National University of Science and Technology POLITEHNICA Bucharest

Doctoral School of Electronics, Telecommunications and Information Technology

Decision No. 234 from 02-10-2024

Ph.D. THESIS SUMMARY

Diana DRANGA

Contributions to functional verification using Artificial Intelligence

---

THESIS COMMITTEE

Prof. Dr. Ing. Mihai CIUC National University of Science and Technology POLITEHNICA Bucharest	President
Conf. dr. ing. Habil. Cătălin DUMITRESCU National University of Science and Technology POLITEHNICA Bucharest	PhD Supervisor
Prof. Dr. Ing. Cătălin CĂLEANU Universitatea Politehnică din Timișoara	Referee
Prof. Dr. Ing. Ștefan TOMA Academia Tehnica Militară	Referee
Conf.Dr.ing. Eduard POPOVICI National University of Science and Technology POLITEHNICA Bucharest	Referee

BUCHAREST 2024

Content

Chapter 1. Introduction into functional verification and thesis goal

1.1 Presentation of the field of the doctoral thesis

Functional verification is a critical process in the research and development of a System-on-Chip (SoC). Its main scope consists of making sure that the chip behaves accordingly, within the descriptions of the documentation provided.

In many cases the verification environment is composed of different instances, written in System Verilog using the Universal Verification Methodology (UVM). It comprises of instances such as:

- uvm_driver, a class which receives UVM sequence items transactions from the UVM Sequencer and drives it to the DUT Interfaces. It converts transaction level stimulus into pin level stimulus [1].
- uvm_monitor, instance which samples the DUT and picks up the information from the transactions, while also sending it out to other components using a TLM analysis port [1].
- uvm_sequencer, it acts more as an arbiter used in controlling transaction flow from multiple sequences [1].
- uvm_agent, an encapsulation of the classes presented above [1].
- uvm_scoreboard, instance which has the main function of checking the behaviour of the DUT. It usually receives transactions from the inputs and output of the DUT through UVM agent analysis ports, runs these inputs against a reference model and compares the expected output vs the actual output [1].

In order to correctly use these classes, there is an UVM Class Library which provides all the support to develop in a swiftly manner robust testbenches and reusable, verification components. In library, there are also utilities and macros.

The UVM Class Library provides even more utilities to further simplify and aid the development of verification environments [1].

1.2 References

[1] Accelera, UVM Guide

1.3 Scope of the doctoral thesis

Artificial Intelligence may play a key role to functional verification of SoCs. Functional verification is a time-consuming process in which multiple components of the chip are tested. Due to time consuming nature of the process, it is essential to implement various strategies of minimizing time using Artificial Intelligence.

Using various techniques, such as LLMs, CNNs and other technologies, the verification step will be enhanced and offered a speed up. This speed up will provide an early error catching mechanism. This will benefit the industry vastly, since now SoC development is prolonged.

Using ChatGPT and CNN models are great starting points for research into diminishing the time spent on functional verification.

1.4 Content of the doctoral thesis

In the first chapter of the thesis there is a summary of the verification field, what is usually used in the industry, specific class constructs, methods, and tasks.

In the second chapter, the Artificial Intelligence field will be introduced into the thesis, explain the broad concepts in order to understand thoroughly.

In the third chapter, further works using Generative AI will be introduced.

In the fourth chapter, a CNN is present, as it has been developed by the engineer, with the goal of minimizing the time spent on analysing large documentations which are usually seen in the industry.

The fifth chapter represents a conclusion of the thesis.

Chapter 2. Artificial intelligence in general

Artificial Intelligence (AI) is flourishing field, sparking more and more interest from a wide range of domains due to its practical applications [1].

A primal example of a Deep Learning model is the feedforward network, commonly referred to as a Multi-Layer Perceptron (MLP). A Multi-Layer Perceptron is essentially a mathematical function that maps a set of input values to corresponding output values, achieved by combining numerous simpler functions.

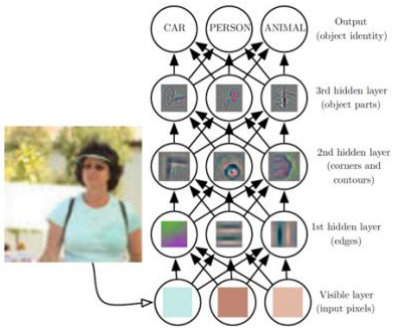


Figure 2.1. Example of a Deep Learning model [1]

Deep learning addresses this challenge by breaking down the complex mapping into a series of nested, simpler mappings, each handled by a different layer in the model as it can be seen in Figure 2.1.

Machine learning falls under the category of AI, a technique that enables computers systems to improve over time through experience and data. Deep learning, a subset of machine learning, gains its power and flexibility by learning to represent the world as a nested hierarchy of concepts [1].

2.1 Machine Learning

2.1.1. Machine Learning Algorithm

Machine learning is enabling researchers and engineers to address tasks that are too complex to solve using fixed programs designed by humans. An example consists of a set of features quantitatively measured from an object or an event that the machine learning process needs to process. For example, in the case of an image, the features might be the different values of pixels of the image [1].

Some of the more common tasks used in machine learning are:

- Classification. Modern object recognition has been significantly enhanced through the use of Deep Learning technique [2][3]. This foundational technology also powers facial recognition systems [4] enabling automatic tagging in individuals in photo collections [1].
- Regression.
- Anomaly decision.
- Denoising.

Unsupervised learning algorithms work with a dataset that contains many features and learn valuable characteristics about the structure of the data. In deep learning the goal is to learn the probability distribution that generated the dataset.

Supervised learning algorithms work with a dataset that includes features, where each example is accompanied by a label or a target.

Many machine learning problems become more increasingly challenging when the data has a high number of dimensions. This issue is known as the problem of dimensionality.

2.1.2. NLP

Natural Language Processing (NLP) refers to the use of human languages, by a computer. Unlike specialized programming algorithms designed for efficient and clear parsing by computers, natural languages are ambiguous and resist formalisation. NLP contains applications such as machine translations, where a system must read a sentence and in one human language and generate an equivalent sentence in another.

2.1.3. Optimization algorithms

Most deep learning algorithms involve some sort of optimization. Optimization refers to the task of minimizing or maximizing a function $f(x)$ by adjusting the variable x.

Optimization algorithms may struggle to find a global minimum when multiple local minimums or plateaus exist. In deep learning, it is generally acceptable to settle for these solutions, even if they are not absolute minimum if they result in sufficiently low value of the cost function [1].

2.1.4. Algorithms with adaptive learning rates

Neural network researchers have long recognized that the learning rate is one of the most challenging hyperparameters to configure, as it significantly affects model performance.

The AdaGrad algorithm adapts the learning rates of individual model parameters by scaling them inversely to the square root of the sum of their historical squared gradients [12].

Adam [6] is another adaptive learning rate optimization algorithm. The name "Adam" stands for "adaptive moments".

2.2 References

[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 Possible uses of Artificial Intelligence in functional verification [1]

This chapter is adapted after the author's article named "Review of Artificial Intelligence enhancements in the field of Functional Verification".

2.3.1. Introduction

In today's world there is a necessity of documenting large amounts of data in in an attempt to provide better services or products to customers.

In order to at least reduce the time spent on verification, artificial intelligence might be used in order to extract or inject, depending on the application, crucial debugging information messages in the verification environment or aid the engineers in reaching maximum coverage.

2.3.2. Existing time-consuming issues

In most cases the verification environment will need an engineer to debug because of multiple issues, thus increasing the time spent on it.

Furthermore, a large chunk of time is also spent on trying to reach maximum coverage. For this, initially a verification plan is designed, which contains the coverage bins, checkers' mappings and tests which are to be written and exercised.

These are two cases where Artificial Intelligence might play a big role into speeding up the process, one of them being debugging and the other reaching 100% coverage. In this paper both cases will be reviewed.

2.3.3 Possible enhancements of Functional Verification with Artificial Intelligence

Deep learning is a subset of Artificial Intelligence, which maps the "input"

features (analogues to prediction variables in traditional statistics) to an output.

There are multiple types of neuronal networks:

1. Feedforward neural network.
2. Recurrent neural network.
3. Convolutional neural networks with multiple layers, including a convolutional layer, non-linearity layer, polling layer and fully connected layer.

2.3.4. Study of debug support using Artificial Intelligence

Text classification is a solution to be taken into consideration when debugging complex verification environments in order to see which language was used for the specific file [1].

Reyes et al. [2] managed to extract the features from a preprocessed source code and trained the RNN (Recurrent Neural Network), by using the order for the text sequences. The researchers obtained these results in 2016, on ten different programming languages, 756 MB of files using the text classification technique [1]. Dam et al. used Natural Language Programming (NLP) [3], specifically the n -grams and skip grams methods to be able to classify the source code on twenty programming languages, 14.000 files utilizing text code classification [1]. Baquero et al. [4] came forward with a model which could predict the programming language from comments text data and code snippets [1]. The study had been conducted on eighteen programming languages, 18.000 files using support vector machine (SVM) method.

2.3.5. Text – based source code classification approach

The pre-processing step is a must for text data. A transformation from text elements to numeric features needs to be performed to be used as inputs for the algorithm. A tokenization method is used in this step, in order to split the text into sperate words and/or tokens. Embedding is the process of creating vectors which contain real numbers that represent the tokens created in the previous step. Each word or element is mapped into a vector or vectors which are learned by the Convolutional Neural Network during its training phase [1].

2.3.6. Image – based source code classification approach

Among other advantages of the image-based approach is the ability to interpret comments of the source code provided. While in the text–based approach, the comments added in the code must be removed in order to interpret correctly the file, here the Convolutional Neural Network will not have this problem. Furthermore, the comments added may be useful information discarded by the text-based approach.

2.3.7. Study of coverage support using Artificial Intelligence

As mentioned in this study, functional coverage is labour-intensive for the engineer, creating a bottleneck in the verification phase. Multiple studies have explored the possibility of having a neural network enhancing the time spent on coverage.

In simulation-based verification, the stimuli that are to be driven to the DUT are generated random-constrained in the test. A quick improvement that can be applied in order to rapidly increase coverage is the integration of CDG (Coverage-directed Test Generation) [5].

Wang et al. [5] have reconsidered the issue of the complex ASICs, which contain complicated RTL (Register Transfer Layer) blocks, with a tremendous number of parameters, configurations and prone to having many faults. Their aim was to develop a verification system that can discern between the parameters and configurations which are relevant, and which are not.

Q.Guo et al. [6] have built a framework that intends to reduce the irrelevant stimuli on the fly. If the stimuli are categorized as being redundant, then it will not be sent to the Design Under Test. Therefore, only the stimuli that are relevant will be sent to the DUT [6].

2.3.8. CUDA enhancement of Artificial Intelligence

The previous subsections exposed a handful of solutions to the existing problems in the field of functional verification. These following subsections prospects to lessen further the time spent by using AI and adopting the Computer Unified Device Architecture (CUDA) platform by Nvidia.

2.3.9. Comparison of CUDA vs OpenCL

Open Computing Language (OpenCL) is a free source standard for multipurpose parallel programming from CPUs, GPUs and multiple processors.

OpenCL has many similarities when compared with CUDA. First the thread architecture of CUDA is almost identical to the one in OpenCL (work-item). Secondly, in CUDA a grid is referring to a set of all the threads executing the same kernel function. The grid contains arrays of blocks which should be the same size. This is very similar with OpenCL definition of workgroup.

Some differences between CUDA and OpenCL are related to memory models. In the case of CUDA, the local memory can be seen by each thread.

Regarding compilation, CUDA uses a static compiler which is responsible of compiling the kernel and host code before throwing it to the GPU. On the other side, OpenCL uses ahead-of-time (AOT) or a just-in-time (JIT) compiler which has good portability. Because of the AOT, OpenCL needs additional initialisation time in order to look devices [7].

2.3.10. Conclusions

This chapter presents the context for the verification of a chip and the challenges of reducing the number of bugs to zero. This process is tedious and is taking a large amount of time. Artificial Intelligence was reviewed as a solution to reduce this blockage by attaining the aimed coverage percentage defined in the test plan.

2.3.11. Bibliographic References

[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 Founding sources

Funding sources for this paper were supported by the authors with the logistical support of the Polytechnic University of Bucharest, Faculty of Electronics, Telecommunication and Information Technology.

Chapter 3. Large Language Models and applications into functional verification [2]

This chapter is adapted after the author's article named "Trials of using Generative AI for APB UVM testbench generation".

The functional verification process consists of multiple parts such drafting the verification plan, environment implementation, tests implementation, debugging, reporting the issues found and fixing the regression failures. As design increases in development, it is integrated into the verification environment, thus tested for bugs. The manner in which functional verification is done is presented in Figure 3.1 (Mammo, 2017).

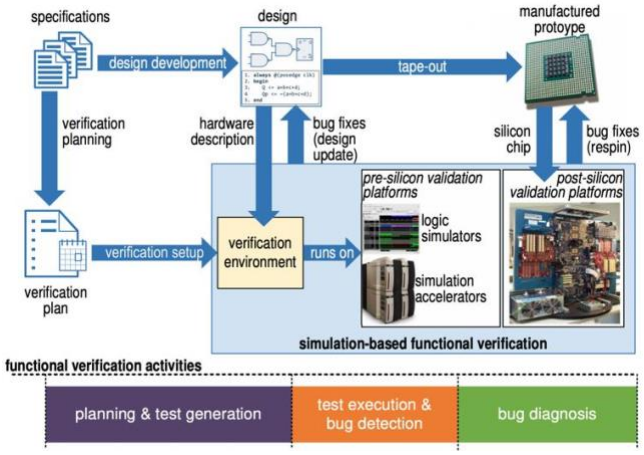


Figure 3.1. Verification process steps (Mammo, 2017)

3.1. AI in functional verification and other research

Artificial intelligence can greatly improve the verification process in multiple ways:

- Coverage completion. In this paper (Dinu et al., 2022), genetic algorithm approaches are used to generate stimuli which are sent to the input port of the Device under Test (DUT).
- Regression clustering and filtering. Various machine learning algorithms can be used to filter different regression failures (Dinu & Ogrutan, 2019) and to focus solely on more important failures;

- In documentation, pattern recognition on different images and text. By using these, the project verification task can be defined better and more efficient;
- Generating various testbenches using Generative AI. This approach may be used to generate full System Verilog UVM testbenches from scratch using ChatGPT4, in order for the engineer just to analyse the code provided and integrate the resulted verification environment into the respective system.

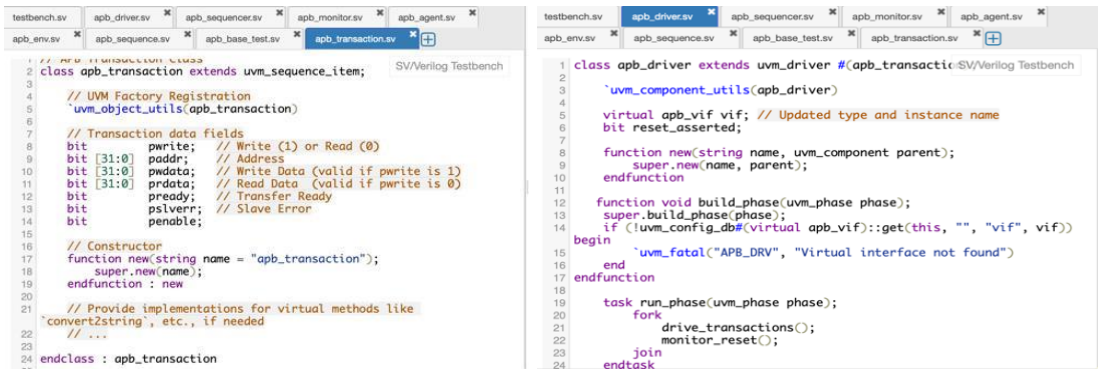
3.2. State-of-the-Art

Large Language Model (LLM) implementations such as ChatGPT have become viable options for engineers in order to navigate or generate large, complex, feature rich code (Khurana et al., 2024).

In this chapter, the approach using GenerativeAI to generate a System Verilog UVM ARM AMBA (Advanced Microcontroller Bus Architecture) APB (Advanced Peripheral Bus) testbench from scratch was used, among a short study regarding System Verilog UVM code generation using a waveform diagram usually found in official documentation.

3.3. UVM Testbench generation from scratch using ChatGPT4

In this chapter most of the UVM components were implemented, aside from the scoreboard. These components have been generated entirely using ChatGPT4. The only part which had to be manually adjusted due to ChatGPT4 limitations is the top.sv file where the signals are connected to the DUT and the virtual interface. The code was analysed and corrected by the verification engineer. The code was simulated using the open source site EDA Playground which provides limited but enough simulation capabilities with all three vendors tools (Cadence INCISIV, Questa, VCS). Below are the snippets of the code generated by ChatGPT4.

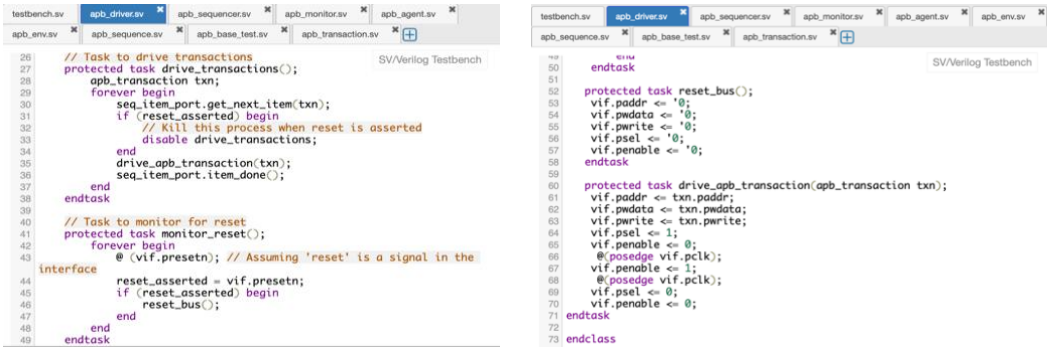


A) **B)**

Figure 3.2 Item and driver classes generated by ChatGPT4

ChatGPT4 was requested to generate an item or transaction class using UVM System Verilog which respects the AMBA APB protocol. In the first inquiry of ChatGPT4 the LLM output was only the signals pwrite, paddr, pdata, prdata and pready, which is incomplete as the pslverr and the penable signals are needed. On the second run, as requested, the model added the two missing signals in the

uvm_sequence_item. For the apb_driver class, the model was asked in the same manner to provide the code for specifying clearly to use the UVM Methodology and SystemVerilog as the verification language. The output was also incomplete as the reset functionality of the protocol, which is indispensable, was missing. The missing parts of code were added by the verification engineer. Figure 3.2 A), B) and Figure 3.3 A), B) both illustrate the transaction and driver classes resulted by using the LLM and correcting the missing items.



A) **B)**

Figure 3.3. UVM Driver class

For completing this APB UVM testbench a sequence was requested by the verification engineer to the model in order to be able to use this sequence in a test.

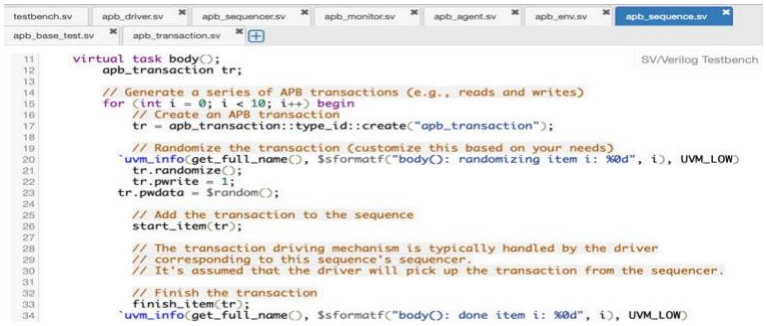


Figure 3.4. UVM Sequence

The sequence in Figure 3.4 was used in a UVM APB test generated by a LLM. After running the simulation, the waves were dumped in order to analyse it and ensure the correctitude of the testbench. **Figure 3.5** shows the waveforms.

Figure 3.5. Results of the ChatGPT generated UVM testbench

The only file that was needed to be created by hand by the engineer was the top.sv file. ChatGPT4 can replace or correct names quite easily, but as needed in a top.sv file it cannot connect the interface to the virtual interface, uses wrong and uncompilable code for uvm_set_config_db() and also for the run_test() method.

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

In this study, a protocol image was fed to the model and asked to provide the respective UVM Driver. This approach is useful when adding an individual UVM component in an already existing testbench. This way, the code is already generated using either an image, diagram or maybe waveforms and it will be imported into the existing testbench. This approach saves valuable time, instead of writing the UVM component from scratch. Figure 3.6 contains the image used and Figure 3.7 A), B) the resulted code.

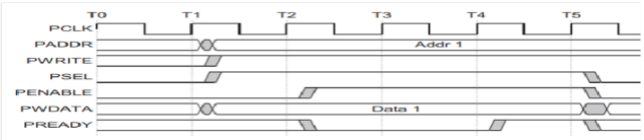


Figure 3.6. APB Write transaction waveforms (ARM, 2023)

```
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
```

```
    // 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
```

A) **B)**

Figure 3.7. Generated classes

3.5. Conclusions

In this chapter, two approaches were presented to decrease the time spend on functional verification. The first approach, where an UVM APB testbench is developed from scratch, might be more suitable when wanting to have more control of what the LLM is asked to output. By using a text-based approach in which the LLM is asked specifically for the classes involved, the methodology used and the programming language, the verification engineer has more control and less time spent

correcting ChatGPT4. The second approach, using a protocol picture (in this case an ARM APB Write transfer) which is present in many documentations, can prove very useful when change requests are needed.

This study aimed to prove, on a small testbench, the benefits and possible ChatGPT4 use cases for the functional verification process. As presented above, two approaches were used by the verification engineer, both with very good results and decreasing time spent on the project."

3.6. References

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].

Chapter 4. Generating System Verilog assertions using a Large Language Model [3]

This chapter is adapted after the author's article named "Generative AI Assertions in UVM-based System Verilog Functional Verification".

4.1. Introduction

Assertion generation for various protocols such as Advanced Peripheral Bus, AHB (Advanced High-Performance Bus), etc using a LLM such as ChatGPT can vastly reduce the functional verification time. These assertions might be also integrated in other verification environments; thus, it is useful to write the assertion code in a good and reusable way.

4.2. Materials and Methods

In this study, ChatGPT has been asked using the prompt to generate System Verilog assertions for an APB UVM testbench. Figure 4.1 presents the way the LLM was asked to provide the results.

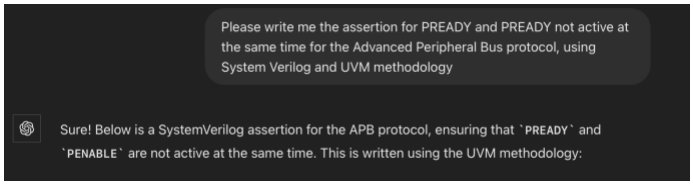


Figure 4.1. ChatGPT text prompt

There is another way of using the LLM, such as providing a protocol waveform in order to let ChatGPT interpret the waveform rules and provide the needed assertions. Figure 4.2 presents the Write transfer transaction waveforms of the APB protocol being fed into the LLM and just asking ChatGPT to output the assertions without writing any definition of the protocol or rules. The LLM's task was to interpret the waveform and generate the assertions according to the rules implied by the waveform.

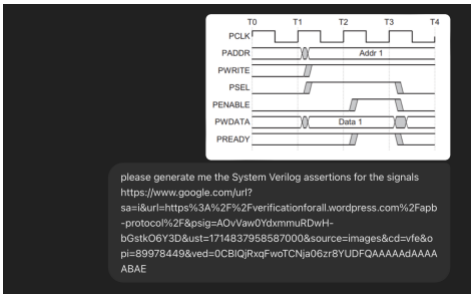


Figure 4.2. ChatGPT prompt fed with APB Write transfer waveform

Using these two approaches, this chapter aims to underline how swiftly the LLM can generate critical code, vastly reducing the time spent on verification. The verification environment, and Register Transfer Layer (RTL) implementation were developed by the researchers.

4.3. Results

Using the first approach the LLM was asked via text to provide the assertions for different signals of the APB protocol. The output of the LLM for an assertion that has the condition of PENABLE signal not active during reset is described in Figure 4.3.

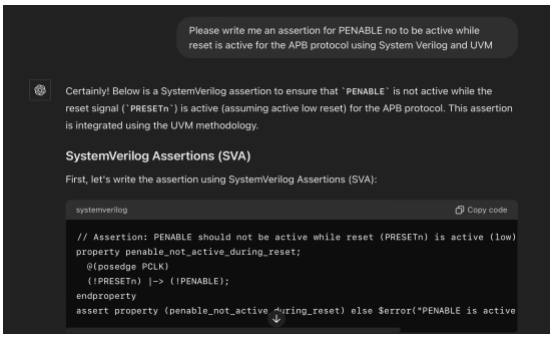


Figure 4.3. ChatGPT output for text prompt

The code of the LLM has been introduced in the top.sv file of the testbench where the top module resides for easier integration.

```
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. Code output by the LLM and integrated in the top module

A test was run with the lines 66-72 active (Figure 4.4). The test failed, because the ChatGPT – 4 generated assertion was invalidated. In Figure 4.5, the failure log of the respective test is presented. It can be seen clearly that the ChatGPT – 4 assertion has correctly indicated the error in the 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. Test assertion failure log

When asking the LLM for basic assertions for the AMBA APB protocol via text prompt, the LLM output is presented in Figure 4.6. The figure presents the LLM

providing the assertions with explanations on each point and carefully structuring the code.

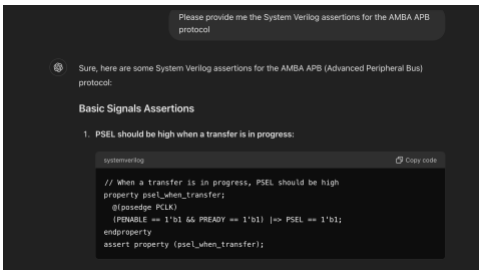


Figure 4.6. Text Prompt for AMBA APB basic assertions

These assertions provided by the LLM were integrated in the APB verification environment. This is shown in Figure 4.7. Figure 4.7 presents all the basic AMBA APB System Verilog assertions integrated in the top.sv file. The LLM suggested five basic assertions such as: PSEL should be high when a transfer is in progress, PENABLE should be asserted only after PSEL is asserted, PADDR should be stable when PSEL and PENABLE are high, PWRITE should be stable PSEL and PENABLE are high, PWDATA should be stable during the enable phase of the transfer.

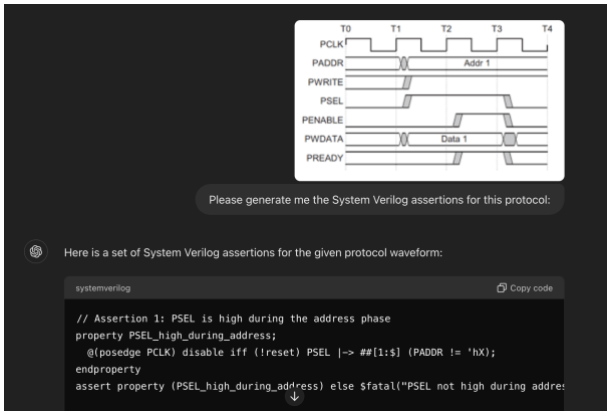
```
// Assertion: PENABLE should not be active while reset (PRESETn) is active (low)
property penable_not_active_during_reset;
  @(posedge clk)
  (!rst) |-> (!vif.penable);
endproperty
assert property (penable_not_active_during_reset) else $error("PENABLE is active while reset is asserted");
// When 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. Assertion integration in the testbench

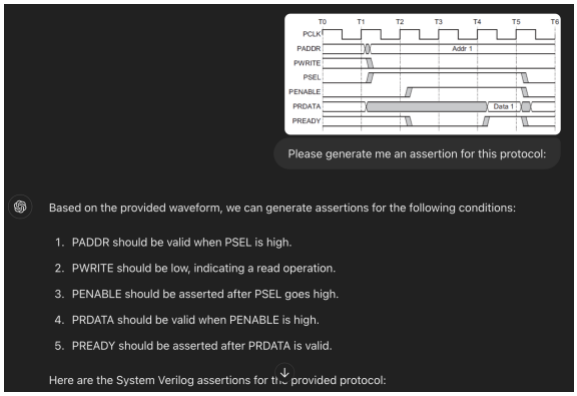
In order to be able to resolve the simulation issues, the researchers have renamed the signals from PCLK, PENABLE, etc. to clk, vif.penable. This was done in order to correct the variables as to point to the APB virtual interface of the testbench. The test was run, with errors triggered by the assertions suggested by ChatGPT – 4.

In the second approach, the LLM was fed an AMBA APB Write transfer image and also an AMBA APB Read transfer image. No further details were added,

just the images themselves and the request to generate System Verilog assertions. Figure 4.8 A) and B) exhibit the way the LLM was asked to provide the code and the images fed into it.



A). Image prompt for an APB Write Transfer



B). Image prompt for an APB Read Transfer

Figure 4.8. Image prompt for an APB Read & Write Transfer

For the write transfer ChatGPT – 4 generated System Verilog assertions which check the functionality of PADDR, PWRITE, PENABLE, PWDATA and PREADY. ChatGPT4 has correctly generated the code for the assertions, although a few modifications are sometimes needed, mostly for the integration in the existing testbench.

While these two approaches generate fast and quite accurate results, there is the need for the verification engineer to supervise the model results in order to correct them if it the case. Thus, here the LLM servers more as co-pilot.

4.4. Conclusions

In this chapter the researchers have proven the impact of using ChatGPT – 4 for generating System Verilog assertions and integrated them into an existing APB UVM testbench. By generating assertions, time spent on functional verification was

reduced to about 20% for the testbench presented in this study (APB UVM testbench), by not implementing a dedicated score boarding class and relying only on assertions. This environment is run on a free platform, which makes it harder to develop than in an already dedicated simulation environment. Taking into consideration these finding, using Generative AI to create assertions makes it a great candidate as to greatly reduce time spent on functional verification.

Chapter 5. AI used in functional verification [4]

This chapter is adapted after the author's article named "Artificial Intelligence Application in the Field of Functional Verification".

5.1. Introduction

As Integrated Circuit development progresses in complexity and scalability, it becomes essential for them to work as expected, being reliable and robust, especially in critical files such as medical, aviation, and automotive files.

By using Artificial Intelligence, the verification process can be faster and completed in less time.

5.2. Related Work

There are a multitude of areas where Artificial Intelligence can aid the verification process. The tests are grouped according to the failure reasons. As stated in [1], some machine learning algorithms can effectively classify the failing tests in a regression according to their failure reasons. Although, according to [2], the outcome is not as expected. Stimulus and test generation is conducted by using supervised and reinforcement ML algorithms [3] to hit the planned coverage for the Device under Test (DUT) [4]. In the verification process of a cache controller, a supervised Deep Neural Network (DNN) [5] was used alongside the Q learning algorithm. The DNN is trained to create sequences for four First-in-First-Out (FIFO)s.

5.3. Methodologies, Study Materials, and Databases

In this study, the researchers implemented the dataset alongside the model. The dataset consists of important or critical information from documentation and non-crucial information. Different specifications and documentations were analysed and combined to create the dataset. Such documentations include, for instance, ARM's Advanced Microcontroller Bus Architecture APB and AXI. The specifications have been thoroughly analysed by the engineer and the foremost information and irrelevant data were categorized into relevant and irrelevant information.

Figure 5.1 is an example of other relevant data that must be considered by the engineer. It contains a description of the PSLVERR, PREADY, and PENABLE signals. These are signals usually used in an APB transfer.

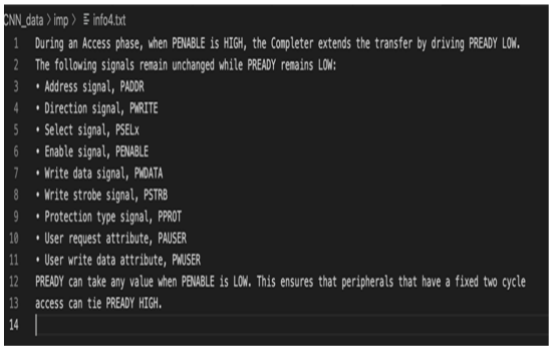


Figure 5.1. ***APB signal short description.***

These are some examples of relevant data in the created database. As for irrelevant data, these can contain parts such as copyright rules that, to the engineer, are not relevant when implementing the verification environment, and the test scenarios and parts may not be implemented in the RTL design. Due to various reasons, some parts of different modules, protocols, etc., may not be implemented in the design. Figure 5.2– Figure 5.4 illustrate an example of irrelevant information mentioned above.

Figure 5.2. Succinct APB list of protocol rules

Figure 5.3. AMBA AXI Copyright information

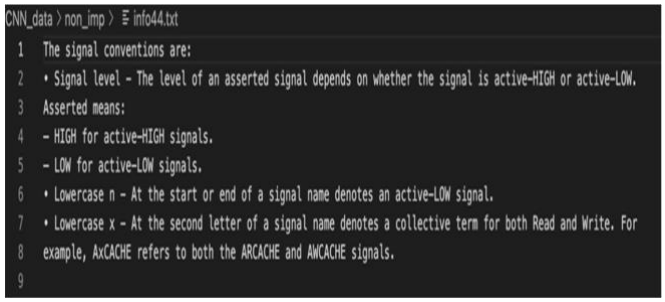


Figure 5.4. AMBA AXI signal conventions

Considering the means above, a database was constructed from scratch using the verification engineer's expertise and experience in the field. For this dataset, information was carefully extracted from the Advanced Microcontroller Bus Architecture Advanced Peripheral Bus (AMBA APB) and from the Advanced Microcontroller Bus Architecture Extensible Interface (AMBA AXI). This dataset will be used to train the model presented in this paper.

5.4. Results

In this presented application, data preprocessing was performed before feeding it into the model.

Data were augmented using special libraries to enhance the database rapidly. For this, the nlpaug library was used together with a synonym method, which was applied to the dataset. The labels were encoded using LabelEncoder, and data were then Tokenized. Test and training data were split into 80% training and 20% tests. Two CNN sequential model implementations were tested. The architecture of the first CNN model used is described in Figure 5.5. The sequential model is a deep learning model that is made from linear stack layers. Layers are added to the model in sequential order, and the output of each layer is the input of the next layer. The first proposed model consists of an Embedding Layer, a convolution layer, an activation Flatten layer, and one dense layer. For the activation layer, the Rectified Linear Unit (ReLU) activation function was used.

Figure 5.5. First CNN architecture

This first proposed model ran 20 epochs, having at the end a test accuracy of 98.333340883255% and a loss of 10.25% with data augmentation applied one time and concatenated with the created dataset. This first CNN model is a try-out to see what needs to be done next to improve the accuracy and loss. There are a couple of options, such as augmenting the data even more, adding more layers, or hyper parametrizing the model. Figure 5.6 describes the training and validation accuracy graph as well as the training and validation loss graph.

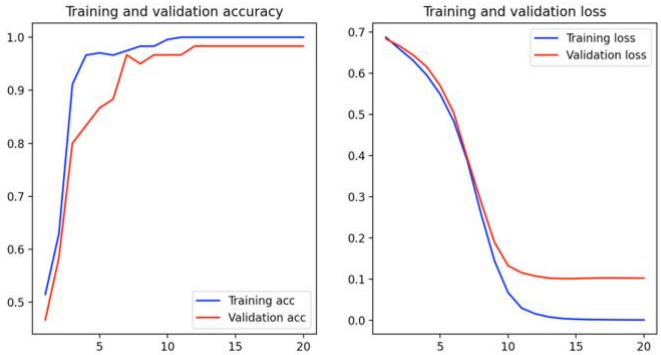


Figure 5.6. First CNN model graph for training and validation.

The model presents high percentages of precision—97%, recall—100%, F1-score—98% for 0, and for 1, precision—100%, recall—96%, F1-score—98%. These results make the model reliable for both cases, with high precision, recall, and F1-score, performing very well in identifying relevant instances and making accurate predictions, with a good balance between precision and recall (achieving high precision and high recall at the same time).

Another sequential model was developed to try and achieve better results. An architecture was defined using three convolutional layers, three activation layers, an embedded layer, one MaxPooling layer, one Flatten layer, and one dense layer, as presented in Figure 5.13.

Figure 5.7

Figure 5.7. Second CNN model architecture.

After applying data augmentation, we split the dataset into 80% training and 20% validation and trained the model with an accuracy of 96.66666984558105% and a validation loss of 5%.

This model architecture uses three convolutional layers, has very high precision scores for both 1 and 0 (100% and 93%), has a high recall percentage (94% and 100%), and has a good F1-score (97%).

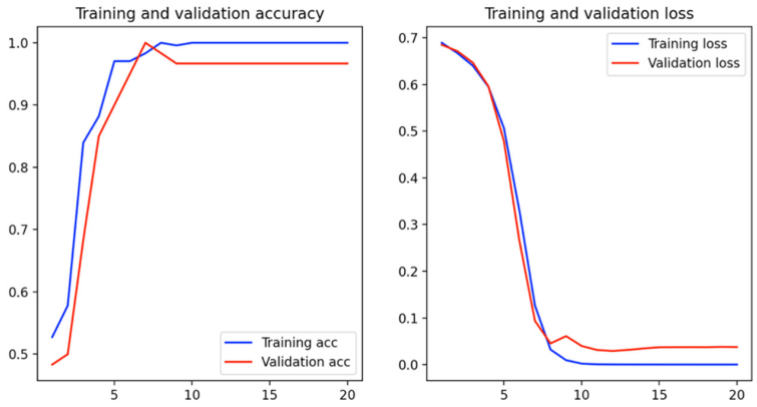


Figure 5.8. Second CNN model with three convolutional layers training and validation

In this paper [6], a multitude of models were used to classify the binary data for two datasets: DATASET-1 (similar to the Stanford Sentiment Treebank with only

negative and positive reviews) and DATASET-2 (Amazon review of Sentiment Analysis Dataset). The best models applied on those datasets, TextConvoNet_4 (four convolutional layers) and TextConvoNet_6 (six convolutional layers), achieved accuracies of 82.2% and 81.9%, respectively, on DATASET-1. For DATASET-2, TextConvoNet_4 has an accuracy of 90.4%, while TextConvoNet_6 has an accuracy of 87.2%. If these results are taken into account, the proposed two CNN architectures would outperform the TextConvoNet_4 and TextConvoNet_6 models. The accuracy that the Bidirectional Encoder Representations from Transformers (BERT), Hierarchical Attention Networks (HAN), and BerConvoNet models have achieved are 77.6%, 80.2%, and 83.1% on DATASET-1. For DATASET-2, the same models have reached 77.2% (Bidirectional Encoder Representations from Transformers-BERT), 86.3% (HAN), and 88.3% (BerConvoNet). Therefore, the models presented in this study are performing better than the ones used in [6].

5.5. Conclusions

In this paper, two AI implementations were used in order to aid the verification process in finishing in less time. The two approaches consisted of using two Convolutional Neural Network architectures, one with two convolutional layers and one with three convolutional layers. In both cases, data augmentation was performed. This means that, on the original dataset, a method was applied to substitute some words with their respective English synonyms to enhance the dataset further without adding any entries. The dataset was created, developed, and analysed by the researchers, using their experience in the functional verification field. Both Convolutional Neural Network models were developed using Python 3.12.3 and ran using an Apple M1 hardware on a dedicated 10-core GPU with an Anaconda environment.

For the first Convolutional Neural Network, a sequential model was defined, and layers were added to it. This model implementation generated fine results, high test accuracy (98333340883255%), high recall, precision and F1-score, and low validation loss (10.25%), making it a reliable, robust, and well-adjusted model.

The second architecture generated high accuracy at 96.66666984558105% and low validation at 5%. The scores for precision, recall, and F1-score are also good overall, making the model trustworthy and solid.

5.6. References

[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]

Chapter 6. Conclusions

The functional verification process involves several key stages, including drafting the verification plan, implementing the verification environment, creating tests, debugging, reporting discovered issues, and addressing regression failures. As the design progresses, it is integrated into the verification environment and tested for bugs.

6.1. Obtained results

Creating an UVM testbench from scratch using an LLM, like ChatGPT or similar models, provides substantial benefits. LLMs are capable of understanding high-level instructions and generating the essential components of a UVM environment, such as drivers, monitors, agents, sequencers, and scoreboards. This automation allows engineers to focus more on refining the design and less on routine coding tasks. The iterative approach used—where the LLM provides a basic framework and the engineer refines it—demonstrates that AI can handle the heavy lifting of template generation, leaving complex logic and architectural decisions to the human expert.

In the second approach, where the LLM generates a UVM component from a given waveform, the AI's ability to recognize patterns from visual inputs and translate them into functional code showcases an advanced application of machine learning.

This automation is particularly useful when dealing with a wide range of protocols or when design specifications evolve rapidly. Instead of rewriting UVM components from scratch or modifying existing ones manually, engineers can rely on AI to handle those changes efficiently. There are a number of ways the LLMs can further increase the speed of the verification process by:

- Efficiency and Time Saving.
- Error Reduction.
- Scalability.
- Flexibility and Adaptability.

Artificial Intelligence can further enhance functional verification by using other strategies such as:

- Stimulus generation refers to the creation of input signals or conditions that drive the DUT to exercise its functionality.
- AI can assist in understanding high-level design specifications and generate tests that validate the implementation against the documentation.

Regarding the results obtained using the LLM to generate System Verilog UVM assertions, they are promising and will provide great aid to the verification process. By using two approaches, one text-based prompt and the other image-based approach the assertions generated are syntactically correct and can be easily integrated into the verification environment. The engineer will act similar to a supervisor having to correct the LLM if the output is not respecting the protocol rules. It is the task of the engineer to make sure that the text-based and image-based approach are correct and the LLM is asked specifically for what it is needed.

By creating the dataset from scratch, architecting two Convolutional Neural Networks, one with two convolutional layers and one with three convolutional layers, debugging, training, and validating them an accuracy of 98.33% and 96.6% and very good recall, precision and F1 score. This approach makes the verification process faster by classifying the critical requirements in the documentation as for the engineer to implement those as a high priority.

6.2. Original contributions

In this thesis, various and complex contributions have been added to the field of functional verification such as:

- Investigation, analysation, and research of possible Artificial Intelligence applications in the field of functional verification. Various strategies, ideas were analysed such as debugging support, text-based classification, and AI ideas in order to achieve 100% functional coverage.
- Generating (using a LLM), supervising, correcting and debugging an UVM APB testbench from scratch by using a text based prompt. This task consists of asking the LLM to generate a full APB UVM testbench without providing any other prior information about the protocol. Mistakes and missing code were corrected by asking the LLM to add additional lines or delete when it was the case. The only file not generated by the LLM is the top.sv file, as it needed the verification engineer's input vastly.
- Generating an UVM Component by feeding an image into the LLM, without any other information. This approach is useful when change requests are in place, alongside with waveform example as the component can be rapidly generated. The verification engineer's role is to supervise and correct the output.
- Generating, guiding, integrating, and correcting assertions for an UVM testbench. The LLM was asked using text prompt to aid in generating System Verilog Assertions. Further, the integration and validation of the assertions was done by the verification engineer, thus underlining even more the usefulness of the LLM in the functional verification field.
- Creating a dataset from scratch using some of the AMBA protocols. This dataset was lately used to train and validate the two Convolutional Neural Network solutions.
- Creating, implementing, training, and validating a CNN model with two convolutional layers which can classify parts of specification text as complex or not complex. The degree of complexity is based on the verification engineer s experience in the field (nine years). The model has an accuracy of 98.333340883255%.

- Creating, implementing, training, and validating a CNN model with three convolutional layers which can classify parts of specification text as complex or not complex. The degree of complexity is based on the verification engineer s experience in the field (nine years). The model has an accuracy of 96.66666984558105%.

6.3. List of original publications

- **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. Perspectives for further developments

There are quite a few perspectives of the impact of AI in the function verification field using the LLM:

- Expansion to Coverage-Driven Verification (CDV): LLMs could be further used to automate the generation of coverage models.
- Automated Debugging Assistance: Expanding LLM capabilities to assist in debugging could provide engineers with automatic suggestions for resolving errors in the UVM environment.
- Enhanced Protocol Support: LLMs could be trained to handle a broader range of protocols, integrating deep knowledge of specific industry standards such as PCIe, USB, and Ethernet, to facilitate faster verification in multi-protocol SoCs.
- Natural Language to System Verilog UVM: Further improving the ability of LLMs to translate natural language descriptions of verification requirements directly into UVM components could make the verification process more accessible to those without extensive coding expertise.

Chapter 7. Bibliography

[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).