Logo, company name

Description automatically generated

**Facultatea de Automatica și Calculatoare**

**Implementarea prin hardware**

**a jocului 2048**

Îndrumător Studenți: Apostol Matei-Gabriel

Grupa: 30215

29.05.2022

**Cuprins**

1.Rezumat

2.Introducere

3.Fundamentare teoretică

4.Proiectare și implementare

5.Rezultate experimentale

6.Concluzii

**1.Rezumat**

Jocul propus spre implementare este 2048, un joc în care sunt manipulate 16 valori. Aceste valori încep de la 2 și continuă să crească din cauza mutărilor până la 2048, de unde provine și numele. Jucătorul are opțiunea de a alege dintre 4 mutări: să mute în sus, în stânga, în dreapta și în jos. Mutatul semnifică faptul că numerele se mută, ilustrativ, spre o direcție precizată de jucător. Dacă 2 numere sunt identice atunci acestea se adună. Scopul jocului este de a cupla atât de multe numere încât sa ajungă un număr la valoarea 2048. De fiecare dată când jucătorul mută, apare un nou 2 pe o poziție liberă. Dacă nu mai există poziții pentru următorul 2, atunci jocul este pierdut. Scopul acestui proiect este de a implementa acest joc într-un mediu hardware, folosind limbajul de descriere hardware VHDL. A fost folosit modulul de simulare pentru a demonstra funcționalitatea implementării. Fiind în majoritate un domeniu concurent, mutările au fost greu de implementat. În mediul software, aceste funcționalități ar putea fi implementate cu ușurință folosind librării existente și proprietatea secvențială a executării liniilor de cod.

**2.Introducere**

În cadrul acestui proiect, este folosit un limbaj pentru descriere hardware, numit VHDL. Acest nume provine de la **V**HSIC (Very High Speed Integrated Circuits) **H**ardware **D**escription **L**anguage. Între anii 1980, IEEE a sponsorizat dezvoltarea acestui limbaj cu scopul de a crește nivelul circuitelor de foarte mare viteză. A devenit limbajul standard al industriei pentru a descrie sisteme digitale. Celălalt limbaj de descriere hardware este Verilog. Amândouă sunt niște limbaje cu o putere considerabilă pentru a descrie și simula siteme digitale complexe.

Cel mai înalt nivel de abstractizare este modelul comportamental pentru a descrie sisteme, deoarece descrie cam ce funcționalitate ar avea componenta respectivă. Proiectul acesta se bazează pe această caracteristică din cauza procesului foarte dificil de a construi o arhitectură logică cât se poate de amanunțită. Descrierea comportamentală specifică relația dintre semnalele de intrare și de ieșire.[1]

Pe de altă parte, o descriere structurală este o mulțime de porți și componente care sunt interconectate să rezolve o anumită problemă. Aceasta poate fi reprezentată cu ușurință de porți logice și componente fizice. Este cea mai aproape de reprezentarea fizică a sistemului.

Jocul 2048 este un joc care are o caracteristică mai mult vizuală din cauză ca acesta este reprezentat prin mișcările unor cuburi într-un chenar. Această ilustrație este a unui exemplu de joc în desfășurare. Acesta se poate găsi pe site-ul <https://2048game.com/>

Jocul este reprezentat într-un chenar 4X4. Se poate observa în imagine o mutare la stânga.

Sunt 4 cuburi cu valoarea 4, fiind situate pe același rând 2 câte 2. Atunci când această mutare se produce, pătratele cu număr se deplasează în această direcție până atunci când întâmpină alt pătrat plin. Acestea se cuplează dacă au aceeași valoare. Dacă nu, pătratul se oprește. Se poate observa din imagini că, pe primul rând se află un pătrat ce conține numărul 4 în cea mai din dreapta poziție. Acesta se mișcă cu o poziție la stânga din cauză ca pătratul din stânga lui este gol, și se cuplează cu pătratul cu același număr.

Graphical user interface, application, calendar

Description automatically generatedA picture containing text, electronics

Description automatically generated

Se poate observa că și pe al doilea rând se face o cuplare deoarece pătratul din stânga lui conține același număr. În a doua imagine a apărut un pătrat cu numărul 2 deoarece s-a efectuat o mutare.

Scopul proiectului nu este de a reprezenta într-o interfață grafică cum aceste chenare se deplasează dintr-o locație în alta, ci de a folosi limbajul VHDL pentru a descrie funcționalitatea acestui joc. Eu am abordat această problemă într-un mod secvențial suportat de limbajul VHDL.

Mi-am propus să rezolv problema mutărilor conceptual, nefiind un proces grafic, ci la nivel de calcule. Implementarea aleasă se folosește de manipularea și construirea vectorilor pentru a putea simula cum ar funcționa jocul.

**3.Fundamentare teoretică**

În cadrul acestui proiect s-a folosit un automat de stări finite pentru a executa în mod secvențial anumite instrucțiuni. Acesta este un model matematic pentru orice sistem care folosește un număr limitat de stări condiționale. Un exemplu practic de automat de stări finite ar fi un set de butoane pe un controller care sunt conectate la niște acțiuni într-un joc. Când utilizatorul apasă pe niște butoane, sistemul știe cum să reacționeze la acești stimuli.[2]

Construirea unui automat de stări finite este conceput prin următoarele considerente:

* Un set de evenimente întâmplate .
* Un set de evenimte ce vor fi parcurse care corespund evenimentelor precedente.
* Un set de stări la care se așteaptă sistemul.

În cadrul unui FSM (finite state machine), toate stările luate în considerare există într-o listă finită, iar sistemul se poate afla într-o singură stare la un moment dat. Acest mod de implementare poate fi un mod prin care se poate studia și testa fiecare intrare și ieșire la un anumit moment dorit.

Atunci când sistemul își schimbă starea se numește o tranziție de stare. Testarea calității unui sistem include verificarea fiecărei stări și tranzițiilor prin a considera toate intrările posibile. Pentru probleme mai simple, aceste informații se pot afișa pe tabele, matrici, ilustrații, dar automatele finite de stări permit inginerilor să modeleze scenarii foarte complexe.

Pentru a putea genera după mutări un 2 pe o poziție aleatoare s-a implementat un generator pseudo-aleator. Un dispozitiv de generare de numere aleatorii, poate genera unul sau mai multe numere aleatorii în intervalul definit. Ambele există generatoare de numere bazate pe hardware și pseudoaleatoare. Generatoarele de numere aleatoare bazate pe hardware pot include utilizarea de zaruri, monede pentru răsturnare și multe alte dispositive.[3]

Un generator de numere pseudoaleatoare este un algoritm folosit pentru a genera secvențe aleatoare. Aproximează proprietățile secvențelor de numere aleatoare. Generatoarele aleatoare bazate pe computer sunt aproape întotdeauna generatoare de numere pseudoaleatoare. Generarea numerelor pseudoaleatoare nu produce numere aleatoare. De asemenea, generatoarele menționate mai sus sunt generatoare de generație pseudoaleatoare. Deși pot genera suficiente numere aleatorii pentru majoritatea aplicațiilor, ele nu ar trebui să fie utilizate în scopuri criptografice. Numărul adevărat aleatoriu se bazează pe fenomene fizice precum zgomotul termic, zgomotul atmosferic sau fenomenele cuantice. Metodele care produc numere aleatoare adevărate includ compensarea oricăror distorsiuni cauzate în timpul măsurării.

În principiu, proiectul ales ar trebui încărcat și testat pe o placă FPGA. Acesta este un circuit hardware care poate fi programat de un utilizator pentru a produce una sau mai multe operații logice. Dacă vrem să folosim un cip standard, ca un procesor CPU din laptop, nu se poate programa, el având deja task-uri existente. Cu aceste cipuri, un utilizator poate să scrie cod pentru a-l încărca mai apoi pe un astfel de cip și să execute funcții. Acest cod poate fi înlocuit sau șters ulterior, dar cipul rămâne neschimbat.[4]

Când se lucrează cu o placă FPGA, nu există un cip. Utilizatorul programează circuitele. Programarea acestei plăci poate consta în implementarea unei porți logice, sau să include funcții complexe.

Folosirea acestor plăci poate acoperi o gamă foarte largă de funcționalități, de la echipament pentru video și imagini până la circuite pentru calculatoare, mașini, aeronave și aplicații militare. Intel folosește FPGA-uri pentru a construi noi cipuri.

Microsoft a folosit cipuri FPGA pentru a accelera inferența. Aplicarea FPGA la accelerarea AI a motivat Intel să dobândească Altera cu scopul de a integra FPGA în procesoarele serverului, care ar putea să accelereze AI, precum și sarcini de scop general.

În iunie 2017, cercetătorii IBM au anunțat o arhitectură, în contrast cu arhitectura Von Neumann, bazată pe calcularea în memorie și matricea de schimbare a fazelor aplicată detectării corelațiilor temporale, care intenționează să generalizeze abordarea calculelor eterogene și sistemelor masive paralele.În octombrie 2018, cercetătorii IBM au anunțat o arhitectură bazată pe procesarea în memorie și modelată în rețeaua sinaptică a creierului uman pentru a accelera rețelele neuronale profunde. Sistemul se bazează pe tablouri de memorie cu modificări de fază.[5]

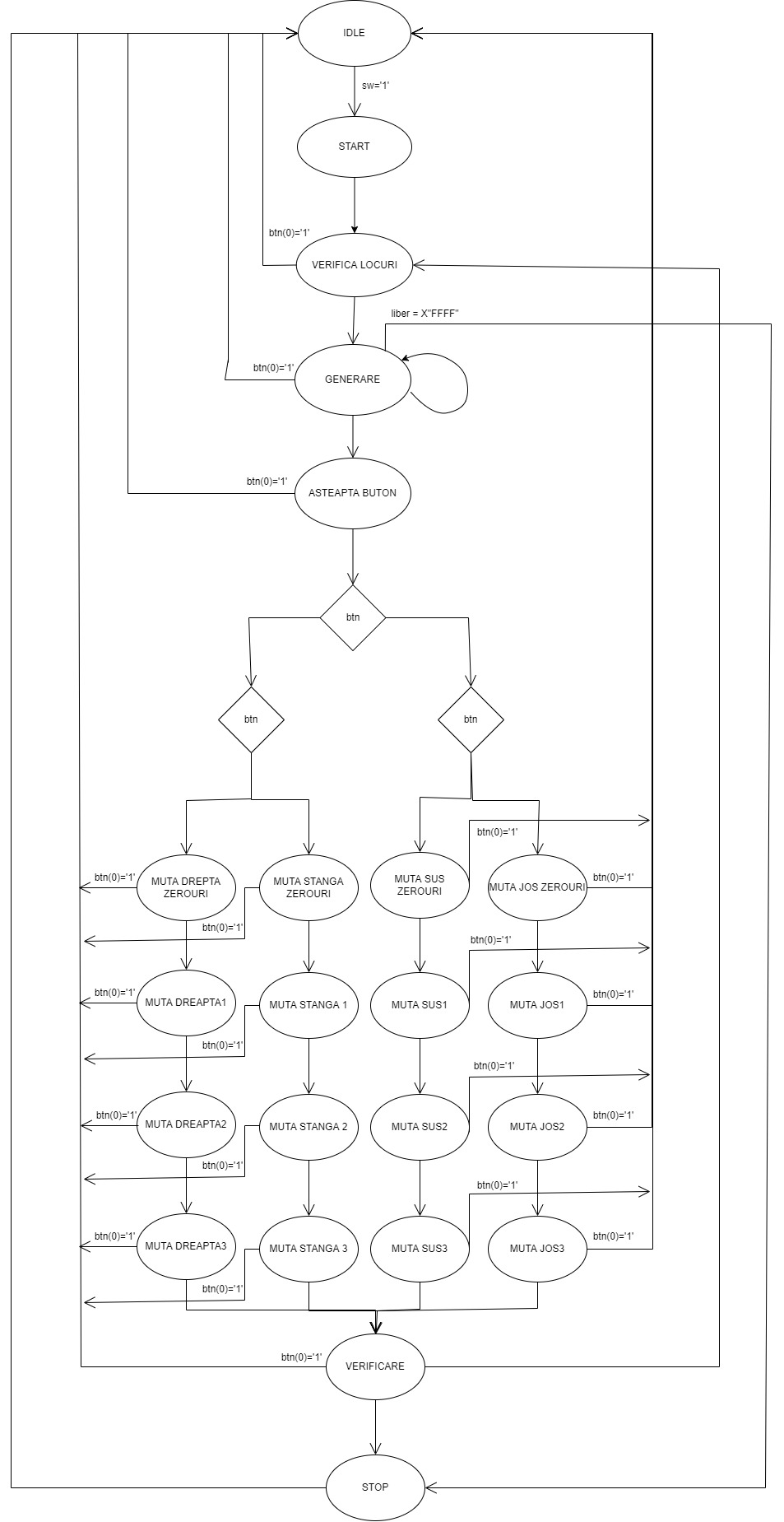
**4.Proiectare și implementare**

Pentru implementarea jocului s-a folosit un automat finit de stări ce se poate observa în imaginea de mai jos. Am luat în considerare chenarul pentru acest joc ca fiind un registru cu 16 vectori de 12 biți. Pe pozițiile în care se află o valoare se vor memora în registru, iar unde spațiul este gol, în memorie vor fi valori de 0.

Prima stare, idle, este folosită pentru a inițializa memoria, deci vom pune în memorie 16 valori de 0. Pentru a începe jocul, se va folosi un switch. Dacă valoarea acestuia este 1, acesta intră în următoarea stare, fiind de start.

Fără alte condiții se va trece într-o stare numită verifică locuri. În această stare este construit un vector de 16 biți. Pentru a putea verifica dacă mai există locuri în memorie, am folosit 2 loop-uri. Pentru a reprezenta faptul că în memorie este reprezentată o matrice, am folosit 2 iteratori, unul reprezentând rândul, iar celălalt coloanal la care mă refer. Ca să mă refer la un element în memorie, mă folosesc de formula 4 \* rând + coloană. Vectorul de biți va conține 0 într-o anumită poziție dacă elementul de la acea poziție din memorie este 0, 1 dacă acea poziție este ocupată, adica există un element diferit de 0.

În starea numită generare se va intra necondiționat. Această stare reprezintă alegerea unei poziții pentru un 2 cu un generator pseudo-aleator. Generatorul pseudo-aleator constă în 3 numărătoare, din care 2 crescătoare, unul numără cu 1, iar al doilea din 3 în 3. Celălalt numărător este descrescător și scade cu 1. Am folosit și un divizor de frecvență deoarece se schimba numărul generat la 10 ns de 2 ori și se generau 2 poziții. Acest generator funcționează concurent, iar ieșirea acestui generator va fi folosită dacă sunt disponibile locuri în matrice. Dacă sunt și elementul de pe poziția generată este 0, atunci se va asigna valoarea 2. Dacă pe poziția generată este deja un număr diferit de 0, atunci se va aștepta generarea altui număr din generator. Dacă nu mai există poziții libere, atunci se va trece în ultima stare.



Pentru a aștepta intrarea specificată de utilizator, am folosit o stare diferită în care se trece după ce s-a terminat de generat pe o poziție un 2. Se așteaptă apăsarea unui buton. Acesta este procesul în care un buton va lua valoarea 1. În funție de ce buton se alege, va avea loc o mutare. Dacă butonul 1 va fi activat atunci mutarea va fi în sus, dacă butonul 2 este activat atunci se mută în partea stângă. Se mută în partea dreaptă dacă butonul 3 este activat, iar în jos dacă butonul 4 are valoarea 1.

Procesul de mutare este alcătuit din 4 algoritmi prin care am simulat procesul descris mai sus. Prima stare se numește starea zerouri. Acesta este procesul prin care atunci când există zerouri între elemente, iar acestea se mișcă în direcția lor, aceste numere le vor lua locul. Algoritmul este construit într-un mod secvențial. De exemplu, dacă se efectuează mutarea la stânga, dacă pe poziția cea mai din stânga de pe un anumit rând se află un zero, iar lângă el se află un număr nenul, atunci tot rândul se „shiftează” la stânga. Acest proces de „shiftare” semnifică faptul că toate elementele de pe pozițiile de lângă, în cazul acesta fiind din dreapta, se vor muta cu o poziție la stânga, iar pe cea mai din dreapta poziție se pune valoarea 0. Asta se întamplă și dacă avansăm spre drepta, adică dacă găsim un 0, iar lângă el se află un număr nenul, atunci facem shiftarea. Din cauză că se folosește un proces, se reevaluează și se verifică din nou aceste lucruri.

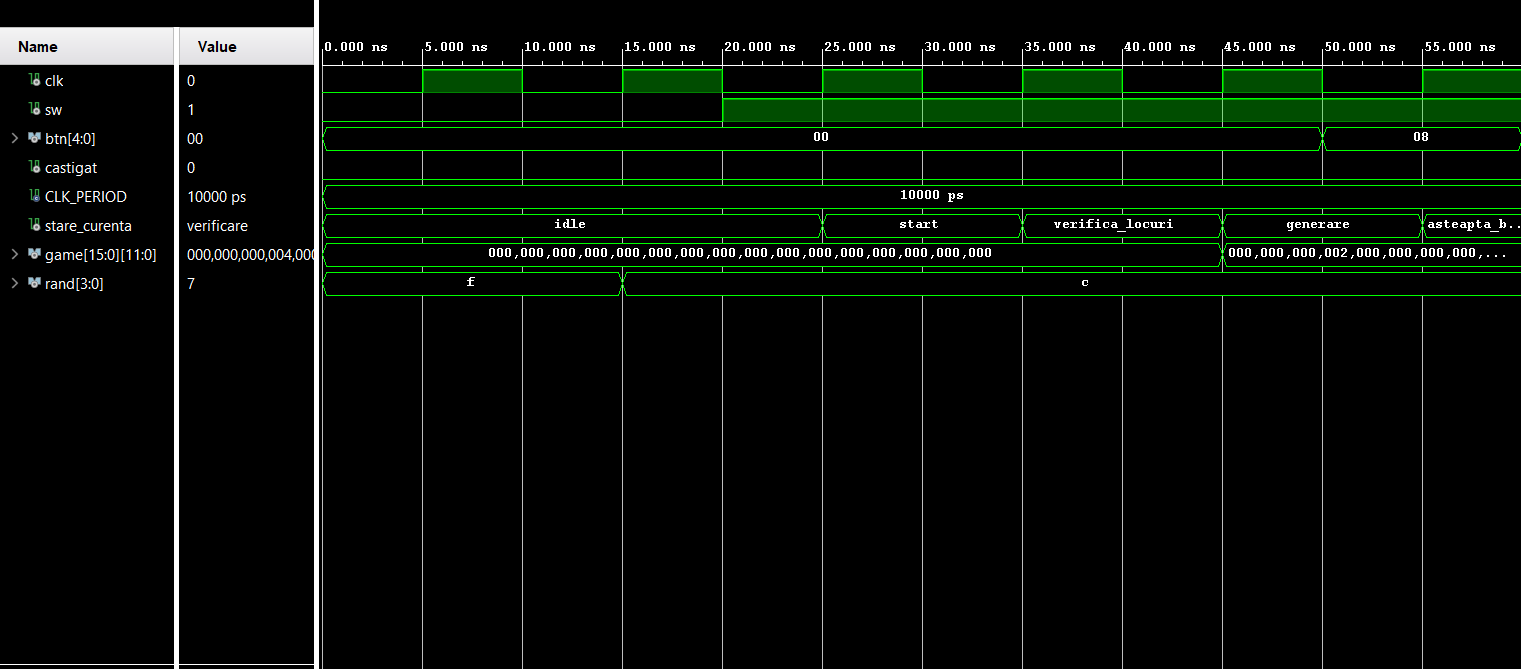
A doua stare prin care se va trece va fi una de cuplare a două valori dacă acestea sunt pe poziții consecutive. În cazul unei mutări la stânga, la un anumit rând, se verifică primele 2 poziții. Dacă acestea sunt egale, atunci se cuplează și restului rândului se „shiftează”. Următoarele 2 stări reprezintă același lucru, numai că se vor evalua pentru a doua și a treia poziție, respectiv a treia și a patra. Am folosit 3 stări pentru această funcționalitate deoarece aceste stări simultane vor provoca rezultate eronate. Din cauză că procesul se va evalua în continuu procesul, se va identifica într-un anumit caz un rezultat diferit de cel așteptat. Se va specifica la vizualizarea experimentelor. Pentru fiecare mutare se folosesc indexii potriviți. Adică se iau în considerare coloanele pentru mutarea la stânga și la dreapta, iar coloanele sunt precizate cu ajutorul unui loop, operațiile repetându-se pe fiecare rând. Asemănător se întâmplă în cazul mutării în sus și în jos. Se fac operațiile pe rânduri, iar colonele se specifică într-un loop.

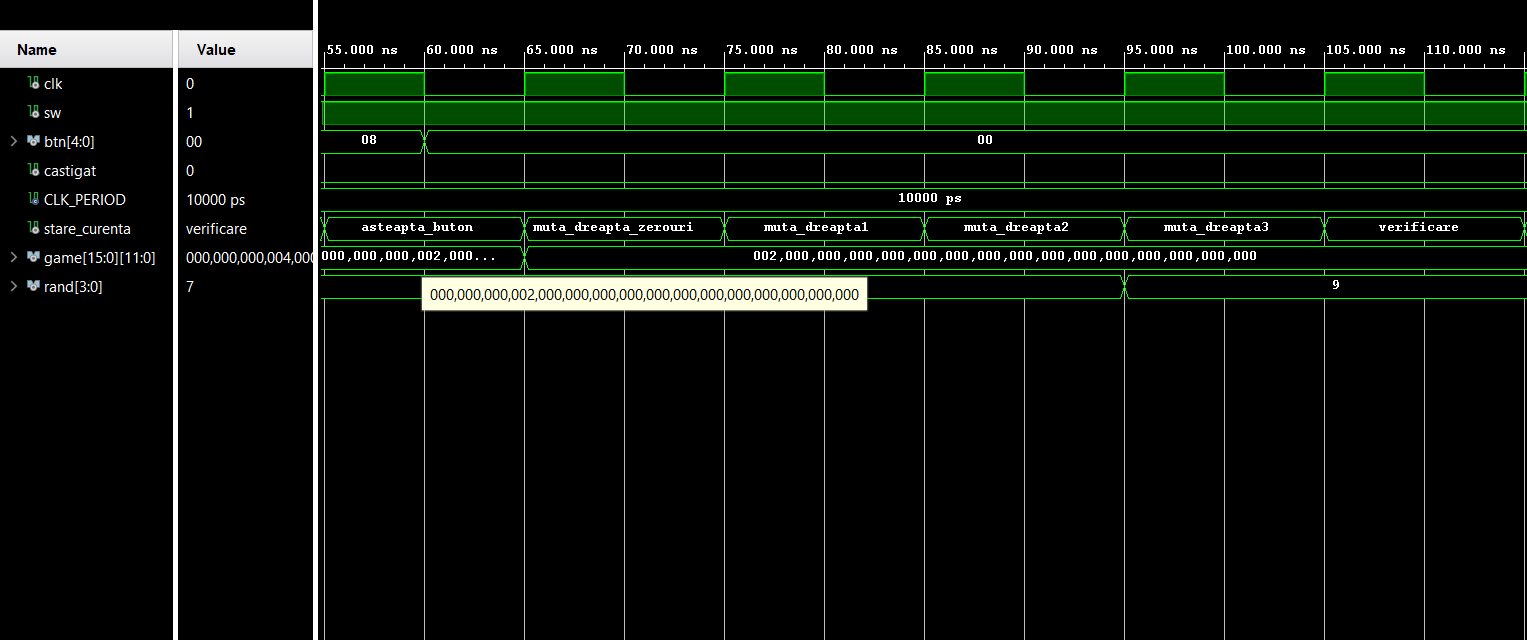
După ce o mutare se efectuează, va ajunge în starea de verificare, în care se verifică dacă există vreun element în memorie cu valoarea 2048. Dacă da, se trece în starea de stop, jocul fiind câștigat. Dacă nu se găsește, atunci se va trece la starea de verificare a locurilor.

Butonul 0 este folosit pentru a reseta jocul asincron. Se poate reseta jocul din orice stare.

**5.Rezultate experimentale**

Pentru început se va lua în calcul începutul jocului. Se inițializează memoria cu 0 și se efectuează o mutare la dreapta.





Din această simulare se poate observa cum intră sistemul în stările pe care le-am implementat, prin intermediul semnalului stare\_curenta. Jocul în sine, memoria, este semnalul game. Din prima imagine se poate observa că se pornește de la o matrice nulă. Atunci când ajunge la starea de generare, pe poziția 12 se pune un 2. Mai apoi se efectuează mutarea la dreapta. Precizez faptul că matricea este afișată sub forma unui vector, de jos în sus, de la dreapta la stânga. Deci dacă în simulator apare în acest format:   
 

Trebuie tratată ca și următoarea matrice :

002 000 000 000

000 000 000 000

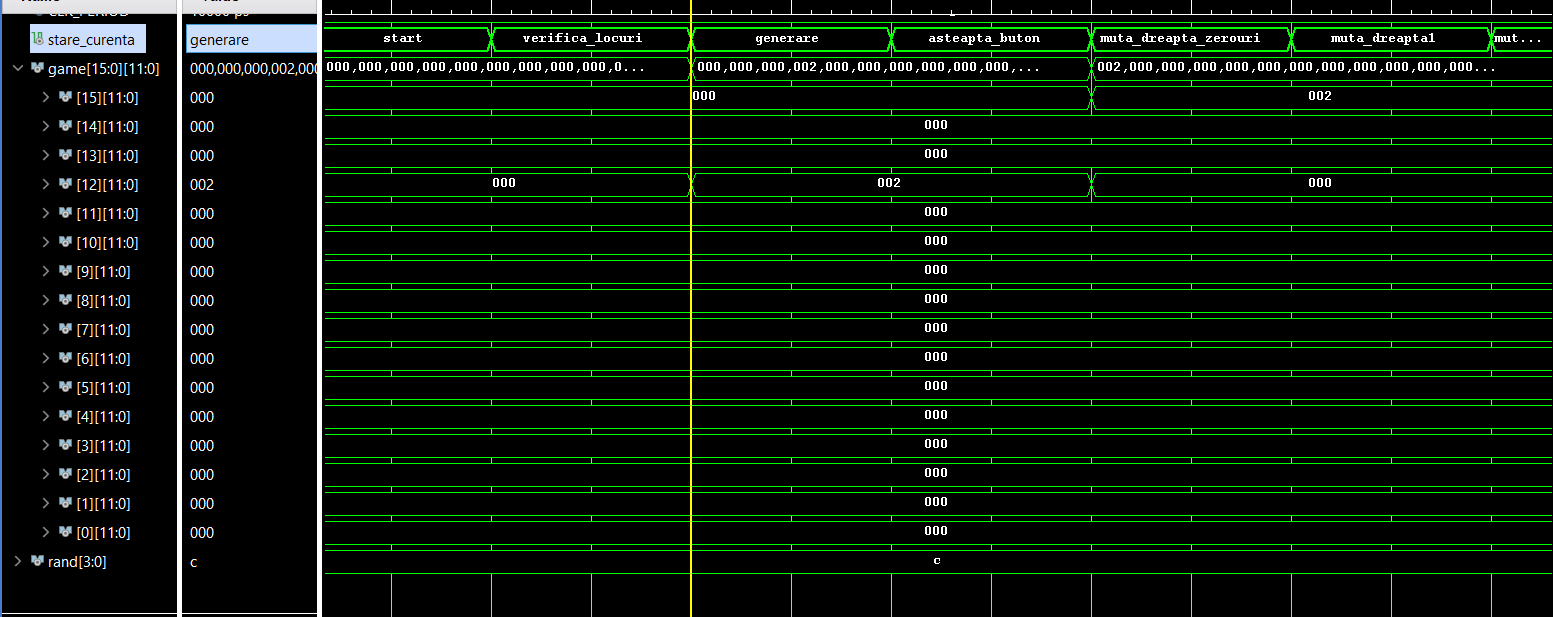
000 000 000 004

000 000 000 004

Deci, în această simulare, dacă matricea inițială arată astfel:



După generare va arăta astfel:



Adică pe poziția 12 se pune valoarea 2. Se poate observa și mutarea elementului 2 la dreapta, însemnând că va ajunge din cea mai din stânga poziție la cea mai din dreapta poziție (se asignează 2 pe poziția 15, iar pe poziția 12 se pune 0).

O imagine care conține shoji, cuvinte încrucișate, public, mozaicat

Descriere generată automatO imagine care conține shoji, cuvinte încrucișate, interior, public

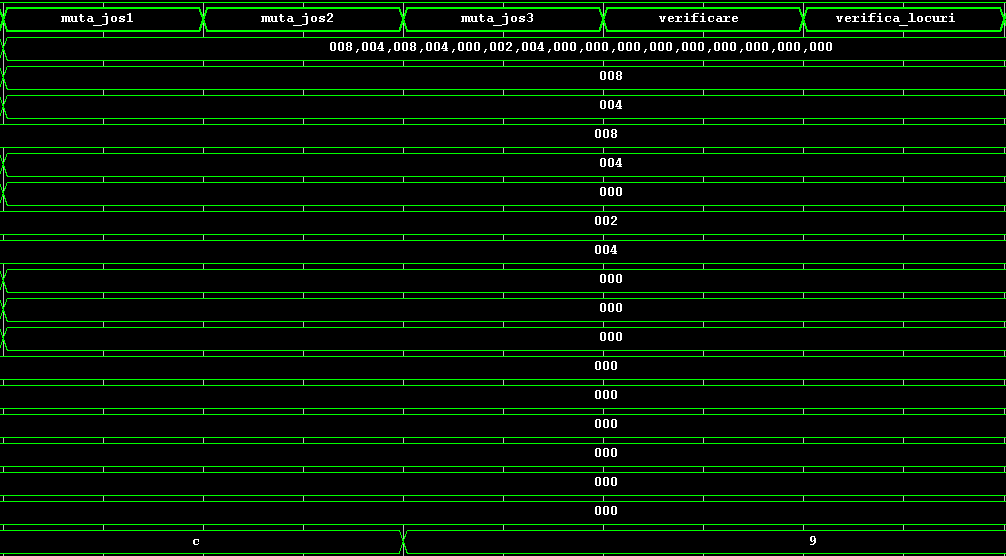
Descriere generată automat

Imaginile de mai sus reprezintă cum s-ar schimba de fapt în joc după mutarea la dreapta.

Voi prezenta alt scenariu la un nivel mai avansat după mai multe mutări în joc:

O imagine care conține text, ecran

Descriere generată automat



Matricea înainte de generare este aceasta:

O imagine care conține text, lumină

Descriere generată automat

Matricea după generare:

O imagine care conține text, lumină, dulap

Descriere generată automat

Matricea după trecerea prin starea mutare\_jos\_zerouri:

O imagine care conține text, lumină, dulap

Descriere generată automat

Matricea după stările de muta\_jos1,2,3:

O imagine care conține cuvinte încrucișate, lumină

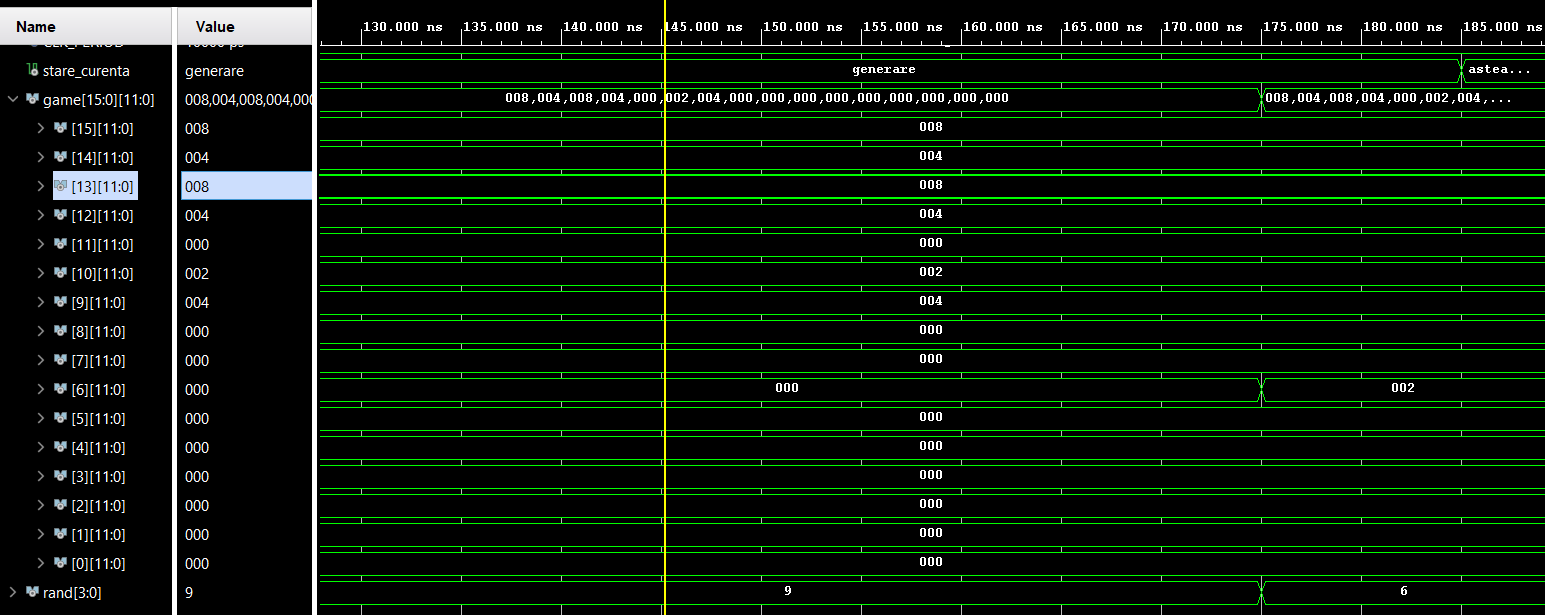
Descriere generată automat

Se poate observa că în loc să se cupleze pe ultima coloană numai rândul 3 cu rândul 4, s-au cuplat de 2 ori. De aici vine și rezultatul 8, din cauză că repetă același pas până ajunge în punctul în care nu mai există elementele egale deasupra lui. Rezultatul așteptat:

O imagine care conține shoji, cuvinte încrucișate, lumină

Descriere generată automat

În această simulare se poate observa procesul de așteptare al stării de generare până primește o poziție valabilă pentru a introduce un 2:



Atunci când rand devine 6, se pune 2 pe poziția 6 din memorie, aceasta fiind nulă înainte. S-a încercat cu poziția 9, dar aceasta este invalidă.

**6.Concluzii**

Pentru a putea încărca acest proiect trebuie folosite generatoare de mono-impuls pentru butoane și un fișier de constrângeri, astfel încât ieșirile din fișierul de constrângeri să fie conectate ieșirile din entitatea fișierului VHDL, asemenea intrărilor. Algoritmii implementați sunt parțial corecți din motivul specificat mai sus, de aceea acest proiect poate fi dezvoltat prin schimbarea acestor algoritmi. Acest joc este reprezentat în memorie și vizualizat în simulator, dar acesta poate fi testat pe o placă FPGA și poate fi proiectat astfel încât aceste mișcări să se întâmple vizual, ca și jocurile de acest gen deja implementate. Cel mai apropiat mod de a reprezenta aceste mișcări sunt afișarea lor cu ajutorul unui port VGA pe un ecran.

**Bibliografie**

[1] University of Pennsylvania Department of Electrical and Systems Engineering [“VHDL Tutorial”](https://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html#_Toc526061341)

[2] [DEFINITION:finite state machine](https://www.techtarget.com/whatis/definition/finite-state-machine)

[3] [Generator De Numere Aleatorii](https://purecalculators.com/ro/random-number-generator)

[4] [What Is an FPGA and Why Is It a Big Deal?](https://www.prowesscorp.com/what-is-fpga/)

[5] [FPGA Based Deep Learning Accelerators Take on ASICs (nextplatform.com)](https://www.nextplatform.com/2016/08/23/fpga-based-deep-learning-accelerators-take-asics/)