Joc de memorie

Pal Roland
Universitatea Tehnica din Cluj-Napoca
Facultatea Automatica si Calculatoare
Specializarea Calculatoare
Seria B
Grupa 30238
Indrumator: Muresan Mircea Paul

January 2026

Cuprins

1 Rezumat

Proiectul a constat în realizarea unui joc de memorie implementat pe placa Nexys4 Artix-7, în care sistemul a afișat pe display-ul cu 7 segmente o secvență de cifre a cărei lungime a depins de nivelul curent al jocului, iar utilizatorul a introdus apoi secvența de la tastatura Pmod KYPD, în aceeași ordine. Cifrele au fost generate pseudo-aleator, iar la introducerea corectă a secvenței jocul a avansat la nivelul următor, în timp ce o introducere greșită a determinat reluarea nivelului. Implementarea s-a realizat în VHDL pentru descrierea logicii hardware (afișarea pe SSD, controlul secvenței, citirea de pe KYPD, generarea numerelor pseudo-aleatoare, logica de validare și schimbarea nivelului), iar Python a fost folosit pentru preluarea prin UART a informațiilor transmise de pe placă și afișarea pe PC a detaliilor despre nivelul curent și starea jocului. Rezultatul a fost un joc, capabil să genereze și să afișeze secvențe, să verifice corectitudinea introducerii și să gestioneze progresul în niveluri. În concluzie, proiectul a demonstrat că un joc interactiv poate fi implementat eficient în hardware.

2 Introducere

În proiectul acesta am realizat un joc de memorie pe o placă FPGA (Nexys4 Artix-7). Ideea jocului este simplă: placa afișează o secvență de cifre pe display-ul cu 7 segmente, iar utilizatorul trebuie să introducă aceeași secvență în aceeași ordine, folosind o tastatură 4x4 conectată pe PMOD. Dificultatea crește pe măsură ce utilizatorul avansează, prin mărirea lungimii secvenței și prin scăderea timpului de afișare al cifrelor. Jocul este interactiv și rulează direct în hardware, iar pe PC sunt afișate informații suplimentare despre nivel și rezultat, printr-o conexiune serială UART.

În ultimii ani, plăcile FPGA au devenit tot mai folosite în proiecte practice, mai ales când este nevoie de control clar asupra intrărilor/ieșirilor și de comportament determinist. Spre deosebire de un program care rulează pe un procesor general, pe FPGA logica este implementată ca circuite care lucrează în paralel: afișarea, citirea tastelor și verificarea secvenței pot funcționa în același timp. Pentru un joc de memorie, acest lucru este util deoarece permite o reacție rapidă la input și o sincronizare stabilă a afișării, fără influențe din partea unui sistem de operare.

Domeniul proiectului este cel de proiectare digitală și sisteme embedded, unde aplicația este construită din module hardware care comunică între ele prin semnale. În această lucrare apar mai multe noțiuni de bază folosite frecvent pe plăci de dezvoltare: display cu 7 segmente (SSD) pentru afișarea cifrelor, tastatură matricială 4x4 (KYPD) pentru introducerea secvenței, un generator pseudo-aleator pentru a crea secvențe diferite, și o comunicație serială UART pentru a transmite informații către PC. Aceste elemente sunt importante deoarece permit realizarea unei interfețe complete (afișare + input + feedback) cu resurse relativ mici.

Problema principală rezolvată de proiect a fost construirea unui sistem complet, care să genereze o secvență de cifre, să o afișeze corect pe SSD, să citească apăsările utilizatorului de pe tastatura 4x4, să compare secvențele și să decidă rezultatul (trecere la nivelul următor sau reluarea nivelului). În plus, a fost urmărită și transmiterea unor informații către PC, astfel încât jocul să fie mai ușor de urmărit și testat, nu doar prin display-ul plăcii.

Obiectivele proiectului au fost:

- implementarea unui flux complet de joc (afișare secvență, input utilizator, verificare, rezultat)
- creșterea dificultății pe niveluri (secvență mai lungă și afișare mai rapidă)
- citirea corectă a unei tastaturi matriciale 4x4 prin scanare rând/coloană

- afișarea stabilă pe SSD folosind multiplexare și temporizări potrivite
- generarea de secvențe diferite folosind o metodă potrivită pentru hardware (pseudo-aleator)
- trimiterea prin UART a informațiilor către PC și afișarea lor într-un program Python.

Soluția propusă a fost realizarea jocului ca o mașină de stări finite (FSM), deoarece acest model se potrivește pentru un sistem care trece prin etape. Jocul are o stare în care se afișează secvența, o stare în care se așteaptă inputul utilizatorului, o stare de verificare și stări de rezultat (câștig/pierdere). Separarea pe stări ajută la organizarea logicii și la evitarea situațiilor în care mai multe acțiuni se amestecă între ele. De asemenea, această abordare permite extinderi ușoare, de exemplu adăugarea unui reset, a unei pauze sau a unor efecte de afișare suplimentare.

Un punct important al soluției a fost modul în care a fost realizată interfața cu utilizatorul, adică SSD + KYPD. În cazul afișajului cu 7 segmente, este necesară multiplexarea mai multor digit-uri, iar pentru ca afișarea să fie stabilă trebuie aleasă o frecvență suficient de mare. În cazul tastaturii, este necesară scanarea rândurilor și coloanelor pentru a identifica tasta apăsată, iar în practică avem nevoie și de un debounce, ca să se evite citiri repetate ale aceleiași taste. Toate aceste lucruri sunt tehnici standard, dar integrarea lor într-un joc care are niveluri diferite aduce provocări legate de sincronizare și control.

Pentru generarea secvenței s-a folosit o metodă potrivită pentru hardware, de tip LFSR (Linear Feedback Shift Register). Un LFSR este des întâlnit în proiecte FPGA deoarece se implementează ușor, folosește puține resurse și poate genera o secvență lungă de valori care par suficient de „aleatoare” pentru aplicații precum jocuri sau testare. Scopul aici nu a fost obținerea unui random perfect, ci obținerea unor secvențe diferite, astfel încât jocul să nu devină repetitiv.

Față de o implementare strict software, soluția de aici rulează direct în hardware, iar afișarea și citirea input-ului au loc în timp real, cu timpi controlați de ceasul plăcii. În plus, integrarea unei conexiuni UART către PC oferă o metodă simplă de a afișa mesaje și informații suplimentare (nivel, rezultat, scor maxim), ceea ce ajută atât utilizatorul, cât și partea de testare și depanare. Această combinație (joc în hardware + monitorizare pe PC) face proiectul mai ușor de urmărit.

În cadrul implementării au fost folosite următoarele instrumente și limbaje: Vivado pentru proiectare și implementare pe FPGA, VHDL pentru

descrierea modulelor hardware, și Python pe PC pentru citirea portului serial și afișarea mesajelor. Toate acestea au fost alese deoarece sunt soluții uzuale și bine documentate pentru astfel de proiecte: VHDL este potrivit pentru logica sincronă, iar Python este simplu de folosit pentru interfața serială și pentru afișarea rapidă a informațiilor.

În continuare, în secțiunea Fundamentare teoretică sunt prezentate noțiunile de bază folosite în proiect (SSD, tastatură matricială, LFSR, UART și FSM). În secțiunea Proiectare și implementare este descrisă soluția aleasă, arhitectura generală și rolul fiecărui modul, împreună cu deciziile de proiectare. În secțiunea Rezultate experimentale sunt prezentate testele, rezultatele obținute pe placă și observațiile legate de funcționare. În final, secțiunea Concluzii sintetizează ce s-a obținut, ce limitări au apărut și ce îmbunătățiri se pot face pe viitor.

3 Fundamentare teoretică

Proiectul a fost realizat pe un FPGA, o placă pe care se poate implementa direct logică hardware folosind cod. Față de un program obișnuit care rulează pe procesor, pe FPGA mai multe funcții pot lucra în paralel: afișarea, citirea tastaturii și verificarea secvenței pot rula simultan, fiind module separate. Din acest motiv, FPGA-ul este potrivit pentru aplicații în care contează timpii stabili și răspunsul rapid la input, fără dependență de un sistem de operare. (Surse: [1] Wikipedia – FPGA; [2] AMD/Xilinx – materiale introductive despre FPGA; [3] Digilent – manualul plăcii Nexys4.)

Pentru descrierea logicii hardware am folosit VHDL, un limbaj dedicat proiectării digitale. În VHDL nu se scrie o succesiune de pași ca într-un program clasic, ci se descriu semnale, registre și modul în care acestea se actualizează în timp, de obicei sincronizat cu semnalul de ceas. De exemplu, modulul de afișaj și modulul de citire a tastaturii funcționează în paralel, iar schimbul de informații dintre ele se face prin semnale. În acest proiect, acest mod de lucru a fost important deoarece jocul include mai multe componente care trebuie să coopereze: generarea secvenței, afișarea pe SSD, citirea de la KYPD, verificarea și transmiterea informațiilor prin UART. (Surse: [9] Wikipedia – VHDL; [2] materiale AMD/Xilinx despre design sincron.)

Organizarea jocului s-a făcut folosind o mașină de stări finite (FSM). O FSM presupune că sistemul se află într-o stare, execută acțiunile corespunzătoare acelei stări, iar apoi trece într-o altă stare atunci când apare o condiție. În cadrul jocului, stările folosite sunt ușor de urmărit: SHOW pentru afișarea secvenței, INPUT pentru introducerea utilizatorului, CHECK pentru verificare, iar WIN/LOSE pentru afișarea rezultatului. Avantajul

acestui model este că logica rămâne clară și ușor de controlat, iar proiectul poate fi extins mai simplu prin adăugarea unor stări noi. (Surse: [6] Wikipedia – Finite-state machine;)

Afișarea s-a realizat pe display-ul cu 7 segmente (SSD) al plăcii. Un astfel de display folosește segmentele a,b,c,d,e,f,g pentru formarea cifrelor, prin aprinderea combinațiilor corespunzătoare. Placa Nexys include mai multe poziții de afișare (digit-uri), iar controlul se face prin multiplexare: digit-urile sunt activate pe rând, foarte rapid, iar ochiul percepe afișarea ca fiind continuă. Pentru ca afișajul să fie stabil, frecvența de multiplexare trebuie să fie suficient de mare, motiv pentru care se folosește un divizor de ceas sau un timer. (Surse: [3] Digilent – manual Nexys4 (SSD); [7] Wikipedia – Seven-segment display;)

Introducerea secvenței s-a făcut cu o tastatură matricială 4x4 (Pmod KYPD). Aceasta este organizată pe rânduri și coloane. Identificarea tastei apăsate se face prin scanare: se activează pe rând fiecare coloană și se citesc rândurile; dacă un rând este activ în timp ce o coloană este activată, tasta de la intersecția lor este apăsată. După detecție, combinația rând–coloană se decodează în cifra sau litera corespunzătoare. În practică este necesară și o formă de filtrare a apăsării (debounce sau detectare de front) pentru a evita citirea multiplă a aceleiași taste. (Surse: [4] Digilent – Pmod KYPD reference manual; tutoriale despre scanarea tastaturilor matriciale; articole despre debounce.)

Pentru generarea secvenței de joc am folosit un LFSR (Linear Feedback Shift Register), o metodă uzuală în hardware pentru generarea de valori pseudo-aleatoare cu resurse reduse. Un LFSR folosește un registru de biți care se deplasează (shift), iar bitul nou se obține prin operații XOR între anumite poziții ale registrului. Dacă valorile sunt alese corect, secvența generată are o perioadă mare și este suficient de variată pentru aplicații precum jocuri. Totuși, LFSR nu este un generator aleator real și nu este destinat aplicațiilor de securitate, însă pentru un joc de memorie este adecvat deoarece oferă secvențe diferite fără o implementare complicată. Din starea LFSR se poate obține apoi o cifră, de exemplu prin selectarea unor biți și maparea lor în intervalul [0, 9]. (Surse: [8] Wikipedia – Linear-feedback shift register; exemple de implementare LFSR în VHDL.)

Pentru a transmite informații de la placă la PC am folosit UART. UART este o comunicație serială simplă, folosită frecvent pe plăci de dezvoltare, care utilizează liniile TX și RX și un baud rate stabilit (de exemplu 9600 sau 115200). Datele sunt transmise sub formă de bytes, folosind formatul clasic cu start bit, biți de date și stop bit. Avantajul UART este că se implementează relativ ușor și este foarte util pentru afișarea de informații suplimentare (debug și monitorizare), mai ales când pe placă există un afișaj

limitat. Pe PC, un program Python a citit datele seriale și a afișat mesaje despre nivel și evoluția jocului, inclusiv un scor maxim raportat pana in acel moment. (Surse: [5] Wikipedia – UART; [10] documentația pyserial (Python);)

Elementul specific al lucrării nu este inventarea acestor tehnologii (SSD, KYPD, LFSR, UART), deoarece ele sunt bine cunoscute, ci modul în care au fost integrate într-un sistem complet care funcționează ca joc de memorie. Proiectul combină generarea secvenței, afișarea, introducerea de la utilizator, verificarea și trecerea între niveluri, toate coordonate printr-o FSM, iar în plus oferă feedback către PC prin UART. Astfel, lucrarea a urmărit în special integrarea corectă a modulelor și obținerea unui comportament stabil, ușor de urmărit și testat. (Surse: ghidul/cerințele proiectului; manuale de laborator FPGA; [3] documentații Digilent/AMD-Xilinx pentru placa și modulele folosite.)

4 Proiectare și implementare

4.1 Metoda experimentală utilizată

Jocurile de memorie sunt o metodă bună pentru a îmbunătăți memoria și concentrarea. Oamenii le folosesc atât pentru a isi dezvolta capacitatea mintala de memorie cât și pentru a se distra. Proiectul a fost realizat ca o implementare hardware pe placa Nexys4 Artix-7. Logica jocului a fost descrisă în VHDL și implementată în Vivado 2024.2, iar pentru partea de monitorizare pe PC am folosit un script Python (rulat din PyCharm) care citește datele trimise prin UART și afișează mesaje despre nivel și rezultat. Testarea s-a făcut atât prin verificări pe placă (afișaj + tastatură), cât și prin verificarea mesajelor primite pe PC prin serial.

4.2 Cerințe și funcționalități urmărite

Sistemul trebuie să funcționeze ca un joc de memorie cu niveluri, cu următoarele funcționalități:

- generarea unei secvențe de cifre pentru nivelul curent;
- afișarea secvenței pe SSD (7 segmente), cu o viteză care crește de la nivel la altul;
- citirea cifrelor introduse de utilizator de la tastatura Pmod KYPD conectată pe PMOD JA;

- compararea secvenței generate cu secvența introdusă (în aceeași ordine);
- la success: trecerea la nivelul următor și afișarea unui simbol de WIN;
- la greșeală: rămânerea la același nivel și afișarea unui simbol de LOSE, apoi reluarea nivelului;
- trimiterea prin UART către PC a unor informații (nivel, WIN/LOSE), ca să se poată afișa mesaje și best score în aplicația Python.

4.3 Soluția aleasă și alternative de proiectare considerate

Am ales să implementez jocul ca o combinație între o FSM (pentru controlul etapelor), un generator LFSR (pentru secvențe diferite) și module separate pentru SSD, KYPD și UART, deoarece această structură e clară și ușor de controlat în hardware. O variantă alternativă ar fi fost folosirea unor secvențe (de exemplu într-un ROM), dar asta ar fi făcut jocul previzibil după mai multe încercări. Am ales varianta cu generator pseudo-aleator bazat pe LFSR, deoarece se implementează ușor în hardware și oferă secvențe diferite fără memorie mare.

Pentru controlul jocului, se putea scrie logica direct cu multe condiții și semnale separate, însă ar fi fost mai greu de urmărit și de testat. Am ales organizarea ca FSM (mașină de stări), deoarece jocul are etape foarte clare (afișare, input, verificare, rezultat), iar FSM-ul păstrează totul ordonat.

Pentru feedback, se putea rămâne strict la ce arată placa (SSD/LED-uri), dar am ales să trimit date prin UART la PC, fiindcă așa pot afișa mesaje mai clare și pot ține și un best score global în Python.

4.4 Arhitectura generală a sistemului

Arhitectura a fost împărțită în câteva module principale, legate de un modul de control (FSM). În linii mari, fluxul este: FSM-ul pornește generarea secvenței, apoi comandă afișarea pe SSD, după care așteaptă input de la KYPD, verifică rezultatul și intră în starea de WIN sau LOSE. La finalul rundei, se trimite prin UART către PC informația despre rezultat și nivel.

Modulele principale folosite au fost:

- **GameFSM (top_uart.vhd)**: componenta principală a jocului, decide stările și tranzițiile (IDLE, PRE_GEN, GEN, TRIGGER_SSD, SHOW, INPUT, CHECK, WIN, LOSE);

- **Sequence Generator - LFSR (random_digits_gen.vhd)**: generează cele N cifre pentru nivel;
- **SSD_Controller (ssd_divider.vhd)**: afișează cifrele secvenței și simbolurile de WIN/LOSE;
- **KYPD_Controller (kypd_controller.vhd)**: citește tastatura prin scanare și scoate cifra apăsată;
- **UART_TX (uart_tx.vhd)**: trimite către PC date despre nivel și rezultat;
- **Python Receiver**: citește serial, afișează mesaje și ține scorul cel mai bun de până atunci (best score).

4.5 Descrierea modulelor

4.5.1 Modulul de control al jocului (Game_FSM)

Controlul jocului a fost realizat cu o mașină de stări finite (FSM), pentru că jocul are pași clari și repetitivi: inițializare, generare secvență, afișare, input utilizator, verificare și apoi rezultat. În implementarea mea, FSM-ul folosește următoarele stări: `IDLE, PRE_GEN, GEN, TRIGGER_SSD, SHOW, INPUT, CHECK, WIN, LOSE`.

Starea `IDLE` este punctul de start. Aici sistemul nu generează nimic și așteaptă cererea de start (`start_req`). Când utilizatorul pornește jocul, se resetează indexul de input (`user_index`) și se trece mai departe.

Starea `PRE_GEN` este o stare scurtă (un ciclu de ceas) folosită ca „pauză" între acțiuni, ca să se stabilizeze semnale precum `num_digits` (numărul de cifre pentru nivel) și ca să evit generarea prea rapidă sau repetată. În această stare se activează semnalul `gen_start`, iar apoi se trece în `GEN`.

În starea `GEN`, modulul de generare a secvenței rulează până când semnalul `done_gen` confirmă că secvența a fost complet generată și salvată în array-ul de secvență (`seq_gen`). Când `done_gen` devine 1, se oprește `gen_start` și se trece către partea de afișare.

Pentru afișare există două stări separate, `TRIGGER_SSD` și `SHOW`, deoarece modulul SSD lucrează cu un handshake: trebuie să primească un impuls de start și apoi își indică starea prin semnalul `show_done`. În `TRIGGER_SSD` se forțează `show_pulse = 1` (un impuls de start), iar când SSD-ul confirmă că a preluat comanda și a intrat în modul „busy" (adică `show_done` devine 0), FSM-ul trece în `SHOW`. Astfel evit situațiile în care comanda de afișare este pierdută sau repetată.

În SHOW are loc afișarea efectivă a secvenței pe SSD. La intrarea în această stare se resetează zona de input (se pune user_index = 0 și se golește array-ul seq_user), ca să pornesc curat pentru introducerea utilizatorului. FSM-ul rămâne în SHOW până când SSD-ul termină afișarea secvenței și semnalul show_done revine la 1. În acel moment se trece în INPUT.

În INPUT, sistemul citește tastatura Pmod KYPD. Pentru a evita citirea repetată a aceleiași apăsări, folosesc detectarea de front (edge detect): se reține valoarea anterioară a semnalului key_valid în key_valid_d, iar o apăsare nouă este detectată când key_valid=1 și key_valid_d=0. Când se detectează o apăsare, codul tastei (key_value) este decodat într-o cifră/valoare (decoded_val) și este salvată în seq_user(user_index). După fiecare apăsare validă, user_index se incrementează, iar când s-au introdus exact num_digits valori, se trece în CHECK.

În CHECK, secvența introdusă (seq_user) este comparată cu secvența generată (seq_gen) element cu element, doar pe intervalul 0..num_digits-1. Dacă toate elementele sunt egale, rezultatul este considerat corect și se trece în WIN; altfel se trece în LOSE. Pentru a evita efecte nedorite, fail_mode este ținut pe 0 în această etapă.

În WIN, nivelul (nivel) este incrementat, dar doar atunci când utilizatorul apasă din nou butonul de start (start_req=1). Am ales acest lucru ca să rămână rezultatul vizibil pe SSD până când utilizatorul decide să continue. După apăsare, se revine în PRE_GEN, ca nivelul să aibă timp să se actualizeze înainte de o nouă generare.

În LOSE, se activează fail_mode=1 (pentru afișarea specifică de pierdere), iar nivelul rămâne neschimbat. La apăsarea butonului de start, fail_mode este resetat și se revine în PRE_GEN, pentru a genera o nouă secvență la același nivel.

Tranzițiile între stări au fost declanșate de condiții simple: de exemplu, SHOW se termină după ce au fost afișate toate cele N cifre, INPUT se termină când utilizatorul a introdus N cifre valide, iar CHECK decide direct WIN sau LOSE. În stările WIN/LOSE sistemul așteaptă un nou start ca să înceapă următoarea rundă.

4.5.2 Generatorul de secvență (Sequence_Generator, LFSR)

Secvența de joc a fost generată de un modul separat, care primește ca intrare câte cifre trebuie să producă (N, în funcție de nivel). Generatorul a folosit un LFSR pe 16 biți inițializat cu x"ACE1". La fiecare pas, starea LFSR este actualizată folosind taps-urile corespunzătoare polinomului $x^{16} + x^{14} + x^{13} + x^{11} + 1$. Din starea curentă s-a obținut o cifră (de exemplu prin selectarea unor biți și maparea lor în 0..9), iar cifra a fost salvată într-un array intern

care reprezintă secvența nivelului.

Funcției LFSR: Anexa A

4.5.3 Controlul afișajului (SSD_Controller)

Modulul de afișare s-a ocupat de două lucruri: multiplexarea display-ului și afișarea efectivă a cifrelor/simbolurilor. Pentru afișarea secvenței, cifrele generate sunt scoase pe SSD una câte una, în ordinea corectă. Viteza de afișare depinde de nivel: pe măsură ce nivelul crește, perioada de afișare scade. În proiect, acest lucru s-a realizat printr-o perioadă calculată de forma:

$$\texttt{period} = \texttt{BASE\_PERIOD} - \texttt{STEP\_PERIOD} \cdot \texttt{lvl}$$

Astfel, nivelurile mai mari au secvențe afișate mai rapid, ceea ce face jocul mai greu.

În starea WIN, SSD-ul afișează un simbol de câștig (linii paralele), iar în starea LOSE afișează liniuțe „-” pe toate pozițiile, ca semnal clar că secvența a fost greșită.

4.5.4 Citirea tastaturii (KYPD_Controller)

Tastatura Pmod KYPD a fost conectată la interfața PMOD JA, iar citirea s-a făcut prin scanare pe coloane. Modulul activează pe rând fiecare coloană (0..3) și citește rândurile; când un rând este activ în același timp cu o coloană activă, se determină tasta apăsată. Combinația rând–coloană este apoi decodată într-o cifră.

După fiecare apăsare validă, modulul scoate două semnale utile pentru restul sistemului:

- cifra detectată (valoarea tastei);
- un semnal de tip `valid` care spune FSM-ului că s-a apăsat o tastă nouă.

Cifrele introduse au fost salvate într-un array de input, iar un contor a ținut evidența câte cifre au fost introduse la nivelul curent.

4.5.5 Verificare (CHECK) și condițiile WIN/LOSE

După ce utilizatorul a introdus N cifre (același N ca secvența generată), sistemul trece în starea CHECK. Aici se compară cele două array-uri: cel generat și cel introdus de utilizator. Compararea s-a făcut element cu element, în ordine. Dacă toate elementele sunt identice, rezultatul este WIN; altfel, rezultatul este LOSE.

În WIN, nivelul se incrementează și se pregătește următorul nivel. În LOSE, nivelul rămâne același, iar la un nou start se generează o altă secvență pentru același nivel (același număr de cifre și aceeași viteză de afișare, conform regulilor nivelului).

4.5.6 Transmiterea prin UART și aplicația Python

Pentru feedback suplimentar, la fiecare rundă terminată (WIN sau LOSE) s-au transmis date prin UART către PC. Modulul UART a fost folosit pentru a trimite cel puțin informația despre nivel și rezultatul rundei, astfel încât aplicația Python să poată afișa mesaje despre progres.

Pe PC, scriptul Python citește datele de pe portul serial și afișează mesaje informative legate de nivelul curent și rezultat. În plus, aplicația păstrează un best score global, care reprezintă cel mai bun nivel atins până atunci.

4.6 Integrare și fluxul de funcționare

O rundă completă de joc funcționează astfel:

1. utilizatorul apasă butonul de start, iar sistemul trece în PRE_GEN, GEN, TRIGGER_SSD și SHOW;

2. în acești pași se generează secvența pentru nivelul curent și se afișează pe SSD, cu perioada stabilită de nivel;

3. după ce se termină afișarea, sistemul trece în INPUT și așteaptă N apăsări valide pe KYPD;

4. după N cifre introduse, sistemul trece în CHECK și compară secvențele;

5. dacă secvențele sunt identice: WIN, nivelul crește și se afișează simbolul de câștig; dacă sunt diferite: LOSE, nivelul rămâne și se afișează simbolul de pierdere;

6. în paralel, se trimite către PC prin UART rezultatul, iar scriptul Python afișează mesajele.

Prin separarea pe module (generator, afișaj, tastatură, UART) și coordonarea prin FSM, sistemul a rămas ușor de urmărit și de extins. De exemplu, un upgrade simplu ar fi adăugarea unui timer de timeout la INPUT sau adăugarea unui sistem de vieți.

4.7 Manual de utilizare

Pentru utilizare, placa trebuie alimentată și programată cu bitstream-ul generat în Vivado. Tastatura Pmod KYPD trebuie conectată pe PMOD JA. Utilizatorul pornește jocul prin butonul de start, urmărește secvența afișată pe SSD și apoi introduce aceleași cifre în aceeași ordine pe tastatură. Dacă secvența este corectă, sistemul afișează simbolul de WIN și trece la nivelul următor; dacă este greșită, afișează simbolul de LOSE și permite reluarea nivelului.

Pentru mesajele pe PC, se rulează scriptul Python pentru informații extra și mesaje amuzante ascunse în script.

5 Rezultate experimentale

Pentru a demonstra că sistemul a fost implementat corect și funcționează stabil, am urmărit două tipuri de rezultate: (1) rezultate din instrumentele de proiectare (sinteză/implementare) și (2) rezultate de funcționare, observate atât pe placă (SSD + tastatură), cât și pe PC (mesaje UART citite în Python). Implementarea a fost realizată și programată pe placa Nexys4 Artix-7, iar verificările au fost făcute pe mai multe runde de joc, cu niveluri diferite.

Figura 1, de mai jos, prezintă ierarhia proiectului din Vivado (Sources), unde se observă modulul de top `top_uart` și principalele componente. Modulul `num_digits_select_by_level` stabilește câte cifre trebuie generate și introduse în funcție de nivel. `kypd_controller` se ocupă de citirea tastaturii Pmod KYPD și furnizează codul tastei apăsate împreună cu un semnal de validare. `random_digits_gen` generează secvența pseudo-aleatoare de cifre (pe baza LFSR) pentru nivelul curent. Pentru afișaj, `ssd_divider` generează temporizările necesare (inclusiv viteza de afișare în funcție de nivel), iar `ssd` este folosit pentru multiplexarea și afișarea efectivă pe display-ul cu 7 segmente. În final, `uart_tx` transmite către PC informații despre rezultat și nivel, pentru afișarea mesajelor în aplicația Python.

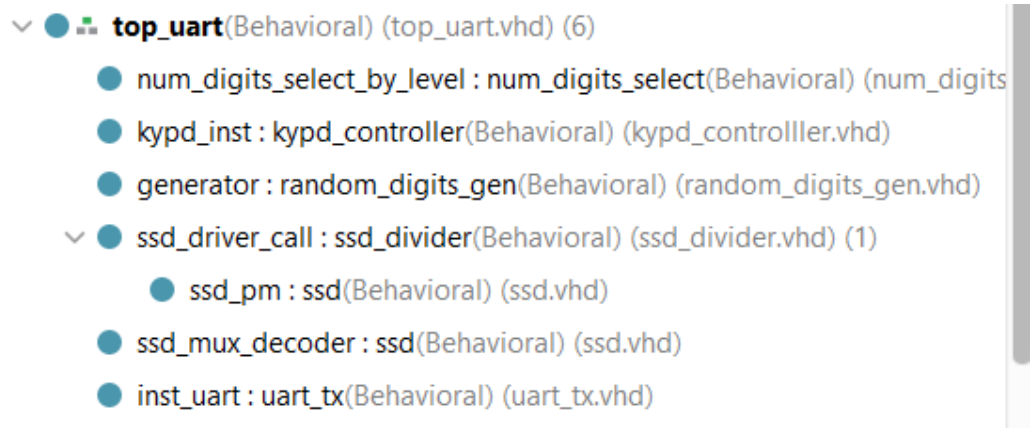

Figura 1: Files

Figura 2 prezintă funcționarea sistemului în simulare, însă aceasta oferă doar o demonstrație limitată. Dintr-o singură captură este greu de surprins complet comportamentul unui joc, deoarece secvențele, tranzițiile dintre stări și interacțiunea utilizatorului se observă corect doar pe parcursul mai multor pași. În plus, simularea verifică logica și sincronizarea semnalelor, dar nu poate reproduce experiența reală de utilizare de pe placă (viteza percepută la afișare, ritmul inputului și „feeling-ul" jocului).

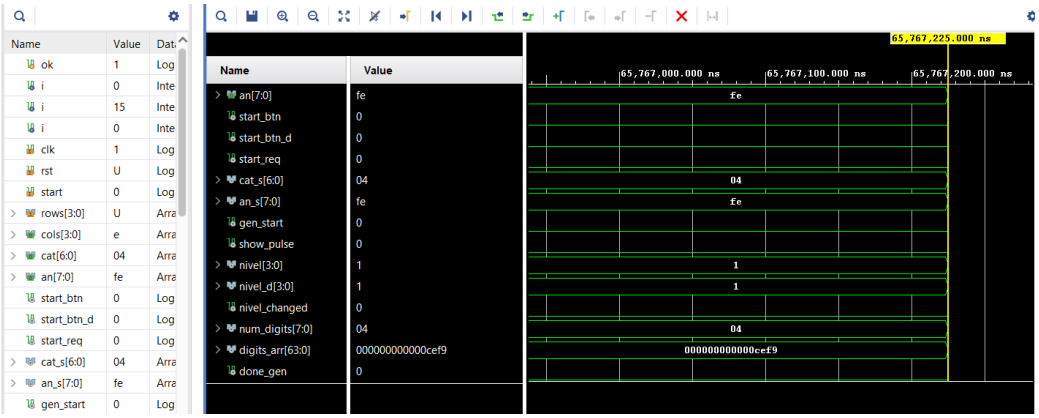

Figura 2: Simulator

Figura 3 prezintă raportul de utilizare a resurselor după implementare, atât pentru modulul de top (`top_uart`), cât și împărțit pe submodule. Se observă că design-ul folosește un număr redus de LUT-uri și registre raportat la resursele disponibile pe placă, ceea ce confirmă că proiectul încape fără probleme și lasă loc pentru extinderi. În plus, tabelul arată contribuția fiecărui modul (generator, KYPD, UART, SSD) la utilizarea totală.

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	DSPs (240)	Bonded IOB (210)	BUFGCTRL (32)
top_uart	403	337	6	1	158	402	1	1	27	1
generator (random_digits_gen)	147	80	6	1	61	146	1	0	0	0
inst_uart (uart_tx)	26	29	0	0	18	26	0	0	0	0
kypd_inst (kypd_controller)	133	81	0	0	43	133	0	0	0	0
ssd_driver_call (ssd_divider)	70	39	0	0	32	70	0	1	0	0

Figura 3: Hierarchy

5.1 Procedura de testare

Pentru a verifica faptul că jocul funcționează corect și stabil, am testat sistemul pe mai multe runde și pe mai multe niveluri, urmărind atât comportamentul pe placă (SSD + tastatură), cât și mesajele primite pe PC prin UART. Testarea a fost făcută prin scenarii simple, repetabile, astfel încât să acopere toate stările importante (SHOW/INPUT/CHECK/WIN/LOSE) și tranzițiile dintre ele.

- **Test WIN (nivel 1):** am pornit jocul, am memorat secvența afișată și am introdus exact aceleași cifre în aceeași ordine; sistemul a intrat în starea WIN, iar la apăsarea butonului de start nivelul a crescut și s-a trecut la o rundă nouă.
- **Test LOSE (nivel 1):** am pornit jocul și am introdus intenționat o cifră greșită; sistemul a intrat în starea LOSE, iar nivelul a rămas neschimbat; la apăsarea butonului de start s-a reluat runda la același nivel, cu o secvență nouă.
- **Test progres pe mai multe niveluri:** am repetat runde câștigate pe 2–3 niveluri consecutive și am verificat că dificultatea crește: secvența devine mai lungă, iar afișarea este mai rapidă față de nivelul anterior.
- **Test tastatură (apăsări rapide / ținut apăsat):** am testat apăsări rapide și situația în care o tastă este ținută apăsată mai mult timp, ca să verific că input-ul este numărat corect și nu apar dublări; datorită detectării pe front (`key_valid` / `key_valid_d`), o apăsare este înregistrată o singură dată.
- **Test UART + Python:** după fiecare rundă (WIN/LOSE) am verificat că pe PC apare mesajul corespunzător (nivel + rezultat) și că best score-ul este actualizat corect atunci când se atinge un nivel mai mare.

5.2 Dovezi de funcționare în utilizare reală

Pe lângă simulare, am verificat funcționarea direct pe placă și prin mesajele afișate pe PC. Captura de ecran, de mai jos arată feedback-ul din codul Python, adică exact ce vede utilizatorul în timpul jocului (mesaje pe PC prin UART).

```
Conectat la COM5. Resetati placa (BTN DOWN) si apasati START (Center).
--- RECORD ACTUAL (High Score): Nivelul 0 ---
--> Am primit 'L', astept nivelul...
[JOC] Nivel initial citit: 1
[WIN] Ai trecut de nivelul 1!
      >>> Incalzirea... Asta e de nota 5, trecem clasa. <<<
*** RECORD NOU! Noul High Score salvat: 1 ***
      -> Urmeaza nivelul 2...
[FAIL] Ai pierdut la nivelul 2.
Recordul tau ramane: 1
[WIN] Ai trecut de nivelul 2!
      >>> Bun! Dar la examen subiectele sunt mai grele. <<<
*** RECORD NOU! Noul High Score salvat: 2 ***
      -> Urmeaza nivelul 3...
```

Figura 4: UART output

5.3 Comparație scurtă (de ce am ales varianta asta)

Fără UART, feedback-ul ar fi fost limitat la ce se poate afișa pe SSD (cifre și câteva simboluri), ceea ce face mai dificilă urmărirea nivelului și depanarea. Prin UART + Python am putut afișa clar pe PC nivelul și rezultatul fiecărei runde și am putut ține și un best score global, ceea ce a făcut testarea mai simplă și experiența de utilizare mai completă.

De asemenea, la citirea tastaturii, dacă aș fi folosit direct semnalul `key_valid` fără detectare pe front, aceeași apăsare ar fi putut fi numărată de mai multe ori. Folosind `key_valid` împreună cu `key_valid_d`, o apăsare este tratată ca un eveniment unic, iar input-ul este înregistrat corect.

În timpul testării au apărut și câteva probleme de sincronizare, mai ales la trecerea dintre etapele jocului și la citirea tastelor. Acestea au fost rezolvate prin separarea pe stări și prin folosirea unor semnale între module și prin detectarea unei apăsări noi de start. În final, testele au confirmat că jocul funcționează corect pe placă, că trecerea între stări se face cum trebuie și că informațiile trimise către PC sunt coerente cu ce se întâmplă în joc.

6 Concluzii

În acest proiect am realizat un joc de memorie implementat pe placa Nexys4 Artix-7, unde sistemul generează o secvență de cifre, o afișează pe display-ul cu 7 segmente, iar utilizatorul trebuie să o introducă de la tastatura Pmod KYPD în aceeași ordine. Obiectivele principale au fost atinse: generarea pseudo-aleatoare a secvenței (LFSR), afișarea corectă pe SSD cu viteză dependentă de nivel, citirea tastaturii prin scanare și verificarea secvenței, plus feedback suplimentar prin UART către PC.

Contribuția mea principală a fost integrarea tuturor acestor părți într-un sistem complet controlat de o mașină de stări (FSM), cu tranziții clare între etape (generare, afișare, input, verificare și rezultat). Am implementat și mecanisme care au făcut jocul mai stabil și mai ușor de controlat, cum ar fi o stare intermediară pentru stabilizarea semnalelor (`PRE_GEN`), un handshake pentru afișaj (`TRIGGER_SSD`) și detectarea unei apăsări noi pe tastatură (edge detect pe `key_valid`). Partea de UART + Python a fost utilă pentru mesaje, debug și pentru a ține un best score, lucru care nu ar fi fost practic doar cu SSD-ul plăcii.

Ca avantaje, soluția este interactivă, rulează direct în hardware și răspunde rapid la input, iar structura pe module (generator, control SSD, control KYPD, UART și FSM) face proiectul ușor de urmărit și extins. Un dezavantaj este că afișajul cu 7 segmente limitează mult ce poți comunica utilizatorului (de aceea UART a fost util), iar partea de timing/optimizare pe FPGA poate deveni o problemă dacă se cere o frecvență de ceas foarte mare sau dacă se adaugă logică suplimentară.

Proiectul poate fi folosit ca exemplu de sistem embedded pe FPGA, util pentru învățarea proiectării cu FSM, integrarea perifericelor (SSD, tastatură matricială) și comunicație serială UART. De asemenea, ideea se poate adapta și pentru alte jocuri simple sau pentru interfețe care cer input rapid și control determinist.

Ca dezvoltări viitoare, proiectul poate fi îmbunătățit prin adăugarea unui mod de dificultate (mai multe reguli de nivel), suport pentru reset/pauză, un sistem mai clar de scor pe SSD (de exemplu afișarea nivelului curent și a scorului maxim), și un mod de joc cu mai multe vieți. Pe partea hardware, se poate lucra și la optimizarea drumurilor critice (de exemplu verificarea secvenței pe mai multe cicluri) pentru a respecta mai ușor constrângerile de timing la frecvențe mai mari. În plus, aplicația de PC ar putea fi extinsă cu o interfață grafică simplă, statistici și salvarea scorurilor între rulări.

Bibliografie

[1] Wikipedia, *Field-programmable gate array*, https://en.wikipedia.org/wiki/Field-programmable_gate_array.

[2] AMD, *Introduction to FPGA Design with Vivado High-Level Synthesis*, https://docs.amd.com/api/khub/documents/dFYuRTF_b2KbHxOTQlrRsw/content.

[3] Digilent, *Nexys A7 (Artix-7) Reference Manual*, documentație online, https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual.

[4] Digilent, *Pmod KYPD Reference Manual*, documentație online, https://digilent.com/reference/pmod/pmodkypd/reference-manual.

[5] Wikipedia, *Universal asynchronous receiver-transmitter (UART)*, https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter.

[6] Wikipedia, *Finite-state machine* https://en.wikipedia.org/wiki/Finite-state_machine

[7] Wikipedia, *Seven-segment display* https://en.wikipedia.org/wiki/Seven-segment_display

[8] Wikipedia, *Linear-feedback shift register (LFSR)*, https://en.wikipedia.org/wiki/Linear-feedback_shift_register.

[9] Wikipedia, *VHDL*, https://en.wikipedia.org/wiki/VHDL.

[10] pySerial Documentation, *pySerial — Serial Port Access Library for Python*, https://pyserial.readthedocs.io/.

Anexe

Anexa A
Funcția pseudo-aleatoare

```
function lfsr_next(s : unsigned(15 downto 0)) return unsigned
    is
  variable temp: unsigned(15 downto 0);
begin
  -- x^16 + x^14 + x^13 + x^11 + 1
  temp := s(14 downto 0) & (s(15) xor s(13) xor s(12) xor s
   (10));
  return temp;
end function;
```

Listing 1: Funcția pseudo-aleatoare

Anexa B
Detectare tastă apăsată pe KYPD

```
-- Determina daca este vreo tasta apasata
process(clk, rst)
begin
  if rst = '1' then
    row_pressed <= '0';
    detected_row <= 0;
    detected_col <= 0;
  elsif rising_edge(clk) then
    if scan_tick = '1' then
      row_pressed <= '0';
      for i in 0 to 3 loop
        if rows_intern(i) = '0' then
          row_pressed <= '1';
          detected_row <= i;
          detected_col <= col_selectata;
        end if;
      end loop;
    end if;
  end if;
end process;
```

Listing 2: Detectare apăsare tastă pe KYPD

Anexa C
Verificarea celor două array-uri

```
when CHECK =>
  fail_mode <= '0';
  ok := '1';

  -- Verificam secventa
  for i in 0 to MAX_DIGITS-1 loop
    if i < to_integer(unsigned(num_digits)) then
      if seq_user(i) /= seq_gen(i) then
        ok := '0';
      end if;
    end if;
  end loop;

  if ok = '1' then
    st <= WIN;
  else
    st <= LOSE;
  end if;
```

Listing 3: Decizie de WIN sau LOSE pe baza celor doua arrayuri