GENERATIVE AI IMPACT FOR GENERATION OF A DUAL CLOCK FIFO UVM TESTBENCH

Madalina-Florentina DEDU, Corneliu ZAHARIA

"Transilvania" University of Braşov, Romania (florentina.dedu@student.unitbv.ro, corneliuzaharia@unitbv.ro)

DOI: 10.19062/1842-9238.2024.22.2.2

Abstract: *The rapid evolution of generative AI technologies has brought transformative changes to the field of testbench generation, significantly enhancing the efficiency and accuracy of hardware and software testing. Traditional testbench development is a labor-intensive process, often requiring substantial manual effort to simulate various scenarios and edge cases. This article explores the impact of generative AI on UVM testbench creation, examining its benefits, challenges, and future potential in the verification and validation domains. Key use cases are also discussed, highlighting how AI-driven testbench automation can streamline design verification, reduce time-to-market, and improve product reliability across industries. The example used is for a dual Clock FIFO Device Under Test (DUT).*

Keywords: *UVM-Universal Verification Methodology, Generative AI-Artificial Intelligence, FIFO-First In First Out, Testbench*

1. INTRODUCTION

In this digital era, technology continues to evolve and innovate at an accelerated rate. Among the many advancements, Artificial Intelligence (AI) stands out as a particularly prominent and transformative trend. The most popular Generative Artificial Intelligence (GenAI) model is ChatGPT which is a chatbot and virtual assistant developed by OpenAI and launched on November 30, 2022 [1]. Based on Large Language Models (LLMs), it enables users to refine and steer a conversation towards a desired length, format, style, level of detail, and language. AI is revolutionizing various industries, from healthcare and finance to entertainment and transportation, by enabling more efficient processes, enhancing decision-making, and creating new opportunities for innovation. As AI technology progresses, it promises to shape the future in unprecedented ways, making it a key focus for researchers, developers, and businesses worldwide.

The semiconductor industry is not unaffected by this trend either. A key question arises: can we leverage GenAI to make the work of engineers easier? By integrating GenAI into semiconductor design and manufacturing processes, engineers can potentially streamline complex tasks, improve precision, and accelerate innovation. GenAI algorithms can assist in optimizing chip design, predicting failures, and enhancing the overall efficiency of production. As the industry continues to embrace GenAI, we may witness significant improvements in both the quality and speed of semiconductor development, ultimately driving further advancements in technology.

We decided to evaluate ChatGPT's and BingAI's ability to write and verify a Dual Clock FIFO module. Microsoft introduced Bing Chat in February 2023 as a revolutionary web search tool and users' "copilot for the web." This AI-powered chatbot version of the Bing search engine is designed for use with the Microsoft Edge browser and requires a Microsoft account [2]. Bing AI, offers a comprehensive range of search services, including web, video, image, and map search capabilities, all developed using ASP.NET [3]. However, we discovered that there are limited online resources available for UVM verification and its components, which constrained their ability to fully develop a comprehensive verification environment. We needed to make several adjustments to ensure a functional environment free of compilation errors. ChatGPT successfully generated RTL code without syntax or compilation issues, but we identified areas where functionality needed improvement. This experience highlighted the potential of GenAI like ChatGPT and BingAI in assisting with code generation but also underscored the importance of resource availability in hardware development projects.

2. FIFO DUAL CLOCK

A Dual Clock FIFO (First-In-First-Out) module is a type of memory buffer that allows for the storage and retrieval of data between two systems operating at different clock frequencies. It is commonly used in digital systems to handle data transfer between components that do not share the same clock domain. In implementing the FIFO module, we decided to use two pointers for reading and for writing. Based on the values of these pointers, we also generate the empty and full signals to indicate the status of the FIFO buffer.

The read pointers help in tracking the current position from which data should be read, as well as the next position to be read. Dual-pointer method ensures accurate data retrieval and helps prevent errors that may arise from concurrent read operations. The write pointer, on the other hand, tracks the position where new data should be written [4]. This pointer-based approach not only simplifies the management of the FIFO but also enhances its efficiency by providing clear indications of its state. By accurately maintaining and comparing the read and write pointers, we ensure reliable data flow and effective buffer management in the FIFO module.

To determine whether a FIFO is full or empty, some form of mathematical manipulation or comparison of the write and read pointers is necessary. The challenge arises because these pointers are generated in two separate clock domains, meaning that one or both pointers must be synchronized with the opposite clock domain before performing any mathematical or comparison operations to ensure accuracy and safety. A good method for passing pointers between clock domains is to use a gray-code counter for the two FIFO pointers. In a binary counter, multiple bits may change simultaneously when counting up or down, which can create issues when sampled by a slower or unsynchronized clock domain. Gray code counters solve this problem by ensuring that only one bit changes at a time between successive values, minimizing the risk of metastability and making it easier to synchronize between clock domains [5]. A high-level overview of the proposed dual clock shared buffer is illustrated in Fig. 1.

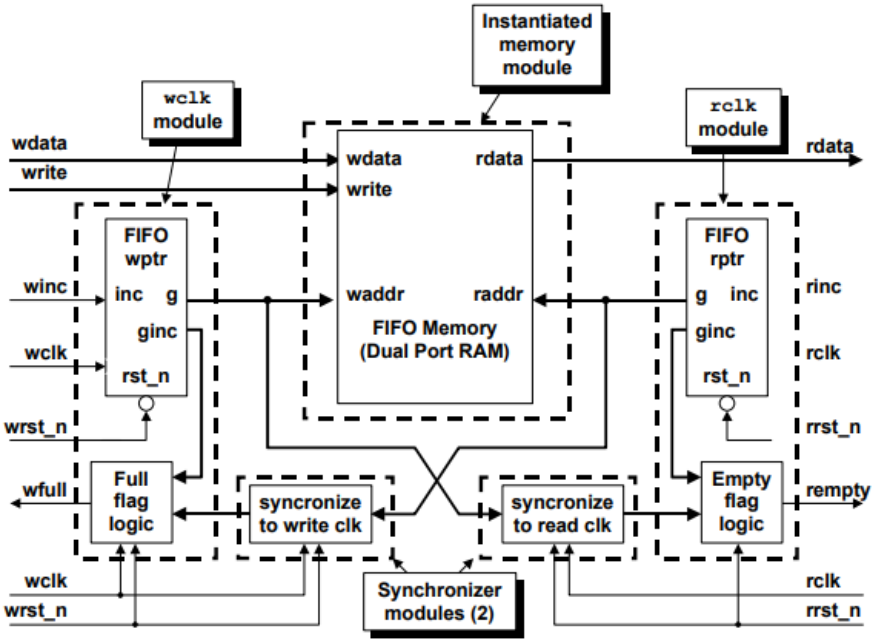


FIG. 1. FIFO Block Architecture (source:[5])

An important problem in systems lacking a single global timing reference is synchronization. The timing relationship between a signal and a clock generally falls into one of five categories :

• synchronous; the signal and the clock are perfectly aligned. Every signal transition occurs precisely with the clock edge, ensuring consistent timing across the system. This alignment facilitates easy data transfer and coordination between different parts of the system.

• mesochronous; the signal and the clock share the same frequency but maintain a constant phase difference. This means that while they operate at the same speed, their transitions are offset by a fixed amount of time. This constant phase difference can be managed through Phase-Locked Loops (PLLs) or delay elements to ensure proper synchronization.

• plesiochronous; the signal frequency is close to but not identical to the clock frequency. This slight difference results in a varying phase difference over time, necessitating sophisticated synchronization techniques. Buffering and elastic storage elements are often used to accommodate the variations and prevent data loss or misalignment.

• periodic; the signal has a repetitive nature but lacks a defined relationship to the clock. Although the signal repeats at regular intervals, it is not synchronized with the clock edges.

• asynchronous; the signal is completely independent of the clock, with transitions occurring at arbitrary times. This lack of timing correlation presents significant challenges for synchronization. Techniques such as handshake protocols and asynchronous FIFOs are employed to manage data transfer and maintain integrity across different clock domains [6].

Metastability is a fundamental issue that arises when interfacing asynchronous blocks. It occurs when registers do not receive a stable input signal near the active edge of the clock signal, leading to unpredictable behavior and potential system failures [7].

Metastability-free operation depends on ensuring that the receiver does not read the signal until a sufficient amount of time has passed, allowing the signal to stabilize and preventing incorrect or unpredictable behavior. This safe time period ensures the signal has fully transitioned to a stable state before being sampled by the receiving system [4]. To mitigate this problem, we decided to use Gray code pointers to synchronize the clocks. This technique helps in minimizing the risk of metastability by ensuring that only one bit changes at a time when pointers are updated, thereby providing a more stable transition and reliable data transfer between different clock domains.

3. UVM METHODOLOGY

The Universal Verification Methodology (UVM) is a standardized methodology for verification that focuses on reusability. By utilizing predefined classes and objects, it provides a consistent approach to building verification environments, making it easier to reuse components and ensure compatibility across different projects. UVM streamlines the process of developing, deploying, and maintaining verification systems, improving overall efficiency and flexibility in the verification process [8].

To validate the proposed concept, an HDL (Hardware Description Language) design specification is prepared, and a small team of HDL designers is assigned to the project. This team is responsible for implementing a digital system based on a Finite State Machine (FSM) following a detailed specification. Once each task is completed, the design is automatically validated using the proposed UVM methodology, and a performance evaluation of the validation process is conducted [9].

The figure below illustrates the complete structure of the Universal Verification Methodology.

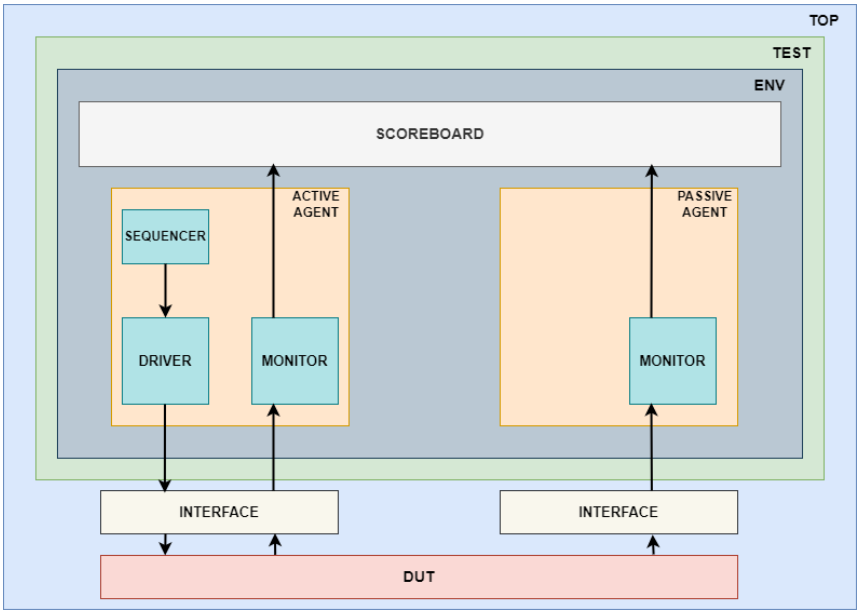


FIG. 2. Structure of UVM

In this diagram, two Agent classes are created. One of the agents is designated as active and the other one is passive. The agent serves as a container that bundles key verification components like Driver, Monitor, and sometimes a Sequencer. It can operate in active mode (driving signals to the DUT and monitoring outputs) or passive mode (only monitoring outputs).

The agent coordinates communication with the DUT and helps manage the flow of data. DUT (Design Under Test) : This is the hardware or system being tested.

The Driver converts high-level transactions (stimulus) into low-level signals that are applied directly to the DUT's pins. It is responsible for generating appropriate signal sequences based on the input provided by the verification environment, typically through a virtual interface. The Monitor observes the signals exchanged between the DUT and other system components. It captures the DUT's outputs and pass them to the Scoreboard.

A virtual interface abstracts the physical signal details of the DUT, providing a high-level interaction layer for components like Driver and Monitor. It connects the testbench to the DUT, allowing Driver to apply signals and Monitor to observe DUT behavior.

The DUT's behavior is verified against expected outputs under various test scenarios driven by the verification environment. The test environment is the overall setup in which the verification of the DUT takes place. The scoreboard is a verification component used to track and compare the expected outputs of the DUT against its actual outputs. It collects data from the Monitor and compares the observed results to a reference model or expected values to identify mismatches or errors. The top refers to the highest level of the testbench hierarchy. It integrates the entire verification environment and connects it to the DUT. It's responsible for instantiating the DUT, agents, and test environment, and configuring them to run simulations. The Coverage Model tracks which parts of the DUT's functionality and scenarios have been tested and ensures that all critical aspects of the design are exercised and validated during verification. It can be instantiated within the agent classes [10].

4. USING GENERATIVE AI

ChatGPT:

In initial attempt at generating a functional testbench and RTL was unsuccessful because we didn't receive the Verilog code for the required files. Instead, it generated a skeleton and provided the following response regarding the difficulties of generation: "Building a UVM testbench is a detailed and iterative process. It's crucial to have a solid understanding of the UVM methodology and SystemVerilog. Due to the complexity and the need for iterative development and debugging, it's recommended to start with basic functionality and gradually add features and checks."

In response, we revised our requirements and requested each component in more detail. This led to the generation of some code, but it contained numerous compilation errors. These included misspelled variable names, an improperly configured reset (active low), incorrect signal assignments from the interface, and an improper sequence—data randomization should have occurred between start_item and finish_item. Additionally, the parameter in the clocking block of the interface needed to be removed.

After addressing these issues, we achieved our first version of a functional environment and a running test. However, the test revealed that the DUT was not functioning correctly, as the empty and full signals were not asserted and deasserted at the appropriate times (see Fig 3). From this point, we began developing a more complex and accurate verification module using UVM methodology with the help of ChatGPT and BingAI.

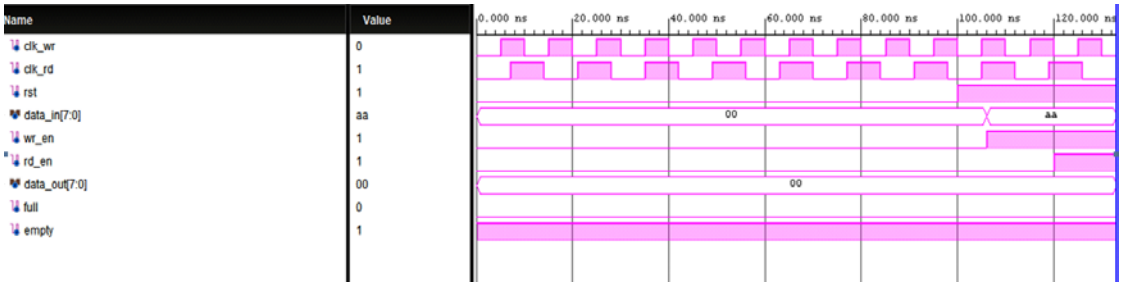


FIG. 3. First successful test

We began by attempting to fix the DUT, focusing on resolving the issues with the empty and full signals. We asked the AI to modify the code, and the proposed solution involved creating a counter that increments during write operations and decrements during read operations. When the counter reaches 16, the full signal is asserted, and when it reaches 0, the empty signal is asserted. While this approach worked, it was not suitable for the design, as it is not synthesizable.

We also encountered challenges with pointer synchronization. After multiple iterations, during which we had to explain in detail how to modify the conditions and specify which pointers to include, we finally succeeded in developing a proper DUT module. Once the DUT was fixed, we started creating more complex sequences and tests, while also making corrections to other components. In the driver, we added a reset condition, and in the monitor, we included all the signals from the interface in the item. Additionally, we created a coverage file and, most challenging of all, a scoreboard file. Finally, we achieved some well-functioning tests. An example is shown in Fig. 4, where we successfully simulated read and write operations.

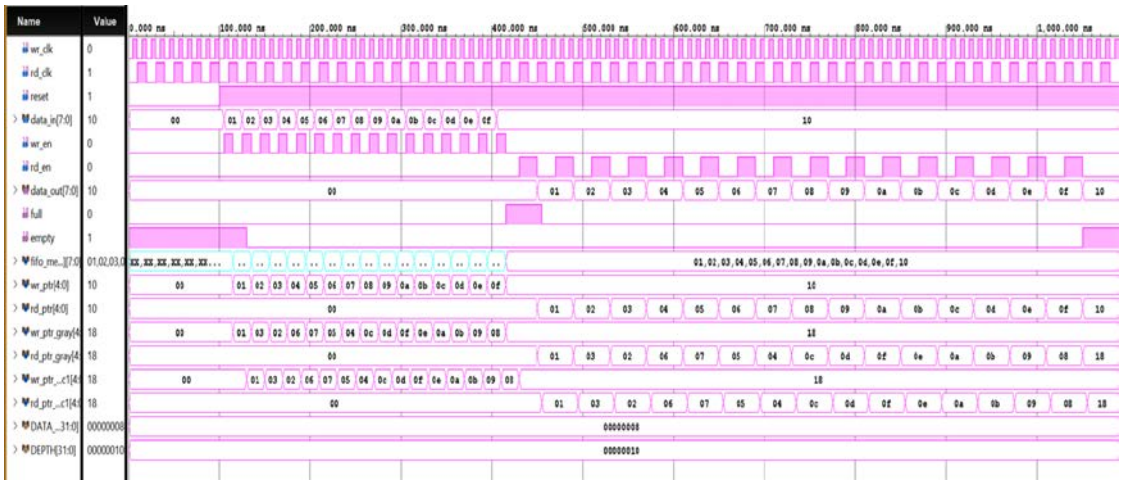


FIG. 4. Write all Read all test

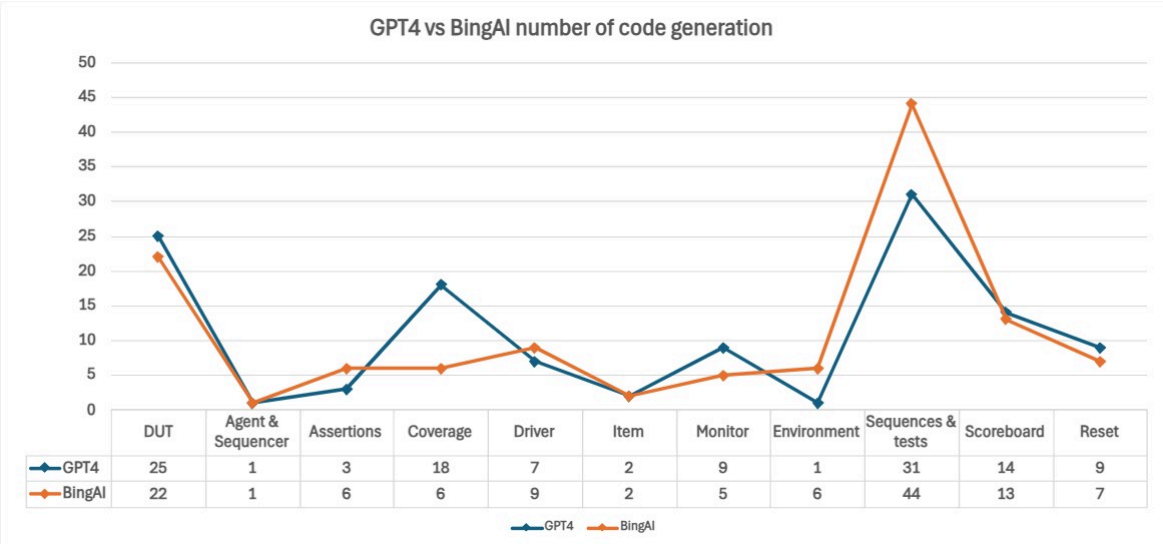
BingAI:

The next step in our process was to attempt developing the same environment using Bing AI. This time, we decided to build upon the basic environment initially generated by ChatGPT, which had several issues and wasn't functioning correctly. By using Bing AI, we aimed to address these problems and enhance the environment's functionality, hoping to achieve a more robust and reliable solution. The experience we have gained from using ChatGPT in the past has been put to good use this time. We better understood how to ask the right questions, which references to include and when to provide them.

This knowledge allowed us to communicate more effectively with AI, which sometimes led to more accurate and relevant results. The most challenging part, similar to those encountered when using ChatGPT, was generating of the scoreboard. This task requires extensive knowledge, including a detailed understanding of the design's specifications, architecture, and behavior. Additionally, it necessitates the ability to manage various test scenarios, including corner cases, and requires thoughtful planning to ensure that these tests cover all potential issues and accurately reflect the intended functionality of the design.

What have we learned:

After generating the DUT and Verification Environment using two large language models—ChatGPT-4 and Bing AI—we compared the number of code generations produced by each model. The graph, in Fig. 5, illustrates the number of code iterations generated by ChatGPT-4 and Bing AI for each component of the verification environment.



	DUT	Agent & Sequencer	Assertions	Coverage	Driver	Item	Monitor	Environment	Sequences & tests	Scoreboard	Reset
GPT4	25	1	3	18	7	2	9	1	31	14	9
BingAI	22	1	6	6	9	2	5	6	44	13	7

FIG. 5. Comparison between the number of code generations for each model

• Communication is key – we learned how to ask the right questions to get the desired answer

• The references are important - there needs to be background on a topic to generate correctly

• In a single question, all the requests need to be clearly and explicitly stated. For instance, the AI won't be able to generate the desired code if we refer to something that was mentioned above in the same conversation.

• Once we knew what the correct prompt was, after several trials with ChatGPT, we were able to use that from the beginning when we generated the code with Bing AI.

• Even if we specified that we wanted the signals to be assigned in separate "always" blocks, if we added a new signal, the generative AI did not take into account this requirement anymore.

5. CONCLUSION

Using generative artificial intelligence to generate design and verification code has enormous potential, offering opportunities to speed up the development process and reduce human error.

GenAI can automate repetitive tasks, generate quick solutions and explore complex scenarios that might be difficult to cover manually. However, there will always be a need for an engineer to verify the accuracy and validity of the information generated. Engineers bring a deep understanding of context, the ability to identify design subtleties and nuances, and are essential to ensuring that the results provided by AI are accurate, relevant, and compliant with industry specifications and standards. In essence, AI can be a powerful tool, but human supervision remains crucial to the ultimate success of any engineering project.

REFERENCES

[1] M. Abdullah, A. Madain and Y. Jararweh, *ChatGPT: Fundamentals, Applications and Social Impacts*, 2022 Ninth International Conference on Social Networks Analysis, Management and Security (SNAMS), Milan, Italy, 2022, pp. 1-8;

[2] G. Bubaš, S. Babić and A. Čižmešija, *Usability and User Experience Related Perceptions of University Students Regarding the Use of Bing Chat Search Engine and AI Chatbot: Preliminary Evaluation of Assessment Scales*, 2023 IEEE 21st Jubilee International Symposium on Intelligent Systems and Informatics (SISY), Pula, Croatia, 2023, pp. 000607-000612;

[3] Y. Mehdi, *Reinventing search with a new AI-powered Microsoft Bing and Edge, your copilot for the web*, Official Microsoft Bing Blog, Microsoft, February 7th, 2023, https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-witha-new-ai-powered-microsoft-bing-and-edge-your-copilot-for-the-web/ (accessed June 30, 2023);

[4] D. Konstantinou, A. Psarras, C. Nicopoulos and G. Dimitrakopoulos, *The Mesochronous Dual-Clock FIFO Buffer*, in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 28, no. 1, pp. 302-306, Jan. 2020;

[5] Verilog, Expert & Cummings, Clifford. *Synthesis and Scripting Techniques for Designing Multi Asynchronous Clock Designs*;

[6] R. W. Apperson, Z. Yu, M. J. Meeuwsen, T. Mohsenin and B. M. Baas, *A Scalable Dual-Clock FIFO for Data Transfers Between Arbitrary and Haltable Clock Domains*, in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 15, no. 10, pp. 1125-1134, Oct. 2007;

[7] W. J. Dally and J. W. Poulton, *Digital Systems Engineering. Cambridge*, U.K.: Cambridge Univ. Press, 1998;

[8] N. Bhuvaneswary, J. Deny and A. Lakshmi, *Exploration on Reusability of Universal Verification Methodology*, 2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), Greater Noida, India, 2022, pp. 1865-1868, doi: 10.1109/ICACITE53722.2022.9823570;

[9] M. Savić and N. Pjevalica, *Advanced UVM-Based Verification Methodology for Consumer Electronics HW Design Evaluation and Validation*, 2024 Zooming Innovation in Consumer Technologies Conference (ZINC), Novi Sad, Serbia, 2024, pp. 104-107;

[10] Accellera, *Universal Verification Methodology (UVM*) 1.*1 User's Guide*. (2011).