irq,          // Interrupt Request

        // SPI Signals
        output wire          o_spi_miso,    // Master-In-Slave-Out / LCD
CMD/DATA
        output wire                   o_spi_mosi,    // Master-Out-Slave-In
        output wire                   o_spi_sck,     // Bus-Clock
        output wire                   o_spi_ss_n      // Slave-Select
);

//----------------------------------------------
// Net Alias
//----------------------------------------------
wire            i_sysclk =  Bus2IP_Clk;          // System Clock
wire            i_sysrst = ~Bus2IP_Resetn;       // System Reset
wire [31:0]     i_din    =  Bus2IP_Data;     // Data from Bus
wire [3:0]      i_byte_en=  Bus2IP_BE;
wire            i_cmd_rd =  Bus2IP_RdCE[1];
wire            i_cmd_wr =  Bus2IP_WrCE[1];
wire            i_dat_rd =  Bus2IP_RdCE[0];
wire            i_dat_wr =  Bus2IP_WrCE[0];
```

```
wire            o_ack_wr;
wire            o_ack_rd;

wire [31:0]     o_dout;     // Data to Bus
wire            o_ack;            // Acknowledge

//---------------------------------------------
// olvasási multiplexer
//---------------------------------------------
reg [31:0] rd_mux;
always @(*)
begin
        case(Bus2IP_RdCE)
                2'b10: IP2Bus_Data <= {20'b0, r_SPICR};
                2'b01: IP2Bus_Data <= {20'b0, r_SPISR};
                default: IP2Bus_Data <= 0;
        endcase
end

//Az IPIF felé menõ jelek meghajtása.
//assign o_dout = rd_mux;
assign o_ack_wr = ( r_ack_wr_c | r_ack_wr_d);
assign o_ack_rd = ( r_ack_rd_c | r_ack_rd_d);

//assign IP2Bus_Data = o_dout;
assign IP2Bus_Error = 1'b0;
//assign IP2Bus_WrAck = o_ack_wr;
//assign IP2Bus_RdAck = o_ack_rd;

assign IP2Bus_WrAck = |Bus2IP_WrCE;
assign IP2Bus_RdAck = |Bus2IP_RdCE;



//---------------------------------------------
// Registers for SPI
//---------------------------------------------
//      SPI CONTROL REGISTER
//      [ SS_reg [11] | Global EN [10] | Interrupt EN [9] | Interrupt Clear [8]
| Baudrate [7:0] ]
reg [11:0] r_SPICR;
//---------------------------------------------
//      SPI STATUS REGISTER
// [ IRQreg [10] | BUSY reg [9] | LCD CMDn/DATA [8] | Data [7:0] ]
reg [11:0]  r_SPISR;
//---------------------------------------------

//---------------------------------------------
// Control Register Value Settings
//---------------------------------------------
reg r_ack_wr_c;
reg r_ack_rd_c;
always @ (posedge i_sysclk)
begin
        if(i_sysrst) begin
```

```
                r_SPICR <= 12'b0;
                r_ack_wr_c <= 1'b0;
                r_ack_rd_c <= 1'b0;
                r_ack_rd_d <= 1'b0;
        end
        else if(i_cmd_wr && (i_byte_en == 4'b1111)) begin
                r_SPICR    <= i_din[11:0];
                r_ack_wr_c <= 1'b1;
                r_ack_rd_c <= 1'b0;
                r_ack_rd_d <= 1'b0;
        end
        else if(i_cmd_rd) begin
                r_ack_wr_c <= 1'b0;
                r_ack_rd_c <= 1'b1;
                r_ack_rd_d <= 1'b0;
        end
        else if (i_dat_rd) begin
                r_ack_wr_c <= 1'b0;
                r_ack_rd_c <= 1'b0;
                r_ack_rd_d <= 1'b1;
        end
        else begin
                r_ack_wr_c <= 1'b0;
                r_ack_rd_c <= 1'b0;
                r_ack_rd_d <= 1'b0;
                if (~r_SPISR[10])
                        r_SPICR[8] <= 1'b0;
        end
end

//-----------------------------------------------
// State Machine
//-----------------------------------------------
// 0    - Idle
// 1    - Start - SS Down, SCK Enable
// 2:9  - Transmission
// 10   - Stop  - SS Up, SCK Disable
//-----------------------------------------------
reg [3:0] r_state = 0;
reg       r_ack_wr_d;
reg       r_ack_rd_d;
reg       r_sck_en;
always @ (posedge i_sysclk)
begin
 if(i_sysrst) begin
        r_state <= 4'b0;
        r_ack_wr_d <= 0;
        r_sck_en <= 0;
        r_SPISR <= 12'b0;                                       // Status
Register Clear
 end
 else begin
        if (r_state == 4'h0) begin                              // Idle
                if (i_dat_wr && (r_SPICR[10])
                        && (i_byte_en == 4'b1111))
```

```
                begin
                        r_state <= 4'h1;
                        r_sck_en <= 0;
                        r_SPISR[9] <= 1;                                // BUSY
Flag Write
                        r_SPISR[8] <= i_din[8];                         // LCD
CMDn/DATA
                        r_SPISR[7:0] <= i_din[7:0];  // Data from cpu bus
                        r_ack_wr_d <= 1;                                //
Acknowledge
                end
        end
        else if  (r_state == 4'h1)  begin               // Start
                r_state <= 4'h2;
                r_ack_wr_d <= 0;
                r_sck_en <= 1;
        end
        else if ((r_state >= 4'h2) &&                   // Transmission
                        ((r_state <= 4'h9)))
        begin
                if (w_spi_sck_fall) begin                       // State Change on
Falling Edge
                        r_state <= r_state + 1;
                end
        end
        else if  (r_state == 4'hA)  begin               // Stop
                r_state <= 4'h0;
                if (r_SPICR[9] == 1) begin                      // If Interrupt
Enable Set
                        r_SPISR[10] <= 1;                               // Then
Interrupt Request
                end
                r_SPISR[9] <= 0;                                         // BUSY
Flag Clear
                r_SPISR[7:0] <= spi_shr_dout;
                r_sck_en <= 0;
        end
        if (r_SPICR[8])
                r_SPISR[10] <= 1'b0;
 end
end

assign o_spi_miso = r_SPISR[8];

//-----------------------------------------------
// Output Signal Generation
//-----------------------------------------------
assign o_irq = r_SPISR[10];
assign o_spi_ss_n = (~r_SPICR[11]);
assign o_spi_sck = (w_sck & r_sck_en);
//-----------------------------------------------
// Internal Signal Generation
//-----------------------------------------------
wire w_sck_en;
assign w_sck_en = (r_sck_en && r_SPICR[11]);
```

```
assign w_shr_ld = (r_state == 4'h1);
assign w_shr_sh = ((r_state >= 4'h2) && (r_state <= 4'hA) && w_spi_sck_fall && (
~o_spi_ss_n));
//-----------------------------------------------

//-----------------------------------------------
// SCK frequency divider module instantiation
//-----------------------------------------------
wire w_spi_sck_rise, w_spi_sck_fall, w_sck;
sckgen spi_sckgen (
        .i_sysclk(i_sysclk),
        .i_sysrst(i_sysrst),
        .i_en(w_sck_en),
        .i_baudrate(r_SPICR[7:0]),
        .o_sck(w_sck),
        .o_sck_rise(w_spi_sck_rise),
        .o_sck_fall(w_spi_sck_fall)
);
//-----------------------------------------------

//-----------------------------------------------
// SPI shift register module instantiation
//-----------------------------------------------
wire [7:0] spi_shr_dout;
shr spi_shr (
        .i_sysclk(i_sysclk),
        .i_sysrst(i_sysrst),
        .i_din(o_spi_miso),
        .i_sh(w_shr_sh),
        .i_ld(w_shr_ld),
        .i_ld_data(r_SPISR[7:0]),
        .o_dout(o_spi_mosi),
        .o_dstr(spi_shr_dout)
);
//-----------------------------------------------



//-----------------------------------------------
endmodule
```

LCD vezérlő shift register module:

```
module shr(
        input                   i_sysclk,
        input                   i_sysrst,

        input                   i_din,                  // Data In
        input                   i_sh,                   // Shift Signal
        input                   i_ld,           // Load Signal
        input  [7:0]    i_ld_data,        // Data to Load

        output                          o_dout,       // Data Out
        output          [7:0]  o_dstr           // Data to Store
);

//-----------------------------------------------
// Shift-Register Block
//-----------------------------------------------
reg [7:0] r_shr;
always @ (posedge i_sysclk)
begin
        if(i_sysrst)
                r_shr <= 8'b0;
        else if(i_ld)
                r_shr <= i_ld_data;
        else if(i_sh)
                r_shr <= {r_shr[6:0], i_din};
end
//-----------------------------------------------

//-----------------------------------------------
// Output Signal Generation
//-----------------------------------------------
assign o_dstr = r_shr;
assign o_dout = r_shr[7];
//-----------------------------------------------
endmodule
```

LCD vezérlő sckgen:

```
module sckgen(
        input                   i_sysclk,               // System Clock
        input                   i_sysrst,               // System Reset
        input                   i_en,                           // Enable Clock
Generator
        input  [7:0]    i_baudrate,      // Baudrate Divider
        output                          o_sck,                          // SPI SCK
        output                          o_sck_rise,  // SCK Rising Edge
        output                          o_sck_fall              // SCK Falling
Edge
);
//-----------------------------------------------
// Counter
//-----------------------------------------------
reg [7:0] r_cntr;
always @ (posedge i_sysclk)
begin
        if (i_sysrst)
                r_cntr <= 8'b0;
        else if(i_en) begin
                if(r_cntr == i_baudrate)
                        r_cntr <= 8'b0;
                else
                        r_cntr <= r_cntr + 1'b1;
        end
        else
                r_cntr <= 8'b0;
end
//-----------------------------------------------
// SCK Register
//-----------------------------------------------
reg r_sck;
always @ (posedge i_sysclk)
begin
        if (i_sysrst)
                r_sck <= 1'b0;
        else if(~i_en)
                r_sck <= 1'b0;
        else if(r_cntr == i_baudrate)
                r_sck <= ~r_sck;
end
//-----------------------------------------------
// Output Signal Generation
//-----------------------------------------------
assign o_sck            = ( r_sck) & (i_en);
assign o_sck_rise       = (~r_sck) & (r_cntr == i_baudrate) & (i_en);
assign o_sck_fall       = ( r_sck) & (r_cntr == i_baudrate) & (i_en);
//-----------------------------------------------

endmodule
```

SimpleIO periféria:

A http://home.mit.bme.hu/~rtamas/mikrorendszerek/EDK/ weboldalról letöltött simpleIO perifériát használtuk, módosítás nélkül.

Testbench forráskódok:

LCD_testbench:

```
module user_logic_tb;
// Inputs
        reg                     Bus2IP_Clk;
        reg                     Bus2IP_Resetn;
        reg   [31:0]            Bus2IP_Data;
        reg   [3:0]             Bus2IP_BE;
        reg   [1:0]             Bus2IP_RdCE;
        reg   [1:0]             Bus2IP_WrCE;
        wire  [31:0]            IP2Bus_Data;
        wire                    IP2Bus_RdAck;
        wire                    IP2Bus_WrAck;
        wire                    IP2Bus_Error;
        wire                    w_spi_irq;
        wire                    w_spi_miso;
        wire                    w_spi_mosi;
        wire                    w_spi_sck;
        wire                    w_spi_ss_n;

//-----------------------------------------------
// Instantiate the Unit Under Test (UUT)
//-----------------------------------------------
user_logic uut(
        // General input signals
        .Bus2IP_Clk(Bus2IP_Clk),
        .Bus2IP_Resetn(Bus2IP_Resetn),

        // Internal signals
        .Bus2IP_Data(Bus2IP_Data),                      // Data from bus
        .Bus2IP_BE(Bus2IP_BE),                          // Modify settings
        .Bus2IP_RdCE(Bus2IP_RdCE),                      // Write data
        .Bus2IP_WrCE(Bus2IP_WrCE),                      // Read data

        .IP2Bus_Data(IP2Bus_Data),              // Data to bus
        .IP2Bus_RdAck(IP2Bus_RdAck),                    // Acknowledge
        .IP2Bus_WrAck(IP2Bus_WrAck),                    // Acknowledge
        .IP2Bus_Error(IP2Bus_Error),

        .o_irq(w_spi_irq),                      // Interrupt request

        // SPI signals
        .o_spi_miso(w_spi_miso),        // Master-In-Slave-Out
        .o_spi_mosi(w_spi_mosi),        // Master-Out-Slave-In
        .o_spi_sck(w_spi_sck),          // Bus-Clock
        .o_spi_ss_n(w_spi_ss_n)                 // Slave-Select
);
//-----------------------------------------------
```

```
//------------------------------------------
// Bus write cycle
//------------------------------------------
task bus_write (input [1:0] addr,input [3:0] byte_en,  input [31:0] data);
begin
        #10 Bus2IP_WrCE <= addr;
        Bus2IP_BE       <= byte_en;
        Bus2IP_Data <= data;
        wait(IP2Bus_WrAck);
        #10 Bus2IP_WrCE <= 2'b00;
        Bus2IP_BE       <= 4'b0000;
        Bus2IP_Data <= 32'h0000_0000;
end
endtask
//------------------------------------------

//------------------------------------------
// Bus read cycle
//------------------------------------------
task bus_read (input [1:0] addr,input [3:0] byte_en);
begin
        #10 Bus2IP_RdCE <= addr;
        Bus2IP_BE       <= byte_en;
        wait(IP2Bus_RdAck);
        #10 Bus2IP_RdCE <= 2'b00;
        Bus2IP_BE       <= 4'b0000;
end
endtask
//------------------------------------------

//------------------------------------------
// Initialize Inputs
//------------------------------------------
initial begin
        Bus2IP_Clk              = 1;
        Bus2IP_Resetn   = 0;
        Bus2IP_Data             = 0;
        Bus2IP_BE               = 0;
        Bus2IP_RdCE             = 0;
        Bus2IP_WrCE             = 0;
end
//------------------------------------------

//------------------------------------------
// Generate clock
//------------------------------------------
always #5 Bus2IP_Clk = ~Bus2IP_Clk;
//------------------------------------------

//------------------------------------------
// Main Block
//------------------------------------------
initial begin
        #100
```

```
        Bus2IP_Resetn = 1;
        #200
        bus_write(2'b01,4'b1111, 32'h0000_0E04);                // Set baudrate,
int clr =0, int en, global en, SS=0
        bus_write(2'b10,4'b1111, 32'h0000_0137);                // Set data to
0x137
        wait(w_spi_irq);                                        //wait for
interrupt
        bus_write(2'b01,4'b1111, 32'h0000_0F01);
        bus_write(2'b10,4'b1111, 32'h0000_00F2);                // Set data to
0xF2
        wait(w_spi_irq);
        bus_write(2'b01,4'b1111, 32'h0000_0F01);
        #10
        bus_read(2'b01,4'b1111);
        #10
        bus_read(2'b10,4'b1111);
end
//-----------------------------------------------

endmodule
```

Simpleio testbench:

```
    module simple_io_tb;
    // Inputs
            reg                     Bus2IP_Clk;
            reg                     Bus2IP_Resetn;
            reg   [31:0]            Bus2IP_Data;
            reg   [3:0]             Bus2IP_BE;
            reg   [3:0]             Bus2IP_RdCE;
            reg   [3:0]             Bus2IP_WrCE;
            wire  [31:0]            IP2Bus_Data;
            wire                    IP2Bus_RdAck;
            wire                    IP2Bus_WrAck;
            wire                    IP2Bus_Error;

            wire                    w_irq;
            reg   [2:0]             r_btn_in;                       //GPIO
    input data.
            wire  [7:0]             w_sw_out;                       //GPIO
    input data.

            reg   [12:0]            r_gpio_i;                  //GPIO input data.
            wire  [12:0]            w_gpio_o;                  //GPIO output data.
            reg   [12:0]            r_gpio_dir;                //GPIO direction
    control (0:out, 1:in).

            //CPLD interface ports.
            wire                    w_cpld_jtagen;    //CPLD JTAG interface
    enable signal.
            wire                    w_cpld_rstn;        //CPLD reset signal.
            wire                    w_cpld_clk;          //CPLD clock signal.
```

```
        wire                    w_cpld_load;        //CPLD load signal.
        wire                    w_cpld_mosi;        //CPLD serial data
output.
        reg                     r_cpld_miso;

//-----------------------------------------------
// Instantiate the Unit Under Test (UUT)
//-----------------------------------------------
simple_io uut(
        // General input signals
        .Bus2IP_Clk(Bus2IP_Clk),
        .Bus2IP_Resetn(Bus2IP_Resetn),

        // Internal signals
        .Bus2IP_Data(Bus2IP_Data),              // Data from bus
        .Bus2IP_BE(Bus2IP_BE),                          // Modify settings
        .Bus2IP_RdCE(Bus2IP_RdCE),              // Write data
        .Bus2IP_WrCE(Bus2IP_WrCE),              // Read data

        .IP2Bus_Data(IP2Bus_Data),              // Data to bus
        .IP2Bus_RdAck(IP2Bus_RdAck),    // Acknowledge
        .IP2Bus_WrAck(IP2Bus_WrAck),    // Acknowledge
        .IP2Bus_Error(IP2Bus_Error),    //

        .irq(w_spi_irq),                                        //GPIO input
data.
        .btn_in(r_btn_in),                              //GPIO input data.
        .sw_out(w_sw_out),                              //GPIO input data.
        .gpio_I(13'd0),                 //GPIO input data.
        .gpio_O(w_gpio_o),              //GPIO output data.
        .gpio_T(),              //GPIO direction control (0:out, 1:in).

        //CPLD interface ports.
        .cpld_jtagen(w_cpld_jtagen),    //CPLD JTAG interface enable signal.
        .cpld_rstn(w_cpld_rstn),        //CPLD reset signal.
        .cpld_clk(w_cpld_clk),          //CPLD clock signal.
        .cpld_load(w_cpld_load),        //CPLD load signal.
        .cpld_mosi(w_cpld_mosi),        //CPLD serial data output.
        .cpld_miso(r_cpld_miso)         //CPLD serial data input.
);
//-----------------------------------------------

//-----------------------------------------------
// Bus write cycle
//-----------------------------------------------
task bus_write (input [3:0] addr,input [3:0] byte_en,  input [31:0] data);
begin
        #10 Bus2IP_WrCE <= addr;
        Bus2IP_BE       <= byte_en;
        Bus2IP_Data <= data;
        wait(IP2Bus_WrAck);
        #10 Bus2IP_WrCE <= 4'b0000;
        Bus2IP_BE       <= 4'b0000;
        Bus2IP_Data <= 32'h0000_0000;
end
```

```
endtask
//-------------------------------------------

//-------------------------------------------
// Bus read cycle
//-------------------------------------------
task bus_read (input [3:0] addr,input [3:0] byte_en);
begin
        #10 Bus2IP_RdCE <= addr;
        Bus2IP_BE       <= byte_en;
        wait(IP2Bus_RdAck);
        #10 Bus2IP_RdCE <= 4'b0000;
        Bus2IP_BE       <= 4'b0000;
end
endtask
//-------------------------------------------

//-------------------------------------------
// Initialize Inputs
//-------------------------------------------
initial begin
        Bus2IP_Clk              = 1;
        Bus2IP_Resetn   = 0;
        Bus2IP_Data             = 0;
        Bus2IP_BE               = 0;
        Bus2IP_RdCE             = 0;
        Bus2IP_WrCE             = 0;
end
//-------------------------------------------

//-------------------------------------------
// Generate clock
//-------------------------------------------
always #5 Bus2IP_Clk = ~Bus2IP_Clk;
//-------------------------------------------

//-------------------------------------------
// Main Block
//-------------------------------------------
initial begin
        #100
        Bus2IP_Resetn = 1;
        #200
        r_cpld_miso = 1;
        bus_write(4'b1000,4'b0100, 32'h00550000);               //disp2 write
        bus_write(4'b1000,4'b0001, 8'b0000_0101);               //led
write
        bus_read(4'b1000,4'b0100);
        #10
        bus_read(4'b1000,4'b0001);
end
//-------------------------------------------

endmodule
```