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

Cseppentő Viktor

PLATFORM JÁTÉK VERILOG NYELVEN

Témalaboratórium dokumentáció

KONZULENS

Dr. Fehér Béla

BUDAPEST, 2018

Tartalomjegyzék

Bevezetés

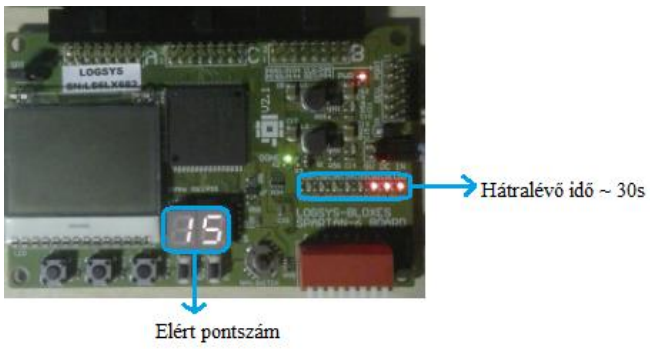
Feladatom egy egyszerű platformer videójáték implementálása volt Verilog hardverleíró nyelven LOGSYS Spartan 6 fejlesztői kártyára. Részfeladatok közé tartozott a játék megjelenítése VGA bemenetű monitoron, a panel perifériáinak a használata, egyszerű audio bővítmény kialakítása és egy szórakoztató játékmenet kreatív megalkotása. Munkám során törekedtem az erőforrások leghatékonyabb kihasználására és a nyelv minél átfogóbb megértésére. Legfőbb célom mégis az volt, hogy egy olyan játékot készítsek, melyre a jövőben is szívesen emlékszem vissza és bármikor elő tudjam venni, hogy egy nagyszerű játékélménnyel gazdagodjak. A projektet Xilinx ISE Design Suite 14.7-es verzióban fejlesztettem.

Konfiguráció a használathoz

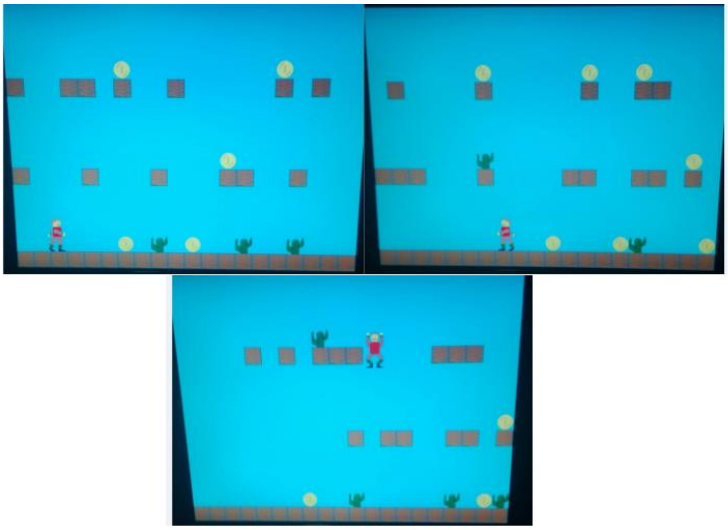
A játékhoz szükség van egy VGA bemenetű monitorra és egy azt támogató kiegészítő kártyára az „A” bővítőcsatlakozóra kötve. Hangszóró használatához tegyük megfelelő állásba a jumpert.

Játék leírása

A játék során a felhasználó a kártyán található nyomógombokkal vagy a navigációs gombokkal irányít egy karaktert egy kétdimenziós oldalnézetes grafikájú pályán. A cél minél több érme összegyűjtése adott idő alatt és ellenséges objektumok (kaktuszok) kikerülése. A karakter jobbra, balra tud mozogni, illetve lehet ugrani is. Járófelületként különböző magasságokban vannak platformok és ezeken helyezkednek el az érmék és a kaktuszok. Ezek az objektumok véletlenszerűen vannak elhelyezve és ha a karakter végigmegy egy pályán, akkor egy újabb pálya generálódik. A játéknak akkor van vége, amikor nekimegy a karakter egy kaktusznak vagy letelik egy előre meghatározott időkeret, alapesetben 80 másodperc. A játék végét egy üres kék háttér jelzi. A panelon folyamatosan látható az eddig megszerzett pontszám a kétszámjegyes hétszegmenses kijelzőn és a hátralévő idő a nyolc ledből álló csíkon.



1. ábra Elért pontszám és hátralévő idő kijelzése



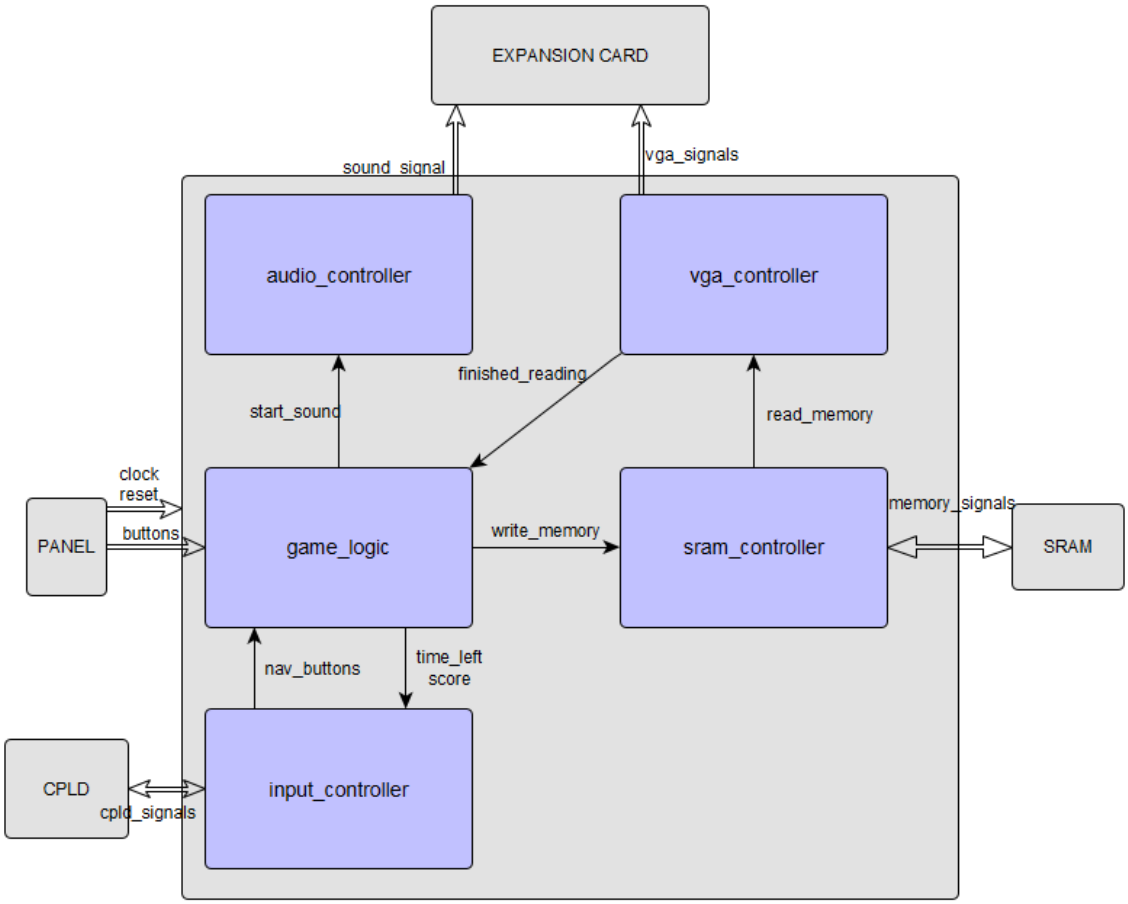
2. ábra Néhány példa a 1,8 * 10^{24} darab lehetséges pályából

Modulok és kapcsolataik

Főbb modulok

A legfelsőbb szinten öt darab modult hoztam létre:

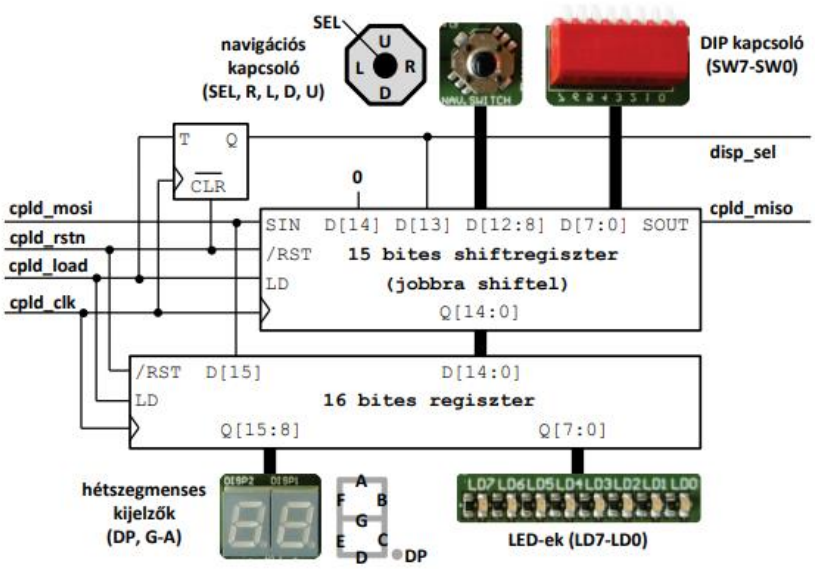
- input_controller: a CPLD IC–vel való kommunikációért felelős, panelen lévő bemeneteket, kimeneteket vezérli.
- sram_controller: külső SRAM-ot illeszti.
- vga_controller: VGA monitor illesztését oldja meg.
- logic_controller: a játék működéséért felelős.
- audio_controller: bővítőkártyán lévő hangszóró modult működteti (IMSc feladat).



3. ábra Egyszerűsített blokkvázlat

Perifériavezérlő

A játékhoz használjuk a navigációs gombokat, a LED-eket és a kétszámjegyes hétszegmenses kijelzőt. Ezek egy soros interfészen keresztül csatlakoznak az FPGA-hoz, melyet egy CPLD IC vezérel. Az input_controller modul ezzel az IC-vel kommunikál, feldolgozza a cpld_miso jelet és kiadja a vezérlő jeleket. A többi modul közül a logikát vezérlő modullal kommunikál.

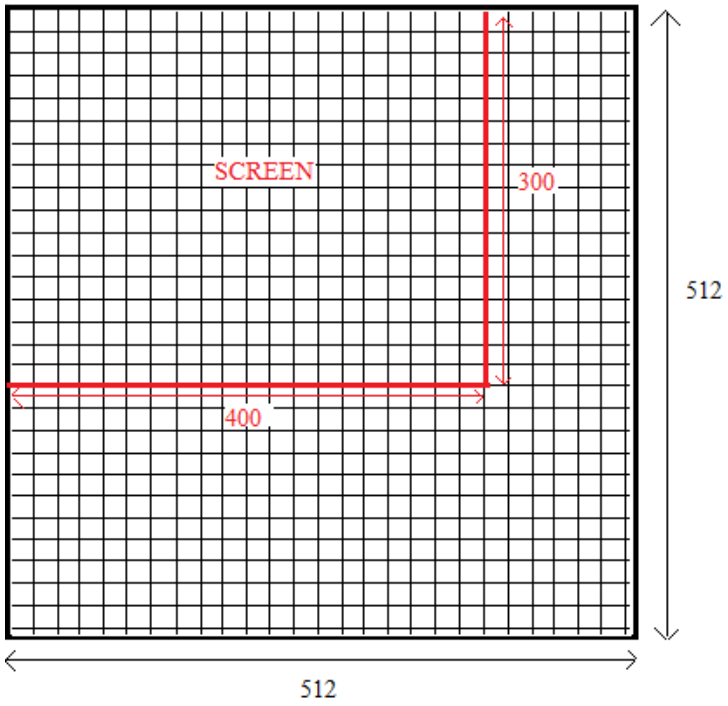


4. ábra CPLD logikája

Memóriavezérlő

A sok adat tárolására felhasználtam a panelen lévő külső statikus RAM-ot. Ennek a kapacitása 256k x 16 bit (512 kB) és ebből csak ~88 kB-ot használok fel. Itt tárolom a monitoron megjelenítendő pixelinformációkat. 400x300-as felbontást és 6 bites színmélységet használok. A monitort a memóriába könnyű leképezésének érdekében a 18 bites címvezetéket kettő darab x és y koordinátát kiválasztó 9 bites vezetékre bontottam fel, és mindegyik címen a 16 bitből csak az alsó 6 bitet használom. Az adatvezeték kétirányú, tehát olvasáskor magasimpedanciás állapotba kell tennünk. A memóriát aszinkron írom és olvasom.

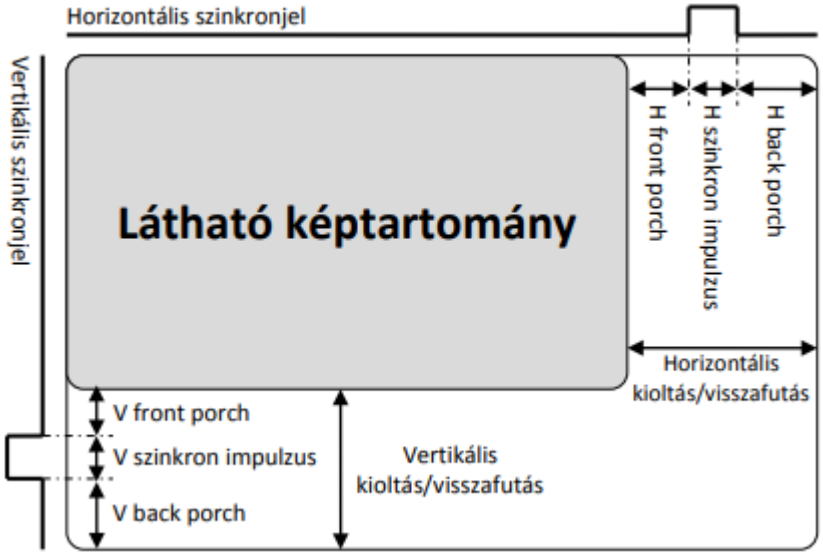
```
assign address = {mem_addr_y, mem_addr_x};
assign data_wire = read ? 16'bz : data_to_write;
```



5. ábra Képernyő leképezése az SRAM-ba

Monitorvezérlő

VGA monitor működtetése két részre osztható. Megfelelő időkben ki kell adni adni egyes pixelekre a megjelenítendő színt, illetve ezek után szinkron impulzusokat kell küldeni. 800x600-as monitorfelbontást használok, amit 72 Hz-en frissítek. Ekkor 50 MHz a pixel órajel, ami azt adja meg, hogy egy pixelre a színinformációt milyen sűrűn küldjem. Mivel az FPGA oszcillátora is 50 MHZ-es órajelet küld, ez könnyen megvalósítható volt. A játékot 400x300-as felbontásban rajzolom ki, ez azt jelenti, hogy 1 logikai pixelem 2x2 valódi pixel lesz. A vezérléshez kettő számlálót hoztam létre, ami bejárja az egész VGA képet. A vízszintes számláló 0-tól 1039-ig, a függőleges számláló 0-tól 665-ig számol. Amikor pixelértékeket kell kiadni ($x < 800$ és $y < 600$), akkor folyamatosan olvassa a memóriát, amikor nem látható tartomány van, akkor pedig egy jelet küld, hogy nem olvassa a memóriát, tehát lehet írni. A memóriában a megfelelő színértéket egy órajellel hamarabb olvasom, mint ahogy kiküldöm a késleltetések miatt. Ha a függőleges számláló elérte a 600-at, küldök egy jelet a játékvezérlőnek, hogy sokáig írható a memória (1,37 ms-ig), így a megjelenített kép nem fog szaggatottan látszódni.

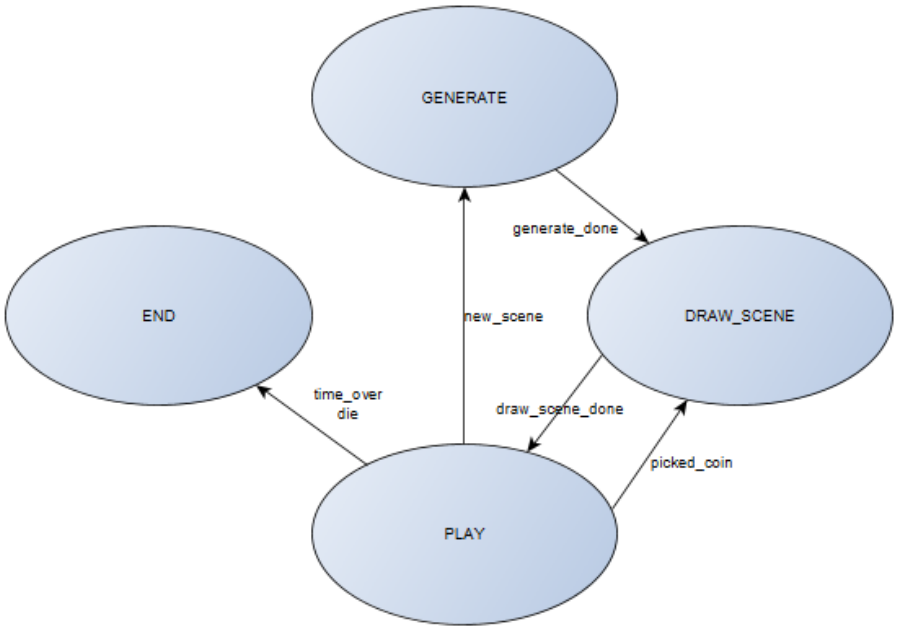


6. ábra VGA kép felépítése

Játékvezérlő

A játék logikájáért felelős egység a legnagyobb és legösszetettebb modul mind közül. Folyamatosan kommunikál a memória-, a hangszóró-, és a perifériavezérlővel. Ezen kívül kell mozgatnia, animálnia a karaktert, észlelnie az esetleges ütközéseket más objektumokkal, generálnia a pályákat (scene), eltárolni a memóriába az objektumok pozícióját. A logika lényegében egy négyállapotú állapotgép.

- Pálya generálása (kezdeti állapot)
- Pálya kirajzolása
- Animálás, mozgatás, ütközésészlelés
- Játék vége (végállapot)



7. ábra Állapotátmenetek

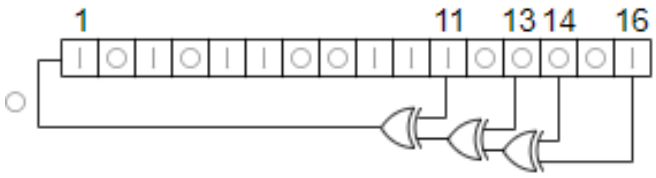
Pálya generálása

Minden egyes pálya gyakorlatilag egy 20x15-ös mátrix, ahol minden cellában vagy nincs semmi, vagy kaktusz, vagy érme, vagy platform van. A felbontás segítségével kiszámítható, hogy egy cella 20x20 pixel. A jelenlegi pályát a Block RAM-ban (scene_memory modul) tárolom, és dual port-ként működtetem aszinkron olvasással és szinkron írással. Egy ilyen mátrix 75 bájtot foglal el. Generáláskor és érmefelvételkor ezt a memóriát írom, rajzoláskor és ütközésészlelésnél pedig olvasom. A karakter pozíciója nem itt van eltárolva.

A generálást a scene_generator modul vezérli. A művelet során két számláló bejárja a mátrixot és egy véletlenszám-generátor segítségével írja a memóriát. A memóriacím két koordináta és a szélesség segítségével könnyen megadható.

```
assign generate_addr = scene_gen_y * SCENE_BLOCK_WIDTH + scene_gen_x;
```

A pálya legalsó sorában biztos, alulról az 5. és 10. sorban pedig egyharmad eséllyel van platform. Minden egyes platformon ~6% eséllyel van kaktusz (kivéve a pálya legelejét), ~8.5% eséllyel van érme. Egyszerre egy helyen csak egy objektum lehet. Pszeudo véletlen számokat egy 16 bites visszacsatolt shift regiszterrel oldottam meg, melynek a soros bemenete megfelelő pozíciók egy kizáró vagy (XOR) függvénye.



8. ábra Megvalósított 16 bites LFSR

9. ábra A shift regiszter alsó négy bitjének szimulációja

Pálya kirajzolása

Új pálya generálása után szükséges nekünk azt valahogy a monitorra kirajzolni. Ezért a game_drawer modul felel. A VGA vezérlő az SRAM-ot folyamatosan olvassa a pixelinformációkért, ezért nekünk oda kell írnunk az új adatot. A karakter kirajzolásával jelenleg még nem törődünk. Az SRAM-ba a megjelenítendő képet négy számláló segítségével írjuk. Ebből kettő bejárja a memóriában lévő pályát, a másik kettő pedig a cellák pixeleit. Az egyes objektumokhoz tartozó pixelszíneket külső fájlokban tárolom, melyeket szintén a Block RAM-ba olvasok. Memóritakarékosság érdekében nem 6 bites színeket tárolok hanem 2 vagy 3 bites információt, mivel egy sprite-ban legfeljebb 8-féle szín van. Ezeket az adatokat egy case szerkezetekben képzem le tényleges színekké. Ha egy sprite-nak egy része átlátszó, tehát a mögötte lévő alakzatnak a színeit kell megjeleníteni, azt is jelzem és nem engedélyezem a memória írását. Ezt a funkciót azonban nem használom ki.

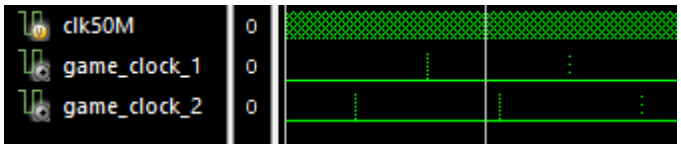
Játék logikája

A PLAY állapotba tartozik többek között az ütközésészlelés, karakter animálása, mozgatása és kirajzolása. Összesen négy darab engedélyezőjel generátorra van szükségünk. Kell egy 1 Hz-es, a másodpercszámláláshoz (az egész játék 80 másodpercig tart), egy 10 Hz-es az animációhoz és kettő 70 Hz-es a mozgáshoz. Az utóbbi kettő generátor gyakorlatilag egy 140 Hz-es engedélyezőjelet ad, csak szét van szedve két jelre. Ennek az okára a későbbiekben térek ki.

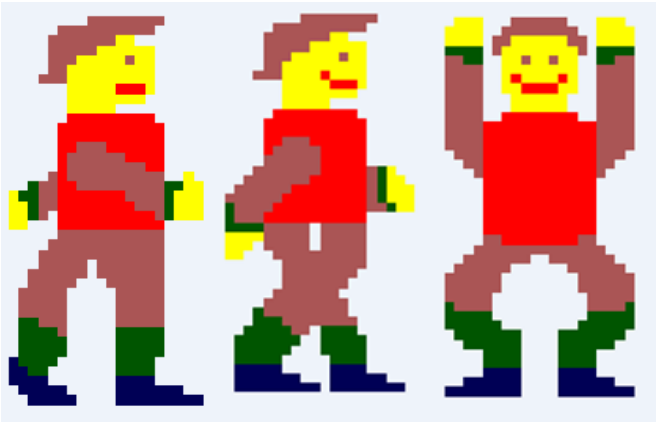
Az ütközésészlelés 140 Hz-es frekvenciával fut, ehhez szükség van a karakter pozíciójára és a pályára. A collision_detector modul jelzi, ha a karakter egy kaktusszal, érmével ütközött, vagy melyik irányban van egy pixelre a pálya széle vagy egy platform. Ezt szokásos módszerrel oldottam meg, hogy két számlálóval végigmegyek az

egész pályán és minden objektumra megvizsgáltam az ütközést. Ha kaktusszal ütközik akkor véget vetek a játéknak és átvált az állapotgép END állapotba. A pálya jobb szélén való túljutás esetén GENERATE állapotba váltunk és egy újabb pályát generálunk. Érme esetén pedig kitöröljük a memóriából azt és DRAW_SCENE állapotba ugrunk, ahol újra kirajzoljuk az egész pályát, megnöveljük a pontszámot és jelzünk a hangszóróvezérlőnek.

A gombok és az akadályok függvényében mozgatjuk a karaktert 70 Hz-es frekvenciával, azonban horizontálisan és vertikálisan külön-külön frissítjük. Ezt azért tesszük így, mert ütközészlelésnél nem figyelünk arra, ha átlósan vagyunk egy pixel távolságra egy platformtól. Ha nem mozgunk átlósan, hanem először függőlegesen és után víszintesen akkor nem tudunk „beleesni” az objektumba. A karakter csak akkor tud ugrani, ha van alatta platform és addig emelkedik a levegőben, amíg nyomva tartja a felhasználó a gombot, azonban van egy maximum emelkedési magasság, egy úgynevezett ugráspotenciál, ami emelkedés során folyamatosan, plafonnal való érintkezés során tízszer jobban csökken, földre érkezés után pedig hirtelen feltöltődik. Mozgás során tároljuk, hogy jobbra vagy balra néz a karakter, illetve folyamatos mozgás során 0,1 másodpercenként más-más sprite-ot jelenítünk meg.



10. ábra Két 70 Hz-es engedélyező jel szimulációja



11. ábra 3 darab karakter sprite, melyek tükrözve is megjelenhetnek a képernyőn

A karakter rajzolása teljesen független a pálya kirajzolásától. PLAY állapotban a game_drawer modul vagy törli a karakter régi pozícióját, vagy kirajzolja a karaktert,

vagy várakozik. Akkor kezdi el az egész folyamatot újra, amikor a vga_controller egység jelzi, hogy most sok ideig írható a memória, így nem fog villódzni a monitor. Ez 72 Hz-es frekvenciával történik. A karakternek a sprite-jai ugyanúgy vannak eltárolva, mint ahogy a többi objektumnak.

Végállapot

Amint lejárt az idő, vagy a karakter nekiment egy kaktusznak, átváltunk END állapotba és egy kék színű hátteret jelenítünk meg. Az elért pontszám meg van jelenítve a hétszegmenses kijelzőkön, és reset gomb hatására újraindítható a játék.

Hangszóróvezérlő (IMSc feladat)

A tervezett feladaton kívül IMSc részfeladatom volt egy egyszerű audio bővítmény kialakítása.

A játék indulásakor, illetve minden érme felvétele után (ezt a játékvezérlő jelzi) a bővítőkártyán lévő hangszórómodulnak küldök egy magas csilingelő hangot impulzusszélesség modulált digitális négyszögjellel. Moduláció során az amplitúdót a kitöltési tényező módosításával tudom változtatni. A hangerősség 7 biten kvantált, de ennyi is elegendő a célra. A hangeffekt amplitúdóértékeit egy külső fájlban tárolom, amit majd a Block RAM-ba töltök. A memóriába ezt rögtön betöltöm és futás alatt csak olvasni fogom. Próbáltam minél takarékosabban bánni a memóriával, ezért különböző módszerek által a hangeffekt csak 112 bájtot foglal.

Észrevettem, hogy a 127, 0, 126, 1, 125, 2, ..., 65, 62, 64, 63 sorozat megfelelő hanghatást ad, ha minden párt 16-szor megismétlek 5120 Hz frekvenciával. Tehát ehhez szükségem volt még egy kb. 5 kHz-es engedélyező jelgenerátorra. A memóriába csak az ismétlés nélküli sorozatot tárolom el, és egy számlálóval megoldottam a párok ismétlését. Ha megvan az amplitúdó akkor egy másik 7 bites számlálóval egyszerű a modulált jel kiküldése.

```
reg [6:0] counter = 0;
always @ (posedge clk)
     if(rst)
          counter <= 0;
     else
          counter <= counter + 1;
assign sound = (counter < duty);
```

12. ábra Használt hangeffekt amplitúdó-idő függvénye

Erőforrásigények

Részlet a leképezés jelentéséből:

```
Design Summary
--------------
Number of errors:      0
Number of warnings:    0
Slice Logic Utilization:
  Number of Slice Registers:                  308 out of  11,440    2%
    Number used as Flip Flops:                306
    Number used as Latches:                     0
    Number used as Latch-thrus:                 0
    Number used as AND/OR logics:               2
  Number of Slice LUTs:                       868 out of   5,720   15%
    Number used as logic:                     831 out of   5,720   14%
      Number using O6 output only:            576
      Number using O5 output only:            115
      Number using O5 and O6:                 140
      Number used as ROM:                       0
    Number used as Memory:                     20 out of   1,440    1%
      Number used as Dual Port RAM:            20
        Number using O6 output only:           20
        Number using O5 output only:            0
        Number using O5 and O6:                 0
      Number used as Single Port RAM:           0
      Number used as Shift Register:            0
    Number used exclusively as route-thrus:    17
      Number with same-slice register load:     6
      Number with same-slice carry load:       11
      Number with other load:                   0
```

Észrevehető, hogy minimálisan vannak kihasználva az erőforrások, a beépített Block RAM helyett a külső SRAM-ot használtam nagyméretű adat tárolására.

Összefoglalás

A félév során elértem a játékkal kapcsolatos céljaimat, az előzetes tervet túlteljesítettem, mindent sikerült implementálnom, amit szerettem volna. Szerencsére nem ütköztem nagyobb nehézségekbe, jól haladtam a feladattal. Magabiztosan használom a Verilog nyelvet és talán a jövőben is szeretnék digitális hardvertervezéssel foglalkozni.

A projekt megtalálható a személyes drive-omon:

https://drive.google.com/open?id=12DNk35ZFl6BoFts7VYnIuSZSlZQqBXmd

Hivatkozások

LOGSYS Spartan-6 FPGA kártya felhasználói útmutató

http://logsys.mit.bme.hu/sites/default/files/page/2009/09/LOGSYS_SP6_FPGA_Board.pdf

LOGSYS VGA, PS/2 és hangszóró modul felhasználói útmutató

http://logsys.mit.bme.hu/sites/default/files/page/2009/09/LOGSYS_SP6_FPGA_Board.pdf

Lineárisan visszacsatolt shift regiszter angol Wikipedia oldala

https://en.wikipedia.org/wiki/Linear-feedback_shift_register

Platform játék fogalma

https://web.archive.org/web/20070219082328/http://www.specusphere.com/joomla/index.php?option=com_content&task=view&id=232&Itemid=32

Mozgás illúzióját keltő animáció készítése

https://www.futurelearn.com/courses/explore-animation/0/steps/12222

Sprite-ok készítését segítő oldal

https://www.pixilart.com/

Impulzusszélesség moduláció

https://barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation

Hangeffekt amplitúdófüggvénye

https://freesound.org/people/ProjectsU012/sounds/341695/