Universitatea Națională de Știință și Tehnologie POLITEHNICA București

Școala Doctorală de Electronică, Telecomunicații și Tehnologia Informației

Decizie nr. 234 din 02-10-2024

Teză de doctorat
Rezumat

Diana DRANGA

Contribuții la verificarea funcționala utilizând Inteligență Artificială

COMISIA DE DOCTORAT

Prof. Dr. Ing. Mihai CIUC Universitatea Națională de Știință și Tehnologie POLITEHNICA București	Președinte
Conf. dr. ing. Habil. Cătălin DUMITRESCU Universitatea Națională de Știință și Tehnologie POLITEHNICA București	Conducător de doctorat
Prof. Dr. Ing. Cătălin CĂLEANU Universitatea Politehnică din Timișoara	Referent
Prof. Dr. Ing. Ștefan TOMA Academia Tehnica Militară	Referent
Conf.Dr.ing. Eduard POPOVICI Universitatea Națională de Știință și Tehnologie POLITEHNICA București	Referent

BUCUREŞTI 2024

Conținut

Capitolul 1. Introducere in domeniul verificării funcționale și scopul tezei

1.1 Prezentarea domeniului

Verificarea funcțională este un proces critic în cercetarea și dezvoltarea unui System-on-Chip (SoC). Scopul principal constă în a asigura că chip-ul se comportă corespunzător, conform descrierilor documentației.

În multe cazuri, mediul de verificare este compus din diferite instanțe, scrise în System Verilog folosind Universal Verification Methodology (UVM). Acesta este format din instanțe precum:

- uvm_driver, o clasă care primește tranzacțiile și le modelează conform protocolului și le transmite la interfețele Design Under Test (DUT). Acesta convertește stimulul la nivel de tranzacție în stimul la nivel de pin [1].
- uvm_monitor, instanță care preia ieșirile de la DUT și extrage informațiile din tranzacții, trimițându-le și către alte componente folosind un port de analiză Transaction Level Modelling (TLM) [1].
- uvm_sequencer, acționează mai mult ca un arbitru utilizat în controlul fluxului de tranzacții din secvențe multiple [1].
- uvm_agent, o încapsulare a claselor prezentate mai sus [1].
- uvm_scoreboard, instanță care are funcția principală de a verifica comportamentul DUT. În mod obișnuit, primește tranzacții de la intrările și ieșirile DUT prin porturile de analiză ale agentului UVM, rulează aceste intrări împotriva unui model de referință și compară ieșirea așteptată cu ieșirea reală [1].

Pentru a utiliza corect aceste clase, există o bibliotecă de clase UVM care oferă tot ajutorul pentru a dezvolta într-un mod rapid medii de testare robuste și componente de verificare reutilizabile. În bibliotecă, există și funcții utilitare și macro-uri.

Biblioteca de clase UVM oferă chiar mai multe utilități pentru a simplifica și sprijini dezvoltarea mediilor de verificare [1].

1.2 Referințe

[1] Accelera, UVM Guide

1.3 Scopul tezei de doctorat

Inteligența artificială poate juca un rol esențial în verificarea funcțională a SoC-urilor. Verificarea funcțională este un proces consumator de timp în care sunt testate multiple componente ale chip-ului. Datorită naturii consumatoare de timp a procesului, este esențială implementarea diferitelor strategii de reducere a timpului folosind Inteligența Artificială.

Folosind diverse tehnici, cum ar fi Large Language Models (LLM), Convolutional Neural Networks (CNN) și alte tehnologii, etapa de verificare va fi îmbunătățită și accelerată. Această accelerare va oferi un mecanism timpuriu de identificare a erorilor. Acest lucru va aduce beneficii semnificative industriei, deoarece dezvoltarea SoC-urilor este îndelungată și laborioasă.

Utilizarea ChatGPT și CNN reprezintă puncte de plecare excelente pentru cercetarea în reducerea timpului alocat verificării funcționale.

1.4 Conținutul tezei de doctorat

În primul capitol al tezei există un rezumat al domeniului verificării, ce este folosit de obicei în industrie, construcții specifice de clase, metode și sarcini.

În al doilea capitol, domeniul Inteligenței Artificiale va fi introdus în teză, explicând conceptele generale pentru a le înțelege pe deplin.

În al treilea capitol, vor fi introduse lucrări suplimentare folosind Inteligența Artificială Generativă.

În al patrulea capitol, este prezentat un CNN, dezvoltat de inginer, cu scopul de a minimiza timpul petrecut pentru analiza documentațiilor extinse, care sunt de obicei întâlnite în industrie.

Capitolul al cincilea reprezintă o concluzie asupra lucrărilor prezentate anterior.

Capitolul 2. Artificial intelligence in general

Inteligența Artificială (AI) este un domeniu în plină dezvoltare, care stârnește din ce în ce mai mult interesul cercetătorilor din mai multe domenii [1].

Un exemplu principal de model de învățare profundă (Deep Learning) este rețeaua feedforward, cunoscută în mod obișnuit ca Multi-Layer Perceptron (MLP). Un Multi-Layer Perceptron este, în esență, o funcție matematică ce corelează un set de valori de intrare către valorile de ieșire corespunzătoare, realizat prin combinarea mai multor funcții simple.

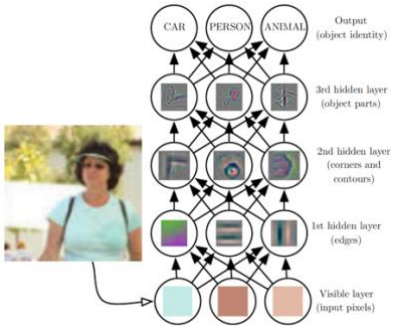


Figure 2.1. Exemplu de modul Deep Learning [1]

Învățarea profundă abordează această provocare prin descompunerea mapării complexe într-o serie de mapări mai simple, fiecare fiind gestionată de un strat diferit în model, așa cum se poate vedea în Figura 2.1.

Învățării automate se încadrează în categoria Inteligența Artificială, o tehnică ce permite sistemelor computerizate să se îmbunătățească în timp prin experiență și date. Învățarea profundă, un subset al învățării automate, își obține puterea și flexibilitatea învățând să reprezinte lumea ca o ierarhie de concepte imbricate [1].

2.1 Învățarea automată (Machine Learning)

2.1.1. Algoritmi de Machine Learning

Învățarea automată permite cercetătorilor și inginerilor să abordeze sarcini care sunt prea complexe pentru a fi rezolvate folosind programe fixe proiectate de oameni. Un exemplu constă într-un set de caracteristici măsurate cantitativ de la un obiect sau un eveniment pe care procesul de învățare automată trebuie să le proceseze. De exemplu, în cazul unei imagini, caracteristicile ar putea fi diferitele valori ale pixelilor din imagine [1].

Unele dintre sarcinile mai comune utilizate în învățarea automată sunt:

- Clasificare. Recunoașterea modernă a obiectelor a fost semnificativ îmbunătățită prin utilizarea tehnicii de Învățare Profundă [2][3]. Această tehnologie fundamentală alimentează și sistemele de recunoaștere facială [4], permițând etichetarea automată a indivizilor în colecțiile de fotografii [1].
- Regresie.
- Decizia de anomalie.
- Eliminarea zgomotului.

Algoritmii de învățare nesupervizată lucrează cu un set de date ce conține multe caracteristici și învață caracteristicile considerate folositoare, valoroase despre structura datelor. În învățarea profundă, scopul este adesea de a învăța distribuția probabilistică care a generat setul de date.

Algoritmii de învățare supervizată lucrează cu un set de date care include caracteristici, unde fiecare exemplu este însoțit de o etichetă sau un țintă.

Multe probleme de învățare automată devin din ce în ce mai provocatoare atunci când datele au un număr mare de dimensiuni. Această problemă este cunoscută sub numele de problema dimensionalității.

2.1.2. NLP

Natural Language Processing (NLP) se referă la utilizarea limbajelor umane de către un computer. Spre deosebire de algoritmii de programare specializați, concepuți pentru a fi interpretați în mod eficient și clar de computere, limbajele naturale sunt ambigue și rezistă formalizării. NLP include aplicații precum traducerile automate, unde un sistem trebuie să citească o propoziție într-o limbă umană și să genereze o propoziție echivalentă într-o altă limbă.

2.1.3. Optimizarea

Majoritatea algoritmilor de învățare profundă implică un anumit tip de optimizare. Optimizarea se referă la sarcina de a minimiza sau maximiza o funcție $f(x)$ prin ajustarea variabilei x.

Algoritmii de optimizare pot întâmpina dificultăți în găsirea unui minim global atunci când există multiple minime locale sau platouri. În învățarea profundă, este în general acceptabil să se opteze pentru aceste soluții, chiar dacă nu sunt minime absolute, dacă rezultă într-o valoare suficient de mică a funcției de cost [1].

2.1.4. Algoritmi cu rată de învățare adaptivă

Cercetătorii din domeniul rețelelor neuronale au recunoscut de mult timp că rata de învățare este unul dintre cei mai dificili hiperparametri de configurat, deoarece afectează semnificativ performanța modelului.

Algoritmul AdaGrad adaptează ratele de învățare ale parametrilor individuali ai modelului, scalându-le invers proporțional cu rădăcina pătrată a sumei gradientelor pătrate [12].

Adam [6] este un alt algoritm de optimizare cu rată de învățare adaptivă. Numele „Adam" vine de la „adaptive moments" (momente adaptive).

2.2 Referințe

[1] I. Goodfellow, Y. Bengio și A. Courville, Deep Learning.

[2] A. Krizhevsky, I. Sutskever și G. and Hinton, ImageNet classification with deep convolutional neural networks., In NIPS'2012 . 23, 24, 27, 100, 201, 371, 454, 458, 2012.

[3] S. Ioffe și C. and Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[4] Y. Taigman, M. Yang, M. Ranzato și L. and Wolf, DeepFace: Closing the gap to human-level performance in face verification. I, In CVPR'2014 ., 2014.

[5] J. Duchi, E. Hazan și Y. and Singer, Adaptive subgradient methods for online learning and stochastic optimization., Journal of Machine Learning Research, 2011.

[6] D. Kingma și J. and Ba, Adam: A method for stochastic optimization., arXiv preprint arXiv:1412.6980 ., 2014.

2.3 Posibile utilizări ale Inteligenței Artificiale în domeniul verificării funcționale [1]

Acest capitol reprezintă articolul tradus și adaptat al autorului: "Review of Artificial Intelligence enhancements in the field of Functional Verification".

2.3.1. Introducere

În lumea de astăzi, există o necesitate de a documenta cantități mari de date pentru a oferi servicii sau produse mai bune clienților.

Pentru a reduce măcar timpul petrecut pe verificarea funcțională, inteligența artificială ar putea fi utilizată pentru a extrage sau injecta, în funcție de aplicație, mesaje esențiale de informare de depanare în mediul de verificare sau pentru a ajuta inginerii să atingă acoperirea testelor maximă.

2.3.2. Probleme existente care consumă timp

În majoritatea cazurilor, mediul de verificare va necesita intervenția unui inginer pentru depanare din cauza unor probleme multiple, crescând astfel timpul petrecut pe acest proces.

În plus, o parte semnificativă de timp este consumată și în încercarea de a atinge acoperirea testelor maximă. Pentru aceasta, inițial se proiectează un plan de verificare, care conține „bins" de acoperire a testelor și propriu-zise testele care trebuie scrise și executate.

Acestea sunt două cazuri în care Inteligența Artificială ar putea juca un rol important în accelerarea procesului: unul fiind depanarea și celălalt atingerea unei acoperiri de 100% a testelor. În această lucrare vom examina ambele cazuri. Pot fi prezente și alte metode.

2.3.3. Posibile îmbunătățiri ale verificării funcționale cu ajutorul Inteligenței Artificiale

Învățarea profundă este un subset al Inteligenței Artificiale, care mapează caracteristicile de intrare (analoge cu variabilele de predicție în statisticile tradiționale) la un rezultat.

Există mai multe tipuri de rețele neuronale:

- Rețea neuronală feedforward.
- Rețea neuronală recurentă.
- Rețele neuronale convoluționale cu mai multe straturi, inclusiv un strat de convoluție, un strat de non-linearitate, un strat de pooling și un strat complet conectat.

2.3.4. Studiu asupra suportului de depanare folosind Inteligența Artificială

Clasificarea textului este o soluție ce trebuie luată în considerare atunci când se depanează medii complexe de verificare pentru a vedea ce limbaj a fost utilizat pentru fișierul specific [1].

Reyes et al. [2] au reușit să extragă caracteristicile dintr-un cod sursă preprocesat și au antrenat o RNN (Rețea Neuronală Recurentă) folosind ordinea secvențelor de text. Rezultatele au fost obținute în 2016, pe zece limbaje de programare diferite, 756 MB de fișiere folosind tehnica de clasificare a textului [1]. Dam et al. au folosit Natural Language Processing (NLP) [3], în special metodele n-grams și skip-grams, pentru a putea clasifica codul sursă în douăzeci de limbaje de programare, 14.000 de fișiere utilizând clasificarea codului text [1]. Baquero et al. [4] au propus un model care ar putea prezice limbajul de programare din datele de comentarii și fragmentele de cod [1]. Studiul a fost realizat pe optsprezece limbaje de programare, 18.000 de fișiere folosind metoda Support Vector Machine (SVM).

2.3.5. Clasificarea codului sursă bazat pe text

Pasul de preprocesare este esențial pentru datele de text. Este necesară o transformare a elementelor text în caracteristici numerice pentru a fi folosite ca intrări pentru algoritm. În acest pas se folosește o metodă de tokenizare pentru a împărți textul în cuvinte separate și/sau alte elemente. Încapsularea este procesul de creare a vectorilor care conțin numere reale ce reprezintă tokenurile create în pasul anterior. Fiecare cuvânt sau element este mapat într-un vector sau vectori care sunt învățați de Rețeaua Neuronală Convoluțională în timpul fazei de antrenare [1].

2.3.6 Clasificarea codului sursă bazată pe imagini

Printre alte avantaje ale abordării bazate pe imagini este și capacitatea de a interpreta comentariile din codul sursă furnizat. În timp ce în abordarea bazată pe text, comentariile adăugate în cod trebuie eliminate pentru a interpreta corect fișierul, aici Rețeaua Neuronală Convoluțională nu va avea această problemă. Mai mult, comentariile adăugate pot fi informații utile ignorate de clasificarea text.

2.3.7. Studiu asupra suportului de acoperire a testelor folosind Inteligența Artificială

După cum este menționat în această lucrare, acoperirea testelor este de asemenea laborioasă pentru inginer. Mai multe studii au explorat posibilitatea de a folosi o rețea neuronală pentru a îmbunătăți timpul petrecut pe acoperire.

În verificare, stimulii care trebuie aplicați la DUT sunt generați constrânși aleatoriu în test. O îmbunătățire rapidă care poate fi aplicată pentru a crește rapid acoperirea este integrarea CDG (Coverage Driven Generation) [5]. Wang et al. [5] au reconsiderat problema ASIC-urilor complexe, care conțin blocuri RTL (Register Transfer Layer) complicate, cu un număr mare de parametri și configurații și predispuși la multe defecte. Scopul lor a fost să dezvolte un sistem de verificare care să poată discerne între parametrii și configurațiile relevante și cele care nu sunt. Q. Guo et al. [6] au construit un cadru care intenționează să reducă stimulii irelevanți din mers. Dacă stimulii sunt categorisiți ca fiind redundanți, atunci nu vor fi aplicați la Design Under Test (DUT). Prin urmare, doar stimulii relevanți vor fi trimiși la DUT [6].

2.3.8. Îmbunătățirile folosind CUDA

În timp ce în secțiunile de mai sus au fost prezentate câteva soluții la problemele existente în domeniul verificării funcționale, această secțiune se referă la perspectiva de a reduce și mai mult timpul petrecut pe verificarea funcțională utilizând Inteligența Artificială, prin adoptarea platformei Computer Unified Device Architecture (CUDA) de la Nvidia.

2.3.9. Comparație între CUDA și OpenCL

Limbajul de Calcul Open (OpenCL) este un standard open-source pentru programarea paralelă cu scopuri multiple de la CPU-uri, GPU-uri și procesoare multiple.

OpenCL are multe similarități în comparație cu CUDA. În primul rând, arhitectura de fire de execuție a CUDA este aproape identică cu cea din OpenCL (work-item).

Unele diferențe între CUDA și OpenCL sunt legate de modelele de memorie. În cazul CUDA, memoria locală poate fi văzută de fiecare fir de execuție.

În ceea ce privește compilarea, CUDA folosește un compilator static care este responsabil de compilarea codului kernel și gazdă înainte de a fi trimis la GPU. Pe de altă parte, OpenCL folosește un compilator ahead-of-time (AOT) sau just-in-time (JIT) care oferă o portabilitate bună. Datorită AOT, OpenCL necesită timp suplimentar de inițializare pentru a detecta dispozitivele [7].

2.3.10. Concluzii

În această lucrare sunt prezentate multiple soluții de Inteligență Artificială și o posibilă optimizare pentru problemele întâmpinate în Verificarea Funcțională.

Lucrarea prezintă contextul pentru verificarea unui chip și provocările de a reduce numărul de erori la zero. Acest proces este laborios și consumă mult timp. Inteligența Artificială a fost analizată ca soluție pentru a reduce acest blocaj prin atingerea procentului de acoperire stabilit în planul de testare.

2.3.11. Referințe

[1] Elife Ozturk Kiyak, Ayse betul Cengiz, Kokten Ulas Birant, Derya Birant, “Comparison of Image – Based and Text-Based Source Code Classification Using Deep Learning”, SN Computer Science (2020)

[2] Reyes J, Ramirez D, Paciello J, “Automatic classification of source code archives by programming language: a deep learning approach”, International Conference on Computational Science and Computational Intelligence, IEEE, 2016.

[3] Dam V, Kennedy J, Zaytsev V, “Software language identification with natural classifiers”, International Con- ference on Software Analysis, Evolution, and Reengineering (SANER), 2016.

[4] Baquero JF, Camargo JE, Restrepo-Calle F, Aponte JH, Gonzalez FA, “Predicting the programming language: Extracting knowledge from stack overflow posts”, Colombian Conference on Computing, Springer, 2017

[5] Wang, F., Zhu, H., Popli, P., Xiao, Y., Bodgan, P., & Nazarian, S. (2018), “Accelerating Coverage Directed Test Generation for Functional Verification”, Proceedings of the 2018 on Great Lakes Symposium on VLSI – GLSVLSI

[6] Guo, T. Chen, H. Shen, Y. Chen and W. Hu, "On-the-Fly Reduction of Stimuli for Functional Verification," 2010 19th IEEE Asian Test Symposium, 2010, pp. 448-454, doi: 10.1109/ATS.2010.82.

[7] Yan W, Shi X, Yan X, Wang L. “Computing OpenSURF on OpenCL and General Purpose GPU”, International Journal of Advanced Robotic Systems. October 2013.

2.3.12 Resurse

Costurile au fost acoperite de autori.

Capitolul 3. Large Language Models și aplicații în verificarea funcțională [2]

Acest capitol a fost tradus și adaptat dupa articolul autorului: "Trials of using Generative AI for APB UVM testbench generation".

Procesul de verificare funcțională constă din mai multe părți, precum elaborarea planului de verificare, implementarea mediului, implementarea testelor, depanarea, raportarea problemelor găsite și remedierea erorilor de regresie. Pe măsură ce designul avansează în dezvoltare, acesta este integrat în mediul de verificare, fiind astfel testat pentru erori. Modul în care se realizează verificarea funcțională este prezentat în Figura 3.2 (Mammo, 2017).

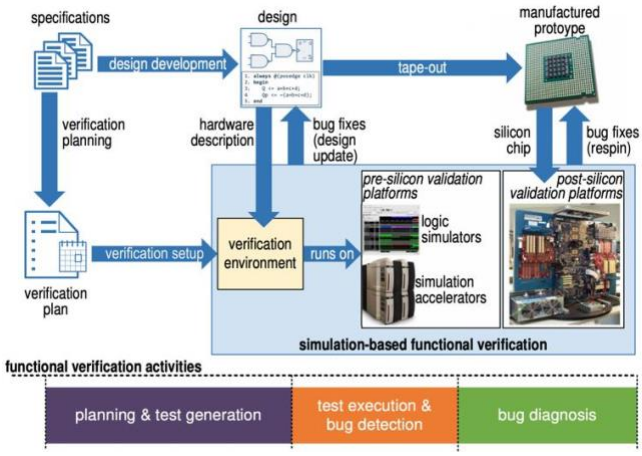


Figure 3.1. Verification process steps (Mammo, 2017)

3.1. Inteligența Artificială în verificarea funcțională și alte cercetări

Inteligența artificială poate îmbunătăți considerabil procesul de verificare în multiple moduri. Poate ajuta în diverse domenii ale verificării:

- Completarea acoperirii testelor. În această lucrare (Dinu et al., 2022), sunt utilizate abordări cu algoritmi genetici pentru a genera stimuli care sunt trimiși la portul de intrare al Device Under Test (DUT).

- Clustering și filtrare a regresiilor. Diferite algoritmi de învățare automată pot fi utilizați pentru a filtra diverse erori ale testelor din regresie (Dinu & Ogrutan, 2019) și pentru a se concentra doar pe erorile mai importante;

- În documentație, recunoașterea modelelor pe imagini și texte. Prin utilizarea acestora, sarcina de verificare a proiectului poate fi definită mai bine și mai eficient;
- Generarea diverselor componente de testare folosind AI Generativ. Această abordare poate fi utilizată pentru a genera de la zero componente de testare System Verilog UVM complete folosind ChatGPT4, astfel încât inginerul să analizeze doar codul furnizat și să integreze mediul de verificare rezultat în sistemul respectiv.

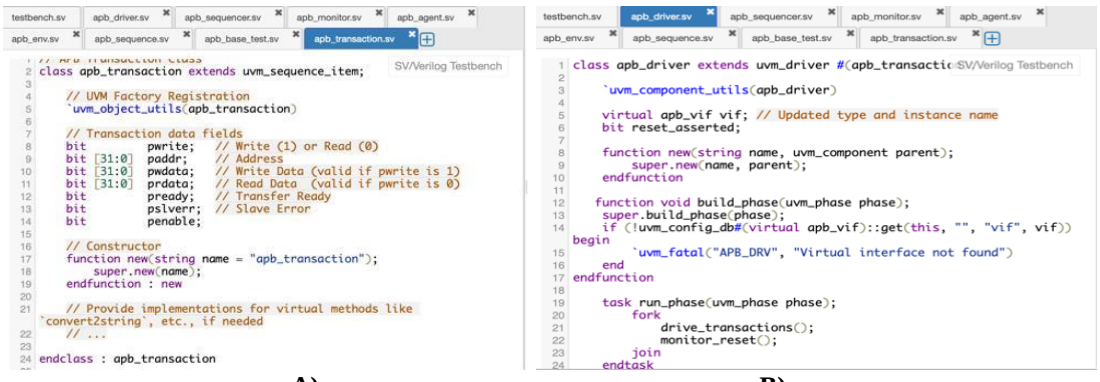
3.2. Stadiul actual al tehnologiei

Implementările cu Large Language Models (LLM), precum ChatGPT, au devenit opțiuni viabile pentru ingineri pentru a naviga sau a genera cod complex, bogat în funcționalități (Khurana et al., 2024).

În această lucrare, a fost utilizată abordarea cu AI Generativ pentru a genera de la zero un mediu de verificare System Verilog UVM ARM AMBA (Advanced Microcontroller Bus Architecture) APB (Advanced Peripheral Bus), împreună cu un studiu privind generarea codului System Verilog UVM folosind o diagramă de forme de unde regăsită de obicei în documentația oficială.

3.3. Generarea unui mediu de verificare UVM de la zero folosind ChatGPT4

În această lucrare, majoritatea componentelor UVM au fost implementate, cu excepția scoreboard-ului. Aceste componente au fost generate integral folosind ChatGPT4. Singura parte care a trebuit ajustată manual din cauza limitărilor ChatGPT4 este fișierul top.sv, unde semnalele sunt conectate la DUT și la interfața virtuală. Codul a fost analizat și corectat de către inginerul de verificare. Codul a fost simulat folosind site-ul open-source EDA Playground, care oferă capacități de simulare limitate, dar suficiente, cu toate cele trei simulatoare ale marilor vendori (Cadence INCISIV, Questa, VCS). Mai jos sunt fragmente din codul generat de ChatGPT4.



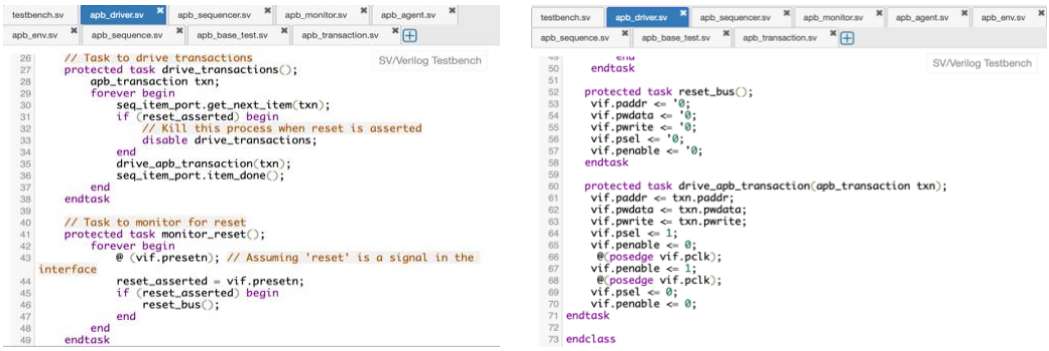
A) B)

Figure 3.2 Clasele item și driver generate de ChatGPT4

ChatGPT4 a fost solicitat să genereze o clasă de item sau tranzacție folosind UVM System Verilog care respectă protocolul AMBA APB. La prima interogare a ChatGPT4, modelul a generat doar semnalele pwrite, paddr, pdata, prdata și pready, care sunt incomplete deoarece semnalele pslverr și penable sunt necesare. La a doua execuție, modelul a adăugat cele două semnale lipsă în uvm_sequence_item, conform cerinței.

În clasa apb_driver, modelului i s-a cerut în mod similar să furnizeze codul specificând clar utilizarea Metodologiei UVM și a limbajului SystemVerilog pentru verificare. Ieșirea a fost, de asemenea, incompletă, deoarece funcționalitatea de resetare a protocolului, indispensabilă, lipsea. Părțile de cod lipsă au fost adăugate de către inginerul de verificare.

Figura 3.4 A), B) și Figura 3.5 A), B) ilustrează atât clasele de tranzacție, cât și cele de driver rezultate prin utilizarea modelului LLM și corectarea elementelor lipsă.



A) **B)**

Figure 3.3. Clasa Driver generată

Pentru a completa acest mediu de verificare APB UVM, inginerul de verificare a solicitat modelului o secvență, pentru ca aceasta să fie utilizată într-un test.

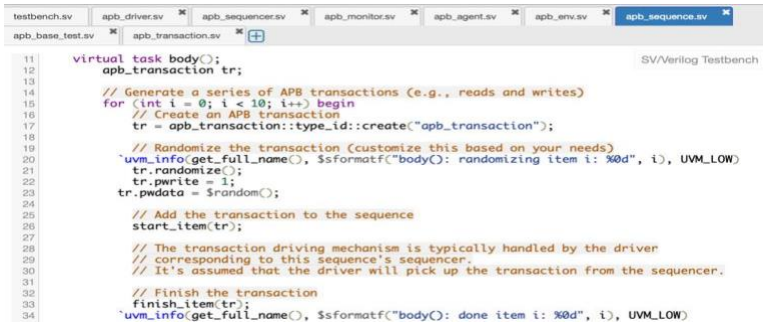


Figure 3.4. UVM Sequence

Secvența din Figura 3.6 a fost utilizată într-un test UVM APB generat de un model LLM. După rularea simulării, formele de undă au fost salvate pentru a fi

analizate și pentru a asigura corectitudinea bancului de testare. Figura 3.7 prezintă formele de undă.

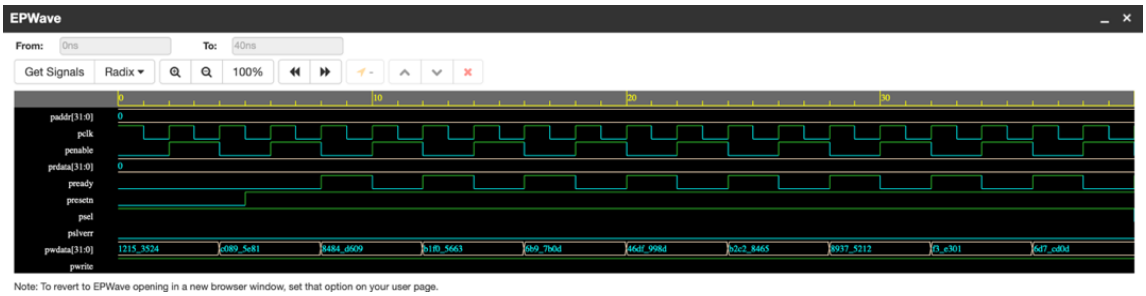


Figure 3.5. Rezultatele mediului de verificare generat cu un LLM

Singurul fișier care a trebuit creat manual de către inginer a fost fișierul top.sv. ChatGPT4 poate înlocui sau corecta numele destul de ușor, dar, așa cum este necesar într-un fișier top.sv, nu poate conecta interfața la interfața virtuală, folosește cod greșit și necompilabil pentru uvm_set_config_db() și, de asemenea, pentru metoda run_test().

3.4. Generating various components of a UVM testbench starting from a protocol diagram

În acest studiu, o imagine a protocolului a fost introdusă în model, solicitându-i LLM – ului să furnizeze driverul UVM corespunzător. Această abordare este utilă atunci când se adaugă o componentă UVM individuală într-un mediu de verificare deja existent. În acest fel, codul este generat folosind fie o imagine, o diagramă sau poate forme de undă și va fi importat în mediul de verificare existent. Această abordare economisește timp, în loc să se scrie componenta UVM de la zero. Figura 3.8 conține imaginea utilizată, iar Figura 3.9 A), B) codul rezultat.

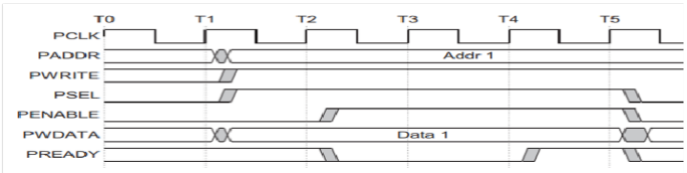
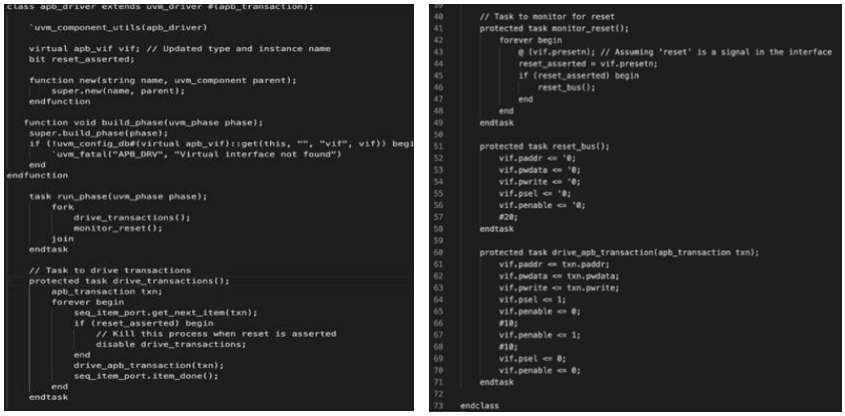


Figure 3.6. Forme de undă a unui transfer APB (ARM, 2023)



```
class apb_driver extends uvm_driver #(apb_transaction);

    `uvm_component_utils(apb_driver)

    virtual apb_vif vif; // Updated type and instance name
    bit reset_asserted;

    function new(string name, uvm_component parent);
        super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    if (!uvm_config_db#(virtual apb_vif)::get(this, "", "vif", vif)) begi
        `uvm_fatal("APB_DRV", "Virtual interface not found")
    end
endfunction

    task run_phase(uvm_phase phase);
        fork
            drive_transactions();
            monitor_reset();
        join
    endtask

    // Task to drive transactions
    protected task drive_transactions();
        apb_transaction txn;
        forever begin
            seq_item_port.get_next_item(txn);
            if (reset_asserted) begin
                // Kill this process when reset is asserted
                disable drive_transactions;
            end
            drive_apb_transaction(txn);
            seq_item_port.item_done();
        end
    endtask
```

A)

```
    // Task to monitor for reset
    protected task monitor_reset();
        forever begin
            @ (vif.presetn); // Assuming 'reset' is a signal in the interface
            reset_asserted = vif.presetn;
            if (reset_asserted) begin
                reset_bus();
            end
        end
    endtask

    protected task reset_bus();
        vif.paddr <= '0;
        vif.pwdata <= '0;
        vif.pwrite <= '0;
        vif.psel <= '0;
        vif.penable <= '0;
        #20;
    endtask

    protected task drive_apb_transaction(apb_transaction txn);
        vif.paddr <= txn.paddr;
        vif.pwdata <= txn.pwdata;
        vif.pwrite <= txn.pwrite;
        vif.psel <= 1;
        vif.penable <= 0;
        #10;
        vif.penable <= 1;
        #10;
        vif.psel <= 0;
        vif.penable <= 0;
    endtask

endclass
```

B)

Figure 3.7. Clasele generate

3.5. Concluzii

Cele două abordări pot reduce timpul petrecut pe verificarea funcțională. Prima abordare, în care un mediu de verificare UVM APB este dezvoltat de la zero, ar putea fi mai potrivită atunci când se dorește mai mult control asupra a ceea ce LLM – ul este solicitat să genereze. Prin utilizarea unei abordări bazate pe text, în care LLM este întrebat în mod specific despre clasele implicate, metodologia utilizată și limbajul de programare, inginerul de verificare poate avea mai mult control și petrece mai puțin timp corectând ChatGPT4. A doua abordare, folosind o imagine a protocolului (în acest caz, un transfer ARM APB Write) prezentă în multe documentații, poate fi foarte utilă atunci când sunt necesare solicitări de modificare, întrucât componentele generate pot fi ușor inserate într-un mediu de verificare existent.

Această lucrare a avut ca scop să demonstreze, pe un mediu de verificare, beneficiile și posibilele cazuri de utilizare ale ChatGPT4 în procesul de verificare funcțională. Așa cum este prezentat mai sus, inginerul de verificare a utilizat două abordări, ambele cu rezultate foarte bune și reducând timpul petrecut pe proiect.

3.6. Referințe

Dinu, A. & Ogrutan, P. L. (2019) Opportunities of Using Artificial Intelligence in Hardware Verification. In: *2019 IEEE 25th International Symposium for Design and Technology in Electronic Packaging (SIITME), October 23-26, 2019, Cluj-Napoca, Romania*. pp. 224-227, doi: 10.1109/SIIME47687.2019.8990751 [Accessed on March 2024].

Dinu, A., Danciu, G. M., Ogrutan, P. L. (2022) Cost-Efficient Approaches for Fulfillment of Functional Coverage during V erification of Digital Designs. *Micromachines*. 13(5), 691. doi:10.3390/mi13050691.

Mammo, B. W. (2017) *Reining in the Functional Verification of Complex Processor Designs with Automation, Prioritization, and Approximation.* Ph.D. thesis, University of Michigan. [Accessed on March 2024].

Capitolul 4. Generarea de aserții SystemVerilog utilizând Large Language Models [3]

Acest capitol reprezintă adaptarea și traducerea articolului autorului: "Generative AI Assertions in UVM-based System Verilog Functional Verification".

4.1. Introducere

Generarea de aserții pentru diferite protocoale, cum ar fi Advanced Peripheral Bus (APB), AHB (Advanced High-Performance Bus) etc., folosind un Large Language Model (LLM) precum ChatGPT, poate reduce considerabil timpul de verificare funcțională. Aceste aserții pot fi integrate și în alte medii de verificare; astfel, este util să se scrie codul de aserții într-un mod corect și reutilizabil.

4.2. Materiale și Metode

În acest studiu, ChatGPT a fost solicitat printr-un prompt să genereze aserții System Verilog pentru un banc de testare UVM APB. Figura 4.3 prezintă modul în care LLM a fost solicitat să furnizeze rezultatele.

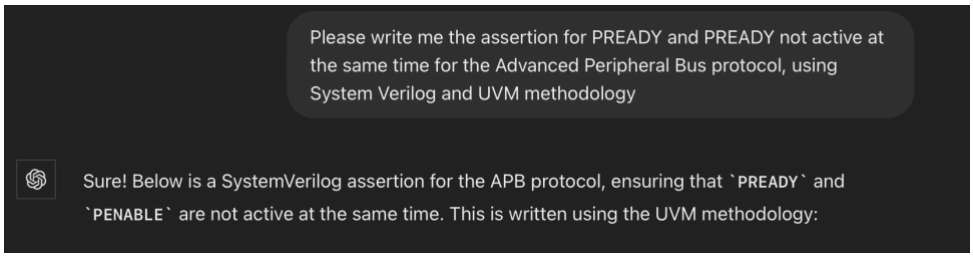


Figure 4.1. ChatGPT text prompt

Există o altă modalitate de a utiliza LLM, cum ar fi furnizarea unei forme de undă a protocolului pentru a permite ChatGPT să interpreteze regulile formei de undă și să ofere afirmațiile necesare. Figura 4.4 prezintă formele de undă ale tranzacției de transfer Write din protocolul APB introduse în LLM, cerându-i doar lui ChatGPT să genereze aserțiile fără a scrie nicio definiție a protocolului sau reguli. Sarcina LLM-ului a fost să interpreteze forma de undă și să genereze aserțiile conform regulilor implicate de forma de undă.

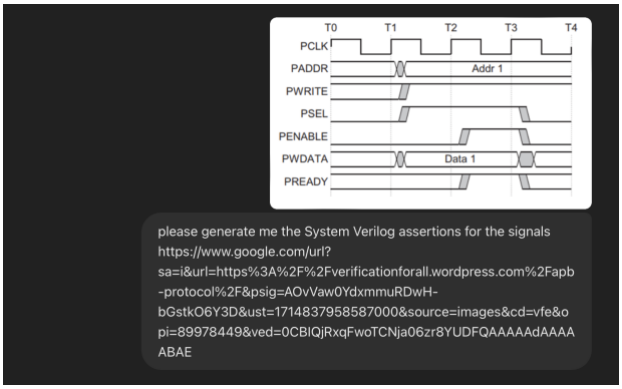


Figure 4.2. ChatGPT prompt fed with APB Write transfer waveform

Utilizând aceste două abordări, această lucrare își propune să evidențieze cât de rapid poate genera LLM cod critic, reducând considerabil timpul petrecut pentru verificare. Mediul de verificare și implementarea design – ului (RTL) au fost dezvoltate de cercetători.

4.3. Rezultate

Utilizând prima abordare, LLM-ului i s-a cerut prin text să furnizeze aserții pentru diferite semnale ale protocolului APB. Rezultatul obținut de la LLM pentru o aserție care are condiția ca semnalul PENABLE să nu fie activ în timpul resetării este descris în Figura 4.5

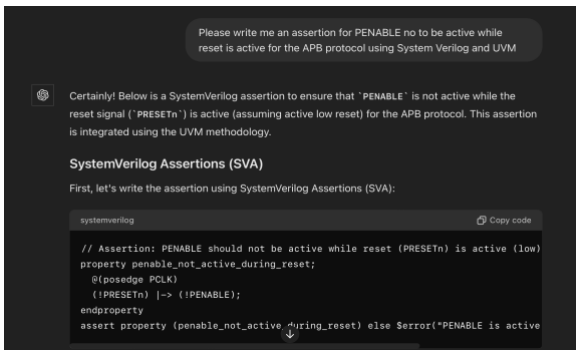


Figure 4.3. Rezultatele ChatGPT

Codul generat de LLM a fost introdus în fișierul top.sv al testbench-ului, unde se află modulul de top, pentru o integrare mai ușoară.

```
66 // Assertion: PENABLE should not be active while reset (PRESETn) is
   active (low)
67 property penable_not_active_during_reset;
68   @(posedge clk)
69   (!rst) |-> (!vif.penable);
70 endproperty
71 assert property (penable_not_active_during_reset) else $error("PENABLE
   is active while reset is asserted");
72 endmodule
73
```

Figure 4.4. Codul generat de ChatGPT, integrat în modulul top

Un test a fost rulat cu codul din Figura 4.6 activ. Testul a eșuat, deoarece aserția generată de ChatGPT – 4 a fost invalidată. În Figura 4.7 este prezentat log – ul

testului respectiv. Se poate vedea clar că aserția ChatGPT – 4 a indicat corect eroarea în test.

```
UVM_INFO apb_agent.sv(24) @ 0: uvm_test_top.env.apb_agt [uvm_test_top.env.apb_agt] build_phase(): created drv and seqr
UVM_INFO apb_sequence.sv(20) @ 0: uvm_test_top.env.apb_agt.seqr@@apb_seq [uvm_test_top.env.apb_agt.seqr.apb_seq] body(): randomizing item i: 0
UVM_INFO apb_sequence.sv(34) @ 4: uvm_test_top.env.apb_agt.seqr@@apb_seq [uvm_test_top.env.apb_agt.seqr.apb_seq] body(): done item i: 0
UVM_INFO apb_sequence.sv(20) @ 4: uvm_test_top.env.apb_agt.seqr@@apb_seq [uvm_test_top.env.apb_agt.seqr.apb_seq] body(): randomizing item i: 1
xmsim: *E,ASRTST (./testbench.sv,71): (time 4 NS) Assertion top_module.penable_not_active_during_reset has failed
PENABLE is active while reset is asserted
UVM_INFO apb_sequence.sv(34) @ 8: uvm_test_top.env.apb_agt.seqr@@apb_seq [uvm_test_top.env.apb_agt.seqr.apb_seq] body(): done item i: 1
```

Figure 4.5. Log – ul de erori al testului

Când s-a cerut LLM-ului printr-un prompt text să genereze aserții de bază pentru protocolul AMBA APB, rezultatul furnizat de LLM pentru această sarcină este prezentat în Figura 4.8. Figura prezintă LLM-ul furnizând aserții împreună cu explicații pentru fiecare punct și structurând codul cu atenție.

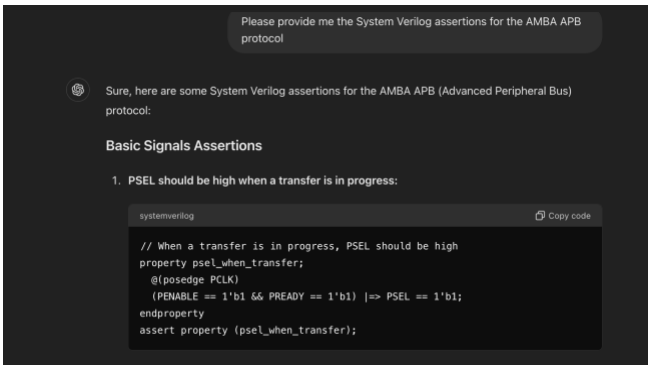


Figure 4.6. Text Prompt

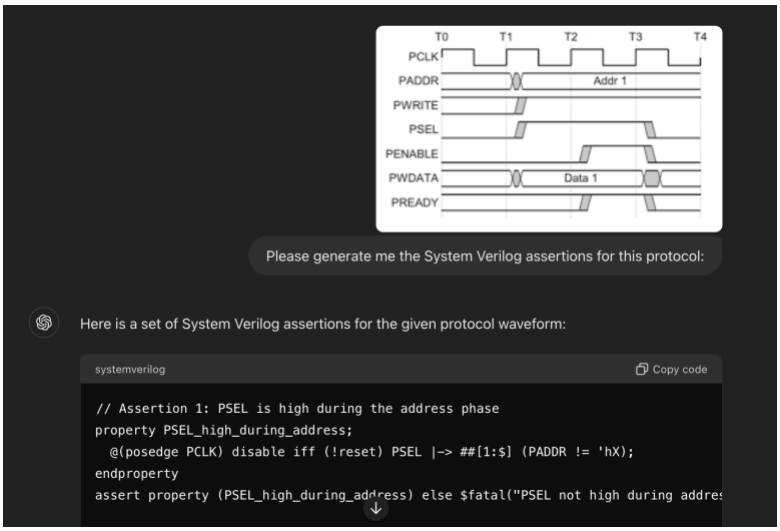
Aserții furnizate de LLM au fost integrate în mediul de verificare APB. Acest lucru este arătat în Figura 4.9. Figura 4.9 prezintă toate aserțiunile de bază SystemVerilog pentru AMBA APB integrate în fișierul top.sv. LLM-ul a sugerat cinci aserțiuni de bază, cum ar fi: PSEL ar trebui să fie activ când un transfer este în curs, PENABLE ar trebui să fie activat doar după ce PSEL este activat, PADDR ar trebui să fie stabil când PSEL și PENABLE sunt active, PWRITE ar trebui să fie stabil când PSEL și PENABLE sunt active, PWDATA ar trebui să fie stabil în timpul fazei de activare a transferului.

```
// Assertion: PENABLE should not be active while reset (PRESETn) is
active (low)
property penable_not_active_during_reset;
  @(posedge clk)
  (!rst) |-> (!vif.penable);
endproperty
assert property (penable_not_active_during_reset) else $error("PENABLE
is active while reset is asserted");
// When a transfer is in progress, PSEL should be high
property psel_when_transfer;
  @(posedge clk)
  (vif.penable == 1'b1 && vif.pready == 1'b1) |=> vif.psel == 1'b1;
endproperty
assert property (psel_when_transfer)
// PENABLE should be asserted only after PSEL is asserted
property penable_after_psel;
  @(posedge pclk)
  vif.psel == 1'b1 |=> vif.penable == 1'b1;
endproperty
assert property (penable_after_psel);
// PADDR should be stable when PSEL and PENABLE are high
property paddr_stable;
  @(posedge pclk)
  (vif.psel == 1'b1 && vif.penable == 1'b1) |=> $stable(vif.paddr)
endproperty
assert property (paddr_stable);
  // PWRITE should be stable when PSEL and PENABLE are high
property pwrite_stable;
  @(posedge pclk)
  (vif.psel == 1'b1 && vif.penable == 1'b1) |=> $stable(vif.pwrite);
endproperty
assert property (pwrite_stable);
  // PWDATA should be stable during the enable phase of a write transfer
property pwdata_stable;
  @(posedge pclk)
  (vif.psel == 1'b1 && vif.penable == 1'b1 && vif.pwrite == 1'b1) |=>
$stable(vif.pwdata);
endproperty
assert property (pwdata_stable);
endmodule
```

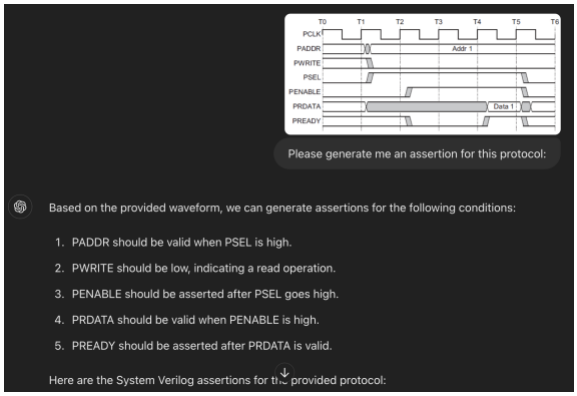
Figure 4.7. Integrarea aserțiilor in mediul de verificare

Pentru a putea rezolva problemele de compilare, elaborare și simulare, cercetătorii au redenumit semnalele din PCLK, PENABLE, etc., în clk, vif.penable. Aceasta a fost făcută pentru a corecta variabilele astfel încât să indice interfața virtuală APB a testbench-ului. Testul a fost rulat, cu erori declanșate de aserțiunile sugerate de ChatGPT – 4.

În a doua abordare a acestei lucrări, LLM-ului i s-au furnizat o imagine cu un transfer de scriere AMBA APB și o imagine cu un transfer de citire AMBA APB. Nu au fost adăugate alte detalii, doar imaginile în sine și cererea de a genera aserții în SystemVerilog. Figura 4.10 A) și B) prezintă modul în care LLM-ului i s-a cerut să furnizeze codul și imaginile furnizate acestuia.



A). Image prompt pentru un transfer Write



B). Image prompt pentru un transfer Read

Figure 4.8. Image prompt pentru APB Read & Write Transfer

Pentru transferul de scriere, ChatGPT – 4 a generat aserții SystemVerilog care verifică funcționalitatea semnalelor PADDR, PWRITE, PENABLE, PWDATA și PREADY. ChatGPT – 4 a generat corect codul pentru aserții, deși uneori sunt necesare câteva modificări, în principal pentru integrarea în testbench-ul existent.

Deși aceste două abordări generează rezultate rapide și destul de precise, este necesar ca inginerul de verificare să supravegheze rezultatele modelului pentru a le corecta, dacă este cazul. Astfel, LLM-ul servește mai degrabă ca un copilot.

4.4. Concluzii

În această lucrare, cercetătorii au demonstrat impactul utilizării ChatGPT – 4 pentru generarea de aserții SystemVerilog și integrarea lor într-un testbench APB UVM existent. Prin generarea de aserțiuni, timpul petrecut pe verificarea funcțională a fost redus la aproximativ 20% pentru testbench-ul prezentat în această lucrare (testbench-ul APB UVM), fără a implementa o clasă de scoreboard dedicată și bazându-se doar pe aserții. Acest mediu rulează pe o platformă gratuită, ceea ce face dezvoltarea mai dificilă decât într-un mediu de simulare dedicat. Luând în considerare aceste aspecte, utilizarea Inteligenței Artificiale Generative pentru crearea de aserții o face un candidat excelent pentru a reduce considerabil timpul petrecut pe verificarea funcțională.

Capitolul 5. AI utilizat în verificarea funcțională [4]

Acest capitol este tradus și adaptat conform articolului autorului: "Artificial Intelligence Application in the Field of Functional Verification".

5.1. Introducere

Pe măsură ce dezvoltarea circuitelor integrate progresează în complexitate și scalabilitate, devine esențial ca acestea să funcționeze conform așteptărilor, fiind fiabile și robuste, în special în domenii critice, cum ar fi cel medical, aviatie și auto.

Prin utilizarea Inteligenței Artificiale, procesul de verificare poate fi mai rapid și finalizat într-un timp mai scurt.

5.2. Alte lucrări în domeniu

Există o multitudine de domenii în care Inteligența Artificială poate ajuta procesul de verificare. De exemplu, testele pot fi grupate în funcție de motivele pentru care au generat erori. Așa cum este menționat în [1], unii algoritmi de învățare automată pot clasifica eficient testele cu erori într-o regresie, în funcție de motivul și clasa erorilor lor. Totuși, conform [2], rezultatul nu este conform așteptărilor. Generarea de stimuli și teste este realizată folosind algoritmi ML de supervizare și de întărire [3] pentru a atinge acoperirea planificată pentru Dispozitivul Testat (DUT) [4]. În procesul de verificare a unui controller cache, o Rețea Neuronală Profundă (DNN) supervizată[5] a fost utilizată alături de algoritmul Q-learning. DNN-ul este antrenat să creeze secvențe pentru patru structuri FIFO (First-In-First-Out).

5.3. Metodologii, Materiale de Studii și Seturi de Date

În această lucrare, cercetătorii au implementat setul de date împreună cu modelul. Setul de date constă în informații importante sau critice din documentație cât și informații neesențiale. Diferite specificații și documentații au fost analizate și combinate pentru a crea setul de date. Astfel de documentații includ, de exemplu, arhitectura vast utilizate în industrie precum ARM, APB și AXI. Specificațiile au fost analizate amănunțit de inginer, iar informațiile principale și datele irelevante au fost clasificate în informații relevante și irelevante.

Figura 5.6 este un exemplu de alte date relevante care trebuie luate în considerare de inginer. Aceasta conține o descriere a semnalelor PSLVERR, PREADY și PENABLE. Acestea sunt semnale utilizate de obicei într-un transfer APB.

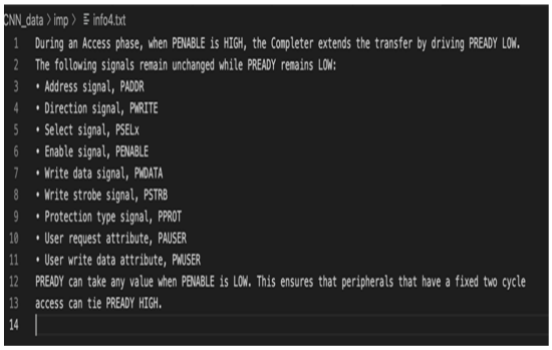


Figure 5.1. Descriere scurta a semnalelor APB.

Acestea sunt câteva exemple de date relevante din baza de date creată. În ceea ce privește datele irelevante, acestea pot conține părți precum reguli de copyright care, pentru inginerul de verificare, nu sunt aplicabile atunci când implementează mediul de verificare, iar scenariile de testare. De asemenea, anumite părți pot să nu fie implementate în design (Register Transfer Layer - RTL). Din diverse motive, necesități ale clientului, probleme legate de cost, unele părți ale diferitelor module, protocoale etc. pot să nu fie implementate în RTL. Figura 5.7 – Figura 5.9 ilustrează un exemplu de informații irelevante menționate mai sus.

Figure 5.2. Lista succinta de reguli ale protocolului APB

Figure 5.3. Informație de Copyright AMBA

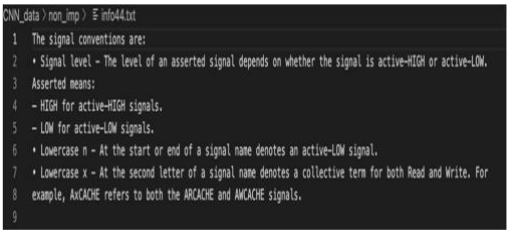


Figure 5.4. AMBA AXI convenții asupra semnalelor

Având în vedere mijloacele de mai sus, a fost construită de la zero o bază de date folosind expertiza și experiența inginerului de verificare în domeniu. Pentru acest set de date, informațiile au fost extrase cu atenție din Advanced Microcontroller Bus Architecture Advanced Peripheral Bus (AMBA APB) și din Advanced Microcontroller Bus Architecture Extensible Interface (AMBA AXI). Acest set de date va fi utilizat pentru a antrena modelul prezentat în această lucrare.

5.4. Rezultate

În aplicația prezentată, o etapă preprocesarea a datelor a fost efectuată înainte de a le introduce în model.

Datele au fost augmentate folosind biblioteci speciale pentru a crește numărul de elemente bazei de date. Pentru aceasta, a fost utilizată biblioteca nlpaug împreună cu o metodă care poate înlocui cuvinte cu sinonime ale acestora, care a fost aplicată pe setul de date. Etichetele au fost codificate folosind LabelEncoder, iar datele au fost apoi tokenizate. Datele de test și de antrenament au fost împărțite în 80% pentru antrenament și 20% pentru validare.

Au fost testate două implementări de model secvențial CNN. Arhitectura primului model CNN utilizat este descrisă în Figura 5.11. Modelul secvențial este un model de învățare profundă format din straturi stivuite liniar. Straturile sunt adăugate modelului în ordine secvențială, iar ieșirea fiecărui strat este intrarea stratului următor. Primul model propus constă într-un strat de embedding, un strat de convoluție, un strat de activare, Flatten și un strat Dense. Pentru stratul de activare a fost utilizată funcția de activare Rectified Linear Unit (ReLU).

Figure 5.5. Prima arhitectură CNN

Primul model propus a rulat 20 de epoci, având la final o acuratețe de 98,333340883255% și o pierdere de 10,25% cu augmentarea datelor aplicată o singură dată și concatenată cu setul de date creat. Acest prim model CNN este o încercare pentru a vedea ce trebuie făcut în continuare pentru a îmbunătăți acuratețea și pierderea modulelui. Există câteva opțiuni, cum ar fi augmentarea și mai mult a datelor, adăugarea mai multor straturi sau hiper-parametrizarea modelului. Figura 5.12 descrie graficul de acuratețe pentru antrenare și validare, precum și graficul de pierdere pentru antrenare și validare (loss).

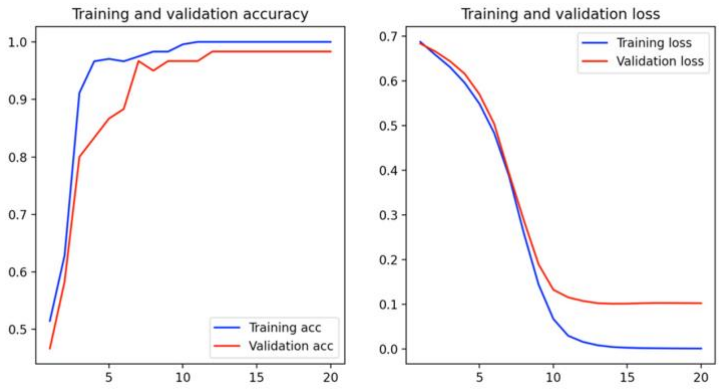


Figure 5.6. Primul model CNN, graficul pentru validare și loss

Modelul prezintă procente ridicate de precizie—97%, recall—100%, scor F1—98% pentru clasa 0, iar pentru clasa 1, precizie—100%, recall—96%, scor F1—98%. Aceste rezultate fac modelul fiabil pentru ambele cazuri, având o precizie, un recall și un scor F1 ridicate, performând foarte bine în identificarea cazurilor relevante și realizând predicții precise, având un echilibru bun între precizie și recall (atingând atât o precizie ridicată, cât și un procent de recall ridicat în același timp).

Un alt model secvențial a fost dezvoltat pentru a încerca să obțină rezultate mai bune. A fost definită o arhitectură folosind trei straturi de convoluție, trei straturi de activare, un strat de embedding, un strat de MaxPooling, un strat Flatten și un strat Dense, așa cum este prezentat în Figura 5.13

Figure 5.7. A doua arhitectură CNN

După aplicarea augmentării datelor, am împărțit setul de date în 80% pentru antrenament și 20% pentru validare și am antrenat modelul cu o acuratețe de 96,66666984558105% și loss de 5%.

Această arhitectură de model folosește trei straturi de convoluție, are procente de precizie foarte ridicate pentru ambele clase 1 și 0 (100% și 93%), un procent ridicat de recall (94% și 100%) și un scor F1 bun (97%).

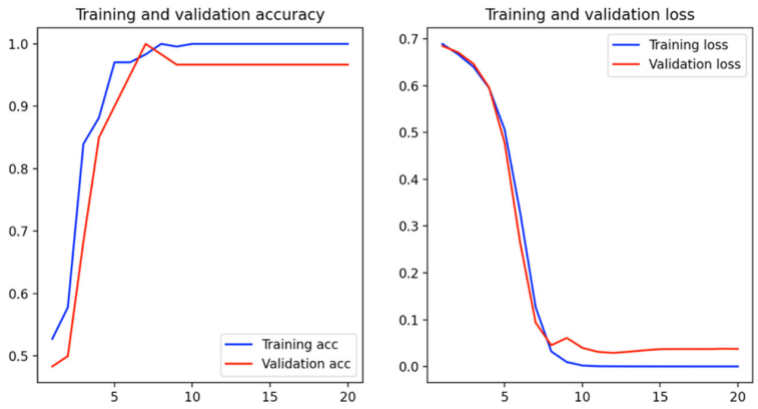


Figure 5.8. A doua arhitectură CNN graficul de antrenare și validare

În această lucrare [6], au fost utilizate o multitudine de modele pentru a clasifica datele binare pentru două seturi de date: DATASET-1 (similar cu Stanford Sentiment Treebank, având doar recenzii negative și pozitive) și DATASET-2 (recenzii Amazon din setul de date Sentiment Analysis). Cele mai bune modele aplicate pe aceste seturi de date, TextConvoNet_4 (patru straturi de convoluție) și TextConvoNet_6 (șase straturi de convoluție), au obținut procente de acuratețe de 82,2% și 81,9%, respectiv, pe DATASET-1. Pentru DATASET-2, TextConvoNet_4 are un procent de acuratețe de 90,4%, în timp ce TextConvoNet_6 are un procent de acuratețe de 87,2%. Dacă aceste rezultate sunt luate în considerare, cele două arhitecturi CNN propuse ar avea rezultate mai bune decât modelele TextConvoNet_4 și TextConvoNet_6. Acuratețea pe care au atins-o modelele Bidirectional Encoder Representations from Transformers (BERT), Hierarchical Attention Networks (HAN) și BerConvoNet este de 77,6%, 80,2% și 83,1% pe DATASET-1. Pentru DATASET-2, aceleași modele au atins 77,2% (Bidirectional Encoder Representations from Transformers-BERT), 86,3% (HAN) și 88,3% (BerConvoNet). Prin urmare, modelele prezentate în această lucrare au performanțe mai bune decât cele utilizate în [6].

5.5. Concluzii

În această lucrare, au fost utilizate două implementări AI pentru a ajuta procesul de verificare, cu scopul de a dimunua timpul petrecut în acest proces. Cele două abordări au constat în utilizarea a două arhitecturi de rețele neuronale convoluționale, una cu două straturi de convoluție și una cu trei straturi de convoluție. În ambele cazuri, s-a realizat augmentarea datelor. Aceasta înseamnă că, pe setul de date original, a fost aplicată o metodă de substituire a unor cuvinte cu sinonimele lor în limba engleză pentru a îmbunătăți setul de date fără a adăuga intrări noi. Setul de date a fost creat, dezvoltat și analizat de cercetători, folosindu-și experiența în domeniul verificării funcționale. Ambele modele de rețele neuronale convoluționale au fost dezvoltate folosind Python 3.12.3 și au rulat pe hardware Apple M1, pe un GPU dedicat de 10 nuclee, într-un mediu Anaconda.

Pentru prima rețea neuronală convoluțională, a fost definit un model secvențial, și straturile au fost adăugate acestuia. Această implementare a modelului a generat rezultate bune, cu un procent acuratețe ridicată la validare de

98,333340883255%, un procent recall, o precizie și un scor F1 ridicate, și un loss scăzut la validare (10,25%), ceea ce îl face un model fiabil, robust și bine ajustat.

A doua arhitectură a generat un procent de acuratețe ridicată de 96,66666984558105% și o pierdere scăzută la validare de 5%. Procentele pentru precizie, recall și scorul F1 sunt, de asemenea, bune în ansamblu, dovedin că modelul este unul de încredere și robust.

5.6. Referințe

[1] - Truong, A.; Hellström, D.; Duque, H.; Viklund, L. Clustering and Classification of UVM Test Failures Using Machine Learning Techniques. In Proceedings of the Design and Verification Conference (DVCON), San Jose, CA, USA, 26 February–1 March 2018.

[2] - El Mandouh, E.; Maher, L.; Ahmed, M.; ElSharnoby, Y.; Wassal, A.G. Guiding Functional Verification Regression Analysis Using Machine Learning and Big Data Methods. In Proceedings of the Design and Verification Conference and Exhibition Europe (DVCon), Munchen, Germany, 25 September 2018.

[3] - Ismail, K.A.; Ghany, M.A.A.E. Survey on Machine Learning Algorithms Enhancing the Functional Verification Process. Electronics2021, 10, 2688. [CrossRef]

[4] - Zaruba, F.A. An Open-Source 64-bit RISC-V Application Class Processor and Latest Improvements; ETH: Zurich, Switzerland, 2018.

[5] - Hughes, W.; Srinivasan, S.; Suvarna, R.; Kulkarni, M. Optimizing Design Verification using Machine Learning: Doing better than Random. In Proceedings of the Design and Verification Conference (DVCON-Europe), Virtual Conference, 26–27 October 2021.

[6] - Soni, S.; Chouhan, S.S.; Rathore, S.S. TextConvoNet: A convolutional neural network based architecture for text classification. Appl. Intell. 2023, 53, 14249–14268. [CrossRef] [PubMed]

Capitolul 6. Concluzii

Procesul de verificare funcțională implică mai multe etape cheie, inclusiv elaborarea planului de verificare, implementarea mediului de verificare, crearea testelor, depanarea, raportarea problemelor descoperite și abordarea erorilor prezente în de regresie. Pe măsură ce proiectarea progresează, aceasta este integrată în mediul de verificare și testată pentru erori.

6.1. Rezultate obținute

Crearea unui testbench UVM de la zero folosind un LLM, cum ar fi ChatGPT sau modele similare, oferă beneficii substanțiale. LLM-urile sunt capabile să înțeleagă instrucțiuni de nivel înalt și să genereze componentele esențiale ale unui mediu UVM, cum ar fi driverele, monitoarele, agenții, secvențiatorii și scorboard-urile. Această automatizare permite inginerilor să se concentreze mai mult pe verificarea designului și mai puțin pe implementarea unor clase de rutină. Abordarea iterativă utilizată — în care LLM oferă un cadru de bază și inginerul îl corectează și îl imbunătățește — demonstrează că IA poate prelua sarcina generării șabloanelor, lăsând logica complexă și deciziile arhitecturale inginerului uman.

În a doua abordare, în care LLM generează o componentă UVM dintr-o fotografie cu o formă de undă dată, capacitatea IA de a recunoaște modele din intrări vizuale și de a le traduce în cod funcțional demonstrează o aplicație avansată ce poate fi utilizată în domeniul verificării funcționale. Această automatizare este deosebit de utilă atunci când se lucrează cu o gamă largă de protocoale sau când specificațiile de evoluează rapid. În loc să rescrie componentele UVM de la zero sau să fie modificate manual, inginerii pot conta pe IA pentru a gestiona eficient aceste modificări, sugestii sau îmbunătățiri. Există mai multe moduri în care LLM-urile pot crește viteza procesului de verificare prin:

- Eficiență și economisire de timp.
- Reducerea erorilor.
- Scalabilitate.
- Flexibilitate și adaptabilitate. Inteligența artificială poate îmbunătăți și mai mult verificarea funcțională prin utilizarea altor strategii, cum ar fi:
- Generarea de stimuli, care se referă la crearea de semnale de intrare sau condiții care determină DUT-ul să-și exercite funcționalitatea.
- IA poate ajuta la înțelegerea specificațiilor de design de nivel înalt și la generarea de teste care validează implementarea în raport cu documentația.

În ceea ce privește rezultatele obținute utilizând LLM pentru a genera aserții SystemVerilog UVM, acestea sunt promițătoare și vor aduce un mare ajutor procesului de verificare. Prin utilizarea aici a două abordări, una bazată pe prompturi text și cealaltă pe imagini, aserțiile generate sunt sintactic corecte și pot fi integrate cu ușurință în mediul de verificare. Inginerul va acționa similar unui integrator sau supervizor, având sarcina de a corecta LLM-ul dacă rezultatele acestuia nu respectă regulile protocolului. Este sarcina inginerului să se asigure că abordările bazate pe

text și imagini sunt corecte și că LLM-ul este solicitat specific pentru ceea ce este necesar.

Prin crearea setului de date de la zero, crearea a două arhitecturi cu rețele neuronale convoluționale, una cu două straturi de convoluție și alta cu trei straturi de convoluție, depanarea, antrenarea și validarea lor, s-a obținut o acuratețe de 98,33% și 96,6% și procente foarte bune de recall, precizie și F1. Această abordare face ca procesul de verificare să fie mai rapid prin clasificarea cerințelor critice din documentație, astfel încât inginerul să le implementeze ca o prioritate ridicată.

6.2. Contribuții originale

În această teză, au fost aduse contribuții variate și complexe în domeniul verificării funcționale, cum ar fi:

• Investigarea, analizarea și cercetarea aplicațiilor posibile ale Inteligenței Artificiale în domeniul verificării funcționale. Diverse strategii și idei au fost analizate, cum ar fi suportul pentru depanare, clasificarea pe bază de text și idei de utilizare a IA pentru a obține acoperire funcțională de 100%.

• Generarea (folosind un LLM), supravegherea, corectarea și depanarea unui testbench UVM APB de la zero prin utilizarea unui prompt bazat pe text. Această sarcină constă în a cere LLM-ului să genereze un testbench complet UVM APB fără a oferi alte informații anterioare despre protocol. Greșelile și codul lipsă au fost corectate cerând LLM-ului să adauge linii suplimentare sau să le șteargă, atunci când a fost cazul. Singurul fișier care nu a fost generat de LLM este fișierul top.sv, deoarece a necesitat în mare măsură contribuția inginerului de verificare.

• Generarea unei componente UVM prin introducerea unei imagini în LLM, fără alte informații. Această abordare este utilă când sunt solicitate modificări, împreună cu un exemplu de formă de undă, deoarece componenta poate fi generată rapid. Rolul inginerului de verificare este de a suoperviza și corecta ieșirea.

• Generarea, ghidarea, integrarea și corectarea aserțiilor pentru un testbench UVM. LLM-ului i s-a cerut, folosind un prompt text, să ajute la generarea de aserțiuni SystemVerilog. În continuare, integrarea și validarea aserțiunilor au fost realizate de inginerul de verificare, evidențiind și mai mult utilitatea LLM-ului în domeniul verificării funcționale.

• Crearea unui set de date de la zero folosind unele protocoalele AMBA. Acest set de date a fost folosit ulterior pentru antrenarea și validarea celor două soluții cu rețele neuronale convoluționale.

• Crearea, implementarea, antrenarea și validarea unui model CNN cu două straturi de convoluție care poate clasifica părți ale textului specificațiilor ca fiind complexe sau nu. Gradul de complexitate se bazează pe experiența inginerului de verificare în domeniu (nouă ani). Modelul are o acuratețe de 98,333340883255%.

• Crearea, implementarea, antrenarea și validarea unui model CNN cu trei straturi de convoluție care poate clasifica părți ale textului specificațiilor ca fiind complexe sau nu. Gradul de complexitate se bazează pe experiența inginerului de verificare în domeniu (nouă ani). Modelul are o acuratețe de 96,66666984558105%.

6.3. Lista publicațiilor originale

- **DRANGA, Diana** and Radu-Daniel BOLCAS. *"Review of Artificial Intelligence enhancements in the field of Functional Verification."* Electrotehnica, Electronica, Automatica (2021), vol 69, no 4., pp.95-102, ISSN 1582-5175, DOI: 10.46904/eea.21.69.4.1108011 **(Scopus, Elsevier – BDI, published December 2021)**.
- BOLCAS Radu – Daniel, **DRANGA Diana**, *"Challenges of facial emotion recognition in machine learning"*, in Electrotehnica, Electronica, Automatica (EEA), 2021, vol 69, no 4., pp.87-94, ISSN 1582-5175, DOI: 10.46904/eea.21.69.1108010 **(Scopus, Elsevier – BDI, published December 2021)**.
- **Diana DRANGA**, "*Trials of using Generative AI for APB UVM testbench generation"*, Romanian Journal of Information Technology and Automatic Control, ISSN 1220-1758, vol. 34(2), pp. 75-84, 2024. https://doi.org/10.33436/v34i2y202406 **(ISI Journal, published June 2024)**.
- **DRANGA, Diana**, and Catalin DUMITRESCU. 2024. *"Artificial Intelligence Application in the Field of Functional Verification"* Electronics 13, no. 12: 2361. https://doi.org/10.3390/electronics13122361 **(MDPI ISI Q2 Electronics Journal, published June 2024)**.
- Valentin Radu, **Diana DRANGA**, Catalin Dumitrescu, Alina Iuliana Tabirca, Maria Cristina Stefan *,"Generative AI Assertions in UVM-based System Verilog Functional Verification"* **(MDPI ISI Q1 Systems Journal, accepted September 2024).**
- Petrica Ciotirnae, Catalin Dumitrescu, Ionut Cosmin Chiva, Augustin Semenescu, Eduard Cristian Popovici, **Diana DRANGA**,*"New method for noise reduction by averaging the filtering results on circular displacements using wavelet transform and local binary pattern*", **(MDPI Electronics Q2, in review).**
- Bogdan Todea, Microchip Technology, Inc., Bucharest, Romania, (Bogdan.Todea@microchip.com), Pravin Wilfred, Microchip Technology, Inc., Bangalore, India (Pravin.Wilfred@microchip.com), Madhukar Mahadevappa, Microchip Technology, Inc., Bangalore, India, (Madhukar.Mahadevappa@microchip.com), **Diana DRANGA**, Microchip Technology, Inc., Bucharest, Romania (Diana.Dranga@microchip.com) *"Break the SoC with Random UVM Instruction Driver"*, presented at **DVCon India, 2019**.

6.4. Perpesctive de dezolvatare ulterioare

Există destul de multe perspective asupra impactului IA în domeniul verificării funcționale folosind LLM:

• LLM-urile ar putea fi utilizate în continuare pentru a automatiza generarea unor modelelor care vor ajuta la atingerea acoperirii testelor la 100%.

• Asistență automată pentru depanare: extinderea capacităților LLM-urilor pentru a asista la depanare ar putea oferi inginerilor sugestii automate pentru rezolvarea erorilor în mediul UVM.

• Suport îmbunătățit pentru Protocoale: LLM-urile ar putea fi antrenate pentru a gestiona o gamă mai largă de protocoale, integrând cunoștințe profunde ale standardelor specifice din industrie, cum ar fi PCIe, USB și Ethernet, pentru a facilita verificarea mai rapidă în SoC-uri multi-protocol.

• NLP în SystemVerilog UVM: îmbunătățirea în continuare a capacității LLM-urilor de a traduce descrierile în limbaj natural ale cerințelor de verificare direct în componente UVM ar putea face procesul de verificare mai accesibil pentru cei fără expertiză extinsă în codare.

Capitolul 7. Bibliografie

[1] - DRANGA, Diana and Radu-Daniel BOLCAS. *"Review of Artificial Intelligence enhancements in the field of Functional Verification."* Electrotehnica, Electronica, Automatica (2021), vol 69, no 4., pp.95-102, ISSN 1582-5175, DOI: 10.46904/eea.21.69.4.1108011 (Scopus, Elsevier - BDI).

[2] - Diana DRANGA, *"Trials of using Generative AI for APB UVM testbench generation"*, Romanian Journal of Information Technology and Automatic Control, ISSN 1220-1758, vol. 34(2), pp. 75-84, 2024. https://doi.org/10.33436/v34i2y202406 (ISI Journal, published June 2024).

[3] - Valentin Radu, Diana Dranga, Catalin Dumitrescu, Alina Iuliana Tabirca, Maria Cristina Stefan, *"Generative AI Assertions in UVM-based System Verilog Functional Verification"* (accepted September 202**4,** MDPI Systems Q1)

[4] - DRANGA, Diana, and Catalin DUMITRSCU. 2024. *"Artificial Intelligence Application in the Field of Functional Verification"* Electronics 13, no. 12: 2361. https://doi.org/10.3390/electronics13122361 (ISI Q2 Journal, published June 2024).