Trials of using Generative AI for APB UVM testbench generation

Diana DRANGA

Faculty of Electronics, Telecommunication, and Information Technology,

National University of Science and Technology Politehnica Bucharest, Romania

dranga.diana4@gmail.com

Abstract: As the production of electronics increases exponentially each year, the need for them to be able to function properly (according to the documentation) is critical. The functional verification process is a complex and time consuming step when delivering a proper System-On-a–Chip, which makes it a great candidate for different AI implementations and try-outs. In functional verification the UVM Methodology, where UVM stands for Universal Verification Methodology is vastly used combined with a hardware verification language such as System Verilog. Writing various, elaborated testbenches for difficult architectures such as RDMA (Remote Direct Memory Access), USB (Universal Serial Bus), etc. and achieving 100% functional coverage is laborious. Here, Generative AI may be able to help to improve this process by a considerable amount. This paper aims to present the different trials, aspects and challenges of producing an APB UVM testbench with Generative AI, using ChatGPT.

Keywords: Functional Verification, Generative AI, UVM, SystemVerilog, APB, System-On-a-Chip, ChatGPT.

Experimente folosind AI Generativ pentru generarea unui mediu de verificare APB UVM

Rezumat: Pe măsură ce producția de electronice crește exponențial în fiecare an, necesitatea funcționării corecte a acestora este esențială. Procesul de verificare funcțională este complex și consumă mult timp pentru livrarea unui System-On-a-Chip, fapt care îl face un candidat ideal pentru implementarea diferitelor experimente cu Inteligență Artificială. În verificarea funcțională se folosește Metodologia UVM (Universal Verification Methodology) în combinație cu un limbaj de verificare hardware SystemVerilog. Implementarea unor medii de verificare, variate, laborioase pentru arhitecturi complexe cum ar fi RDMA (Remote Direct Memory Access), USB (Universal Serial Bus) etc. cu 100% acoperire a testelor necesită multă muncă. În acest caz AI generativ poate fi de ajutor în facilitarea acestui proces. Acest articol își propune să prezinte diferite implementări, experimente și probleme întâmpinate în generarea unui mediu de verificare APB (Advanced Peripheral Bus) utilizând Metodologia UVM cu ajutorul ChatGPT.

Cuvinte cheie: Verificare funcțională, AI Generativ, UVM, SystemVerilog, APB, System-On-a-Chip, ChatGPT.

1. Introduction

As today's Integrated Circuits development evolves in both complexity and competence, the need for the Integrated Circuits to behave according to documentation is critical in order to ensure the safety and correct running of the application. Industries such as aviation, medical and automotive need Integrated Circuits which are reliable and robust.

In general, when developing a System-On-a-Chip, there are several steps needed such as defining the specification and need for the system, architecting the system, implementing the system in a HDL (Hardware Development Language) such as Verilog or HDL, functional verification, physical design, and a lot more processes as presented in the Figure 1.

https://doi.org/10.33436/v34i2y202406

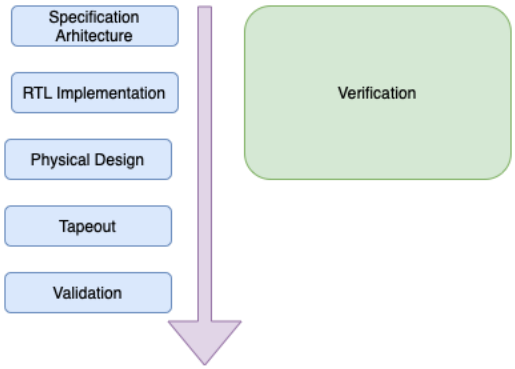


Figure 1. Steps when developing a System-on-a-Chip

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 verification activity continues even after tape out into silicon. Each component must be verified standalone and also it must be ensured that each one of them should communicate correctly and efficiently between each other, depending on what the system may require. The manner in which functional verification is done is presented in Figure 2 (Mammo, 2017).

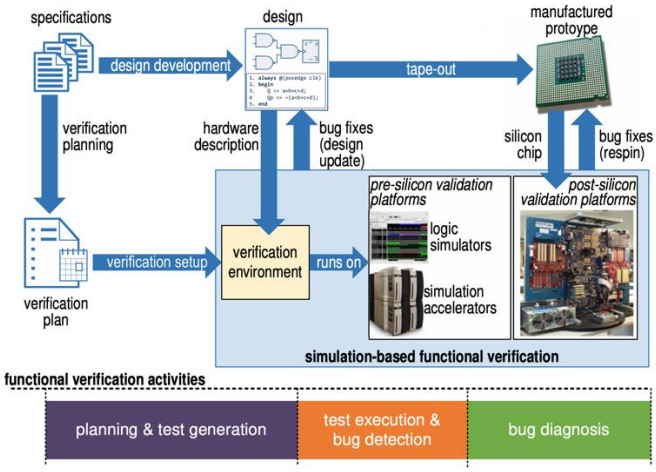


Figure 2. Verification process steps (Mammo, 2017)

The method presented in Figure 2 is of high complexity and time consuming. The most significant part of this process is focused on running, debugging, fixing the tests and also on completing coverage. Coverage is a metric that makes sure that the scenarios and configurations of the design have been verified using the tests run in the regression. The design itself is a simulation on software-based simulators like Cadence INCISIVE, Mentor QuestaSim and Synopsys VCS. Even though the simulators have been vastly improving throughout the years, becoming more and more faster and implementing advanced debugging capabilities, this is still not good enough for the verification process to complete in less time. This is where the aid of Artificial Intelligence might play an important role.

2. AI in functional verification and other research

Hardware functional verification is done usually in SystemVerilog as the verification plans, are usually written in natural language. The translation of these and the implementation of the verification environment must be done by the verification engineer.

Artificial intelligence can greatly improve the verification process in multiple ways. It can help in multiple areas of verification:

- Coverage completion. Using Artificial Intelligence, the stimuli of the test can be changed dinamically depending on which coverage score it is hit. 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). This process of obtaining high coverage is done by using Python script after the crucial parameters of the genetic algorithms and of the DUT functionality are set. The stimuli needed are read by sequences in the verification environment from .txt files output by Python. After compiling the verification environment and the DUT the simulation is run, and a coverage report is provided. Coverage information is taken out of this report and considered as a "fitness" metric (Dinu et al., 2022). After each completion, the best stimuli are saved on the drive;
- 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. State-of-the-Art

When the transformer neural network architecture was introduced in 2017 a new era in machine learning by making available different generative AI pretrained models (foundation models) has commenced (Morris, 2023). These models open up possibilities for interaction between humans and computers, as well as collaboration with AI, enhancing the capabilities of researchers and engineers (Morris, 2023). The LLMs such as ChatGPT3 can offer great support in order to help researchers simulate specific environments (Morris, 2022). Even though there are a lot of advantages of Generative AI, it still has many limitations such as offensive content, ethical content, etc. There is also the problem of the origin of the training materials, whether they are correct or not (Bender et al., 2021; Weidinger et al., 2021).

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). The recent growing LLMs assistants like ChatGPT (OpenAI, ChatGPT) and (Tom et al., 2020) supports researchers and engineers to understand various specifications and navigate multiple intricate lines of code. Furthermore, what is more interesting is the ability of the LLMs to generate code for the application it was asked for. Applying LLMs to software developement has gathered attention and wide popularity for general purpose use (Murr, Grainger & Gao, 2023). The model's evaluation has been tried with different tasks such as Parson's Problems (Reeves et al., 2023), CS1 Problems (Denny, Kumar, Giacaman et al., 2022), Data science (Lai et al., 2022) and competitive programming (Li, 2022). In Murr et al. (2023) the authors highlight four approaches in the way the task was asked to be completed by the LLM, in order to asses the apadability of the model. The four approaches consist of:

- Prompt only, where the problem was presented as a statement with a minumum amout of information;
- Prompt with tests, where the problem was given alongside with example testcases;
- Prompt tests only, where only the tests were given without the body text of the problem;
- Prompt generic tests, where tests were provided to the LLM but different functions were masked.

From the point of view of the model performance, Murr et al. (2023) have presented the Prompt Tests Only approach was the most efficient (231/315), followed by Prompt with Tests (228/315), Prompt Only (222/315) and the last approach Prompt Generic Test (200/315) being the least effective. The model that outperformed vastly the other ones was ChatGPT3.5.

Aside from the text capabilites, with the release of ChatGPT4, new opportunities have emerged since the latest release can be fed image into the LLM. This might improve vastly the way the LLM can interpret both images and text since now it can correlate between the two. In one paper (Johnson et al., 2023) ChatGPT4 was given a number of images and asked four questions: As an AI expert can you identify this image?, What type of image do you think it is?, What is / are the usefulness of the image and the information it conveys? and What useful information can you extract from it?. This was aimed in order to assess ChatGPT4‘s capabilities on text and images correlated tasks (Johnson et al., 2023). When the LLM was fed and asked the above questions it responded correctly regarding what the images were and their respective meaning. When the images of a wide spectrum (a chart, an architectural framework for transfer learning model for heart signal classification, a dog, a couple of pictures of humans and different plots of machine learning dynamics) were being given into the LLM, ChatGPT4 described the images with high accuracy defining what they are, gave good suggestions for the questions asked, thus promising very optimistic results in the future. Although, some details such as a pen in one pocket in the image with a human was missed (Johnson et al., 2023).

In order to analyse and comprehend the impact of Generative AI on hardware verification it is required to conducts tryouts and studies like this one.

In this paper, 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.

4. UVM Testbench generation from scratch using ChatGPT4

In these days, the UVM methodology alongside with SystemVerilog is used to define and implement testbenches. The usual UVM SystemVerilog testbench consists of:

- Item, a class which contains user defined data members which represent the protocol properties;
- Driver, which is responsible to drive the data correctly into the DUT, according to the protocol rules defined in the specification;
- Monitor, its role consists of collecting the data from the bus in accordance with the protocol rules;
- Sequencer, a handshake between driver and sequence;
- Agent, contains the driver, monitor, sequence and item. An agent can be passive or active. Passive in the terms of just collecting data with the monitor and activate when it is needed to drive data into the DUT;
- Scoreboard, where the comparison with the actual value from the DUT is compared to the expected computed value;
- Sequence, is an object which has the duty to create transactions (or items) or initiate other sequences. If the sequences create transactions it is a must to run them on a sequencer;
- Test, which is a component that instantiates the sequences if it is the case or just overrides the base sequence with the one needed for that specific scenario.

These explanations are clarified more on Figure 3.

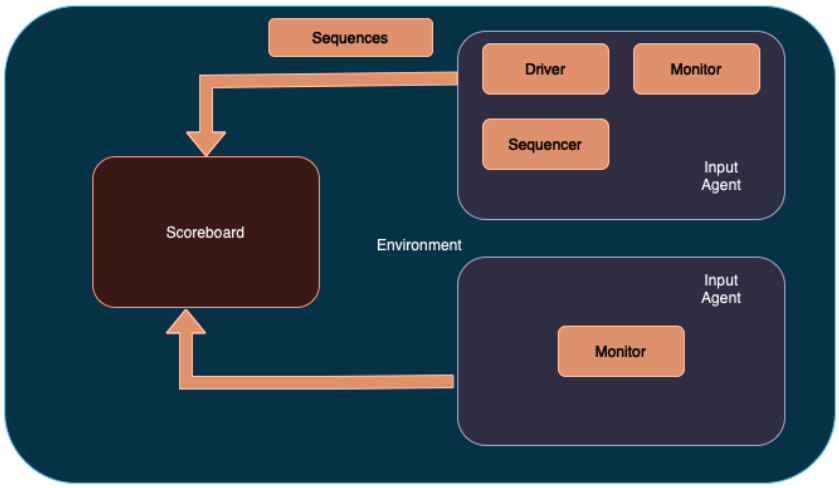
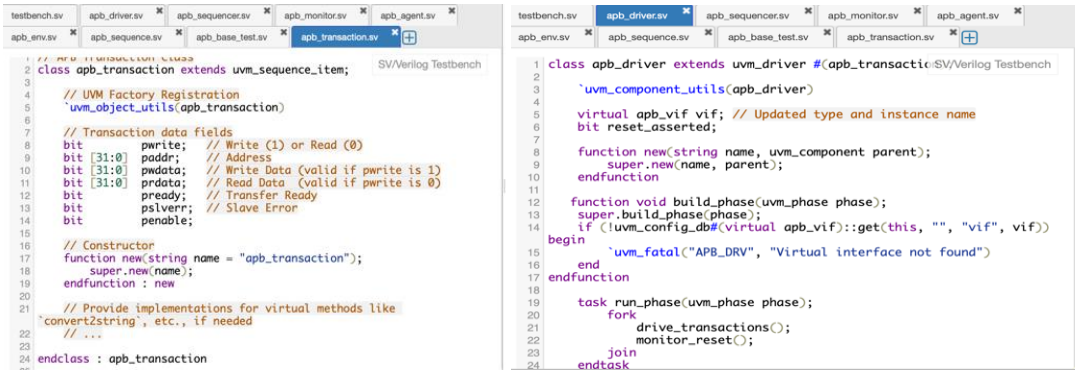


Figure 3. Generic UVM testbench

In this paper 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 components have been generated using general English and asking the LLM to provide the code to the output. 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.



```
class apb_transaction extends uvm_sequence_item;

    // UVM Factory Registration
    `uvm_object_utils(apb_transaction)

    // Transaction data fields
    bit        pwrite;   // Write (1) or Read (0)
    bit [31:0] paddr;    // Address
    bit [31:0] pwdata;   // Write Data (valid if pwrite is 1)
    bit [31:0] prdata;   // Read Data  (valid if pwrite is 0)
    bit        pready;   // Transfer Ready
    bit        pslverr;  // Slave Error
    bit        penable;

    // Constructor
    function new(string name = "apb_transaction");
        super.new(name);
    endfunction : new

    // Provide implementations for virtual methods like
`convert2string`, etc., if needed
    // ...

endclass : apb_transaction
```

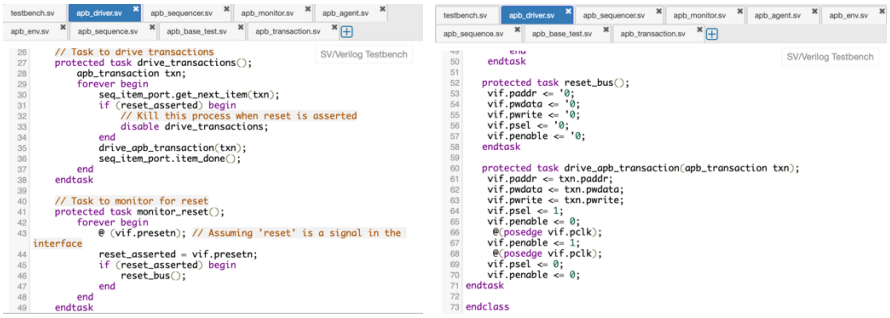
```
class apb_driver extends uvm_driver #(apb_transactio

    `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))
begin
        `uvm_fatal("APB_DRV", "Virtual interface not found")
    end
endfunction

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

A) B)

Figure 4 A) and B). Item and driver classes generated by ChatGPT4

ChatGPT4 was requested to generate a item or transaction class using UVM System Verilog which respects the AMBA APB protocol. In the first inquiry of ChatGPT4 the LLM output 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. In 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. Figure 4 A), B) and Figure 5 A), B) both illustrate the transaction and driver classes resulted by using the LLM and correcting the missing items.

The model was guided by the verification engineer to add the respective tasks and correct the missing code which it did successfully on a second run. From UVM perspective the UVM Factory was missing, this mechanism is important when trying to override components from the other classes, by not modifying any other line of code.



A) B)

Figure 5 A) and B). 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. The sequences instantiates an APB transaction item, creates it in a for loop for ten iterations, sets the pwrite signal to 1 signalling a write transfer and randomizing the data written for each iteration.

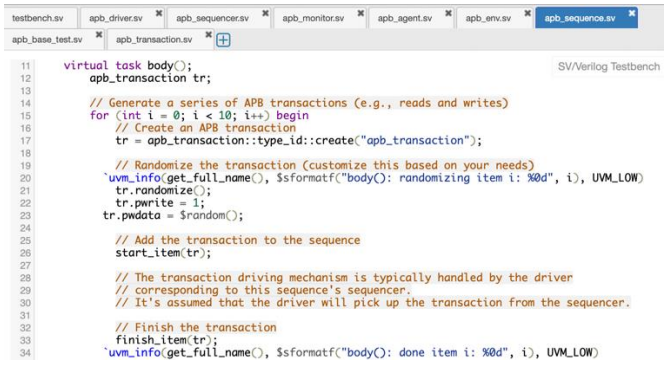


Figure 6. UVM Sequence

The sequence in Figure 6 was used in a UVM APB test generated by a LLM. The test created the sequence in the build_phase() function, created, randomized and started it in the run_phase() task of the class. The APB sequencer was used to start the sequence itself. After running the simulation, the waves were dumped in order to analyse it and ensure the correctitude of the testbench. Figure 7 shows the waveforms. Regarding the design part of this study, it was also generated using ChatGPT4, by asking the model to provide an APB module.

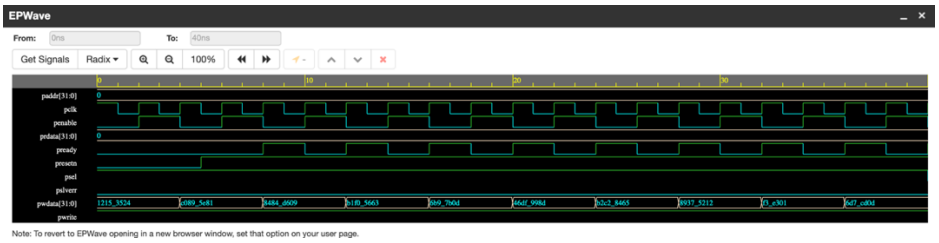


Figure 7. Results of the ChatGPT generated UVM testbench

The only file that was needed to be created by hand by the engineer was the top.sv file. ChatGPT4 can replace or correct name quite easily, but as needed in a top.sv file it cannot connect the interface to the virtual interface, uses wrongly and uncompilable code for uvm_set_config_db() and also for the run_test() method. Thus, it is a must for the moment to have the top.sv file edited manually. The Verilog implementation was also done by ChatGPT4 as requested by the engineer. At a first try, the LLM just output the code of the APB module, though missing the pslverr and penable implementations, but when retrying and asking specifically for these two signals to be added and driven, the model had no issue in doing so.

As a conclusion to this chapter, it has been proved that ChatGPT4 can generate a fully functional verification environment with respect to the protocol that was asked for. Instead of

spending a half a day on implementing the above testbench, using ChatGPT4 time spent reduced to about two hours (50% less spent time) in which most of it was spent on analysing the output and running it on a free platform which has some impediments as opposed to running on a dedicated environment. Although, the model can generate the code quite effective and correctly, it is still mandatory for the verification engineer or the researcher to supervise and correct possibile errors and missing items such as presented above. This problem can only escalate when larger and more complex designs are taken into consideration.

5. 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 generate 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. For simplicity reasons, an image of an APB write transaction was used in order to ask the model to generate the UVM Driver with respect to the protocol rules provided by the image. Figure 8 contains the image used and Figure 9 A), B) the resulted code.

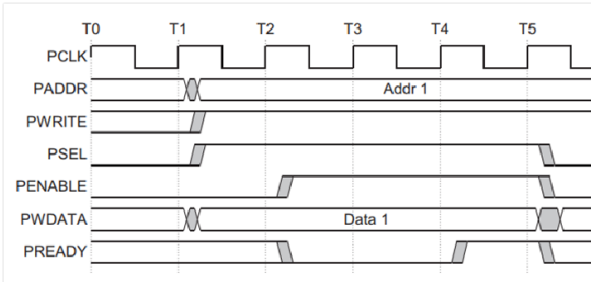


Figure 8. APB Write transaction waveforms (ARM, 2023)

```
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
```

A)

```
40      // Task to monitor for reset
41      protected task monitor_reset();
42          forever begin
43              @ (vif.presetn); // Assuming 'reset' is a signal in the interface
44              reset_asserted = vif.presetn;
45              if (reset_asserted) begin
46                  reset_bus();
47              end
48          end
49      endtask
50
51      protected task reset_bus();
52          vif.paddr <= '0;
53          vif.pwdata <= '0;
54          vif.pwrite <= '0;
55          vif.psel <= '0;
56          vif.penable <= '0;
57          #20;
58      endtask
59
60      protected task drive_apb_transaction(apb_transaction txn);
61          vif.paddr <= txn.paddr;
62          vif.pwdata <= txn.pwdata;
63          vif.pwrite <= txn.pwrite;
64          vif.psel <= 1;
65          vif.penable <= 0;
66          #10;
67          vif.penable <= 1;
68          #10;
69          vif.psel <= 0;
70          vif.penable <= 0;
71      endtask
72
73  endclass
```

B)

Figure 9 A) and B). Generated classes

When asked for the output, the LLM was not provided with any information regarding to the protocol rules, which protocol it is or other details, but the LLM understood that it is the ARM APB protocol. Some challenges that rose when using only the waveform is that, inside the drive_apb_transaction() method, instead of the #10 delays the LLM must be asked to synchronize with the protocol clock (pclk) as an alternative of just adding arbitrarily delays. If used such, these delays might generate desynchronizations and errors if the class is integrated in the environment and the environment itself is integrated into a higher level SoC environment. Moreover, such faults

will cause errors which might be hard to trace and debug. The verification engineer must thoroughly understand and analyse the code given by the LLM for the number of errors to be diminished.

This approach is valuable when new change requests are added in the documentation and then into the RTL design. Change requests are usually modifications added into the documentation and the RTL design which can change slightly or more the behaviour of the RTL. The Digital Design engineer can provide in the documentation or using any of the simulators (CADENCE INCISIV, Questa VCS and Siemens) a correct waveform of what a new transfer must look like. Thus, the verification engineer can feed these waveforms into ChatGPT4 and generate the updates required for the verification environment.

6. Conclusions

With exponential growth in electronic production, ensuring their proper operation as specified in the documentation becomes crucial. The functional verification phase is a complex and time-consuming aspect of System-on-a-Chip development, making it an ideal area for applying and experimenting with various AI technologies. The Universal Verification Methodology (UVM) is commonly used in this phase, alongside a hardware verification language as System Verilog. Implementing difficult and tricky testbenches for protocols such as RDMA, USB etc. and achieving 100% functional coverage is challenging. This paper aimed to prove the usage of LLMs such as ChatGPT4 in code generation of a fully functional UVM APB testbench and using images to generate protocol specific UVM components.

For the first approach, in this paper an UVM APB testbench was generated using ChatGPT4. The LLM was asked to provide the item, driver, monitor, sequencer, agent, scoreboard, sequence and test for the UVM APB Testbench and the verification engineer being a supervisor to correct the results output by the LLMs. The missing software constructs were added by ChatGPT4 when requesting it explicitly by the engineer. The top module itself was not generated ChatGPT4, as when trying to do so, the LLM just output generic top modules and when asked to modify certain interfaces and parameters it was not accurate resulting in different changes than expected. Thus, it is more convenient for the functional engineering process to manually configure the top module instead of relying on ChatGPT4. The RTL implementation of the APB modules was also done using ChatGPT4, albeit on a first run the LLM did not provide any implementation for the pslverr and penable signals. On a second run and requesting explicitly to be added the model added the respective implementation for the signals. Time spent on analysing the outputs and running the code on a free platform was about two hours, as compared to four hours which would’ve taken to implement each class by hand, consequently reducing the time by 50%. Another note might be that most of the two hours were spent on configuring and running the code on the free platform, which in a dedicated environment would take even less time.

On the second approach, a picture was given to the model and asked to generate a UVM driver class for it. It was not specified what protocol it is and what are the rules of the protocol, but the LLM realized it is an APB write transaction and lightly implemented the driver class as asked. In this approach, the engineer when provided with different waveforms can give these to the model and generate the classes need, instead of implementing them one by one. This will prove as a great advantage when having to implement countless protocols and classes as this method will reduce time spent on writing respective classes. The focus will shift on correcting and verifying the output that it corresponds to what is necessary.

These two approaches can decrease time spent 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 might have more control and more less time spent correcting ChatGPT4. This approach might be more viable when developing brand new verification environments, where there are a lot of configurations, details and scenarios needed. 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 means the module will suffer changes either trivial or substantial. By using the image present usually in any module or protocol specification, the engineer can quickly add new classes or correct the existing ones in the verification environment with respect the new changes. The verification engineer must be sure that the image provided to the LLM is correct and its output is also accurate. If there are any errors either ChatGPT4 can be asked using general English to correct the errors, or the verification engineer can do it manually.

At this moment there is an absence of Artificial Intelligence research used in functional verification, despite the rapid advances in AI. Therefore, it is a clear opportunity and demand for further research to integrate AI in the functional verification process. Such papers, may be able to alter dramatically the way tasks are conducted, providing numerous benefits from which the most important is reducing the time spent on verification through various techniques using different techniques for instance testbench generation, stimuli generation, etc.

This paper 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.

REFERENCES

ARM, (2023) *AMBA APB Protocol Specification*. https://developer.arm.com/documentation/ihi0024/latest/ [Accessed on January 2024].

Bender, M. E., Gebru, T., McMillan-Major, A. & Shmitchell, S. (2021) On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency. FAccT '21, March 3-10, 2021, Virtual Event, Canada*. New York, NY, United States, Association for Computing Machinery. pp. 610-623 [Accessed on March 2024].

Denny, P., Kumar, V., Giacaman, N. et al. (2022) Conversing with copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. To be published in *Human-Computer Interaction*. https://arxiv.org/abs/2210.15157 [Accessed on March 2024].

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 Verification of Digital Designs. *Micromachines*. 13(5), 691. doi:10.3390/mi13050691.

Johnson, O. V., Alyasiri, O. M., Akhtom, Dua'A & Johnson, O. E. (2023) Image Analysis through the lens of ChatGPT-4. *Journal of Applied Artificial Intelligence.* 4(2), 32-46. doi:10.48185/jaai.v4i2.870.

Khurana, A., Subramonyam, H. & Chilana, P. K. (2024) Why and When LLM-Based Assistants Can Go Wrong: Investigating the Effectiveness of Prompt-Based Interactions for Software Help-Seeking. To be published in *Human-Computer Interaction*. Accepted for publication in the *Proceedings of the 29th International Conference on Intelligent User Interfaces (IUI'24), March 18-21, 2024, in Greenville, SC, USA.* https://arxiv.org/abs/2402.08030 [Accessed on March 2024].

Lai, Y., Li, C., Wang, Y. et al. (2023) DS-1000: A Natural and Reliable Benchmark for Data Science Code Generation. To be published in *Proceedings of the 40th International Conference on Machine Learning, PMLR, 202, 2023. pp.18319-18345* and *Software Engineering*. [Preprint] https://arxiv.org/abs/2211.11501 [Accessed on March 2024].

Li, Y., Choi, D., Chung, J. et al. (2022) Competition-Level Code Generation with AlphaCode. To be published in *Programming Languages*. [Preprint] https://arxiv.org/abs/2203.07814 [Accessed on March 2024].

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].

Morris, M. R. (2023) Scientists' Perspectives on the Potential for Generative AI in their Fields. To be published in *Computers and Society*. https://arxiv.org/abs/2304.01420 [Accessed March 2024].

Morris, M. R., Carrie J. Cai, C. J., Holbrook, J., Kulkarni, C. & Terry, M. (2022) The Design Space of Generative Models. To be published in *Artificial Intelligence*. https://arxiv.org/abs/2304.10547 [Accessed on March 2024].

Murr, L., Grainger, M. & Gao, D. (2023) Testing LLMs on Code Generation with Varying Levels of Prompt Specificity. To be published in *Software Engineering*. https://arxiv.org/pdf/2311.07599 [Accessed on March 2024].

OpenAi, ChatGPT (2022) https://openai.com/blog/chatgpt [Accessed January 2024].

Revees, B., Sarsa, S., Prather, J. et al.(2023) Evaluating the Performance of Code Generation Models for Solving Parsons Problems With Small Prompt Variations. *ITiCSE 2023: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1, June 2023.* pp. 299–305. doi:10.1145/3587102.3588805.

Tom, B.B., Mann, B., Ryder, N. et al. (2020) Language Models are Few-Shot Learners. To be published in *Computation and Language*. [Preprint] https://arxiv.org/abs/2005.14165 [Accessed on March 2024].

Weidinger, L., Mellor, J., Rauhet, M. et al. (2021) Ethical and social risks of harm from Language Models. To be published in *Computation and Language*. https://arxiv.org/abs/2112.04359 [Accessed on February 2024].

Diana DRANGA is a Ph.D. student at the Faculty of Electronics, Telecommunication, and Information Technology, National University of Science and Technology Politehnica Bucharest. She holds a master's degree in advanced software technologies in Communications and a bachelor degree in engineering, obtained from the Polytechnic University of Bucharest, Faculty of Electronics, Telecommunication, and Information Technology. She is a verification engineer with almost nine years of experience in various projects and companies such as Infineon, Microchip, NXP, Amazon and Intel. Her main subject of research consists of how AI can aid the process of functional verification in different aspects of the project.

Diana DRANGA este student doctorand la Facultatea de Electronică, Telecomunicații și Tehnologia Informației, Universitatea Națională de Știință și Tehnologie Politehnica București. Deține o diplomă de master în Tehnologii Software Avansate în Comunicații (TSAC) și o diplomă de licență în inginerie obținută de la Universitatea Națională de Știință și Tehnologie Politehnica București, Facultatea de Electronică, Telecomunicații și Tehnologia Informației. Cu aproape nouă ani de experiență în domeniul ingineriei de verificare, aceasta a lucrat în diferite proiecte și diverse companii precum Infineon, Microchip, NXP, Amazon și Intel. Cercetarea sa se concentrează pe modul în care Inteligența Artificială poate îmbunătăți procesul de verificare funcțională.

This is an open access article distributed under the terms and conditions of the Creative Commons Attribution-NonCommercial 4.0 International License.