ISSN 0002-306X. Proc. of the RA NAS and NPUA Ser. of tech. sc. 2024. V. LXXVII, N3

UDC 621.3.049 **MICROELECTRONICS**

DOI: 10.53297/0002306X-2024.v77.3-367

A.G. MANUKYAN

DESIGN AND UVM BASED VERIFICATION OF FTL MEMORY CONTROLLER

This paper presents the design and implementation of a memory controller based on FIFO-to-FIFO transfer level (FTL) design methodology, which lifts the abstraction level from RTL to value-passing semantics of FIFO to FIFO transactions, or mixed model where global communication is done with FTL and local modules can be implemented with lower level abstractions such as RTL. While FTL architectures provide a powerful framework for handling complex data transactions, verifying these designs poses significant challenges due to their intricate communication patterns and timing violations. For the functional verification of such systems, a special testing environment is needed where the data transfers with FIFO modules will be calculated. To create such a testing environment, the Universal Verification Methodology (UVM) is used, which makes it possible to perform testing that meets the modern requirements of digital circuits. The proposed method makes it is possible to test the FTL memory controller by increasing the test coverage to 100%.

Keywords: functional verification, UVM testbench, Memory Controller, Random access memory, FIFO-to-FIFO tranfer level.

UMV. The Universal Verification Methodology is a standardized framework for functional verification that provides a comprehensive approach to verifying complex digital designs [1]. Developed to address the increasing complexity of modern semiconductor designs, UVM is built upon the IEEE 1800 SystemVerilog standard and incorporates the best practices in verification methodology. UVM emphasizes key principles such as reusability, standardization, and automation, making it a powerful tool for creating scalable and maintainable testbenches [2-5].

Introduction to FTL. FTL (FIFO to FIFO Transfer Level) transactions represent a design model that specifies data transactions through sequential processes, utilizing message-passing interfaces for non-blocking communication [6,7]. This model is particularly suited for managing complex data flows in digital systems by leveraging FIFO (First-In, First-Out) queues. In the FIFO Transfer Level (FTL) abstraction, FIFO queues replace traditional registers to handle data storage and transfer between various components. This design approach is advantageous in scenarios where data flow must be managed sequentially, ensuring that data is processed in the exact order it was received [8].

A key advantage of the FTL approach is its support for asynchronous data transfer. When transferring data from one module to another, the initiating module does not need to wait for a response from the receiving module. Instead, it can simply push data into the FIFO queue and continue processing other transactions or interacting with different modules. This asynchronous mechanism allows for greater flexibility and efficiency in data management, as the module can proceed with other tasks without being blocked by the data transfer process.

The use of FIFOs in the FTL abstraction streamlines data flow, enhances modularity, and allows for scalable system design. It supports precise control over data processing and reduces the risk of data corruption or loss, making it ideal for complex systems requiring reliable and efficient data handling.

One more advantage of FTL is it's correct by constructions design pattern where design functionality is separated from data communication, which gives strong modularity to the design. Utilizing the FIFO-2-FIFO data transactions between the modules gives flexibility in design customization at every stage [9].

FTL Memory Controller. The FTL Memory Controller is a crucial component responsible for managing the flow of data between the system's FIFO (First-In-First-Out) buffer and the memory modules. It operates based on a finite state machine (FSM) with six distinct states: **IDLE, WAIT_ADDRESS_MODE, GET_ADDRESS_MODE, WAIT_TO_PUSH_DATA, WAIT_DATA,** and **GET_DATA**. The controller's primary functions include handling read and write operations, setting appropriate control signals, and managing data flow to and from the memory modules. The FSM diagram of FRL MEMORY CONTROLLER is shown in Fig. 1.

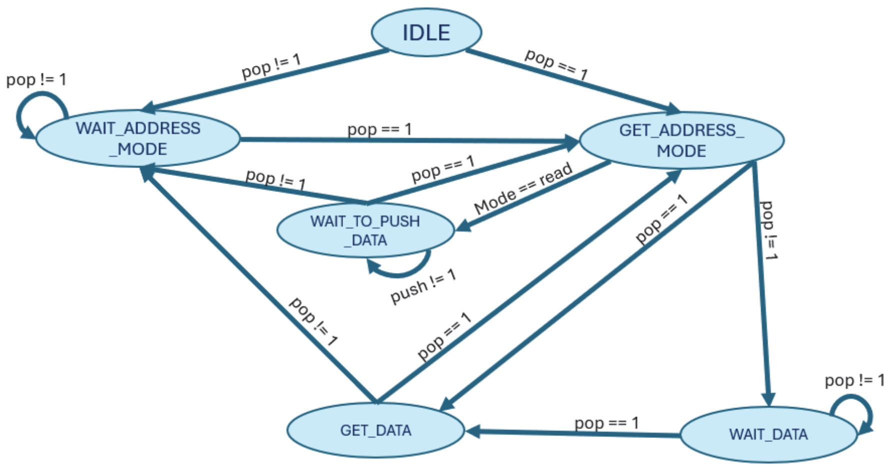

Fig. 1. FTL Memory Controller FSM

IDLE: The initial state where the controller inactivates MemoryEnable ports for all memories and awaits the arrival of a packet in the RX FIFO. If the RX FIFO is empty, the controller remains in the **WAIT_ADDRESS_MODE** state. Once the RX FIFO contains data, the controller transfers to the **GET_ADDRESS_MODE** state.

WAIT_ADDRESS_MODE: In this state, the controller waits for the first packet from the RX FIFO, which contains the MODE (either READ or WRITE) and the target memory address. If the RX FIFO remains empty, the controller stays in the **WAIT_ADDRESS_MODE** state. Upon receiving the packet, it transfers to the **GET_ADDRESS_MODE** state.

GET_ADDRESS_MODE: The controller processes the MODE and addresses information from the first packet. Based on the MODE:

- WRITE Mode: The controller sets the WriteEnable signal for the target memory module. If the RX FIFO has more data, the controller transitions to **GET_DATA**. If the RX FIFO is empty, it transthers to the WAIT_DATA state.
- READ Mode: The controller sets both MemoryEnable and ReadEnable signals for the target memory module and will move to **WAIT_TO_PUSH_DATA** state.

WAIT_TO_PUSH_DATA: In this state if f the TX FIFO is full, the controller stays in the **WAIT_TO_PUSH_DATA** state. When TX fifo becomes not full controller will move to **GET_ADDRESS_MODE** state if RX FIFO is not empty or will move to **WAIT_TO_PUSH_DATA** when RX fifo is empty. Upon receiving the packet, it transitions to the **GET_ADDRESS_MODE** state.

WAIT_DATA: If the RX FIFO is empty, the controller remains in the **WAIT_DATA** state. Once the RX FIFO contains data, the controller transitions to the **GET_DATA** state.

GET_DATA: In this state, the controller retrieves the data packet from the RX FIFO for writing to the memory. After setting the MemoryEnable to 1, the controller writes the data into the designated memory module. If additional data packets are available in the RX FIFO, the controller transfers to the **GET_ ADDRESS_MODE** state othervise controller transfers to the **WAIT_ADDRESS_MODE** state.

Architecture of Memory Controller with multi memories. The memory controller is designed to interface with o ROM module as Instruction register and two RAM module as Stack and Data register of varying sizes and configurations:

- **Instruction Segment:** 128x8 ROM
- **Stack Segment:** 1024X32 SRAM
- **Data Segment:** 2048X32 SRAM

The memory controller is connected to the following tree ROM and RAM memory modules, each with a unique configuration as shown in Fig. 2. Each of these modules is independently controlled by the memory controller through dedicated control signals such as MemoryEnable and WriteEnable. The architecture is centered around efficiently managing read and write operations to these memory modules, using a combination of an address decoder, control logic, and a FIFO buffer to streamline data transactions.

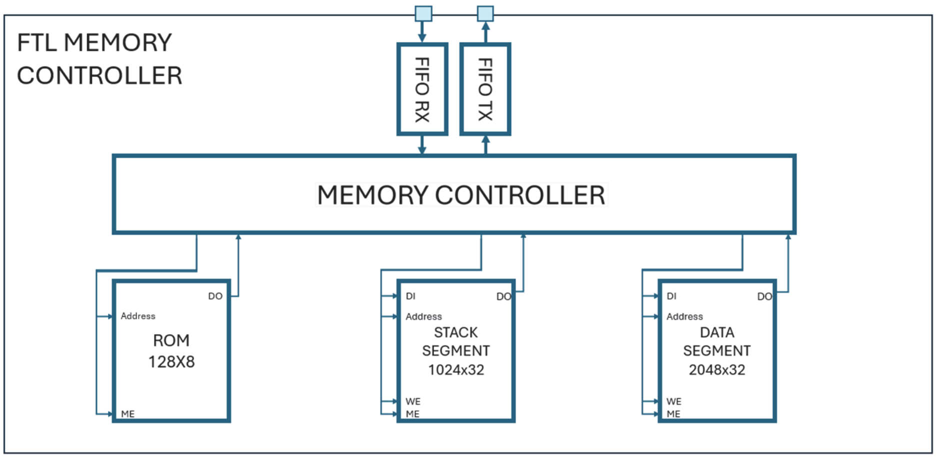

Fig. 2. Architecture of Memory controller with multi memories

UVM Testbench Architecture for FTL Memory Controller. The proposed testbench consists of an UVM environment and interface components to connect with the RX and TX FIFOs of the memory controller. Within the environment, an UVM Agent component is responsible for driving the verification process, which includes a UVM Driver that sends data packets to the RX FIFO, a UVM Sequencer that generates the sequences, and a UVM Monitor components that collects inputs and outputs from the RX and TX FIFO interfaces as shown in figure. These collected signals are then sent to the Scoreboard for comparison with expected data and to the Coverage component to evaluate the test coverage using UVM analysis ports.

The testbench is designed to generate random address and data sequences for comprehensive memory testing. It creates sequences to write data into and read data from memory, while also managing FIFO operations through push and pop signals. To ensure thorough validation of all aspects of the memory controller's functionality, multiple sequencers are utilized. These sequencers are responsible for managing the different types of memory operations and are started to manipulate the ROM and RAM memories. In Fig.3, a UVM-based testbench architecture of FTL memory controller is shown.

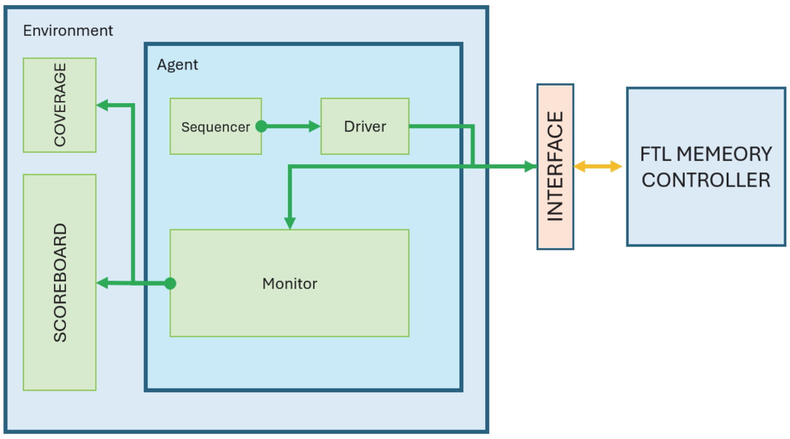

Fig. 3. UVM Environment Architecture of Memory Controller

For simulation different packets were created to validate all RX and TX FIFOs memory controller and sub memories. Using traditional methods to push WRITE_ADDRESS, WRITE_DATA and READ_ADDTESS packets to RX FIFO then pop read data from TX FIFO and compare with written data in scoreboard, we can achieve about 40% present of coverage:

$$read\_data_{addres(N)} = writte\_data_{addres(N)}.$$

To increase coverage by 20%, we used randomization to generate random address and push WRITE_ADDRESS, WRITE_DATA to RX FIFO multiple time to write all addresses in memories, then pushed READ_ADDRESS to RX FIFO with randomly generated addresses, then pop read the data from TX FIFO and compared with the written data kept in scoreboard. Using this techniques we can cover only about 60%. The remaining 40% are the cases where RX and TX FIFOS are full and the pushed data are in fifo queue:

$$read\_data_{addres(N)} = memory_{core[address]}.$$

The main challenge is to keep the written data in scoreboard for further comparison during read operation taking into account the number of the sent write/read packets to the DUT which can be data that can lead to overwritten of data in the same address. To do this the new mechanism is developed in scoreboard. The 2 dimensional SystemVerilog queue primitive was initiated with N+1 bit data width and M address length to keep the written data of the corresponding address in queue if the write packet has been sent or n+1 length of sequence of 1s if the read packet has been sent. The illustration of 2 dimensional SystemVerilog queue primitive is shown in Fig. 4.

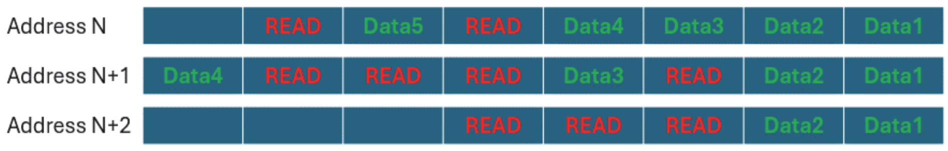

Fig. 4. Illustration of data packet keeping in scoreboard

$$read\_data_{addres(N)} = memory_{core[address][i]}, if\ i + 1 = \ READ.$$

When Pop is send to TX FIFO the poped data should be sent to the scoreboard where it will be compared with the last WRITTEN after the last READ data from the corresponding addres queue. The address is also sent to the scoreboard via systemverilog queue primitive.

The simulation was conducted in edaplayground.com platform with Synopsys VCS, leveraging various techniques tailored to different aspects of the test cases. Random test cases are used to explore a broad spectrum of input conditions and validate the robustness of the memory controller, while directed test cases focus on specific scenarios to verify critical functionality and edge cases. This mixed approach is ensured thorough validation of the memory controller's performance and accuracy across diverse operating conditions. The System Verilog code of design and UVM implementation could be found in edaplayground.com/x/GVWx link. The simulation results are shown in Fig. 5.

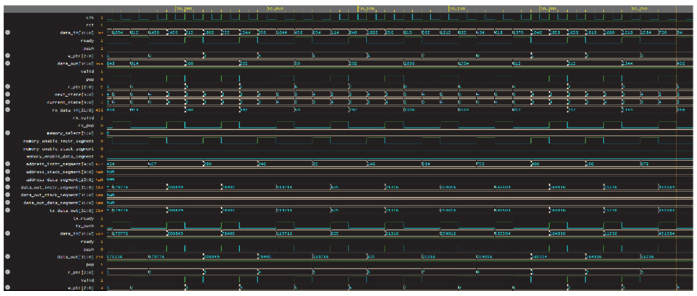

Fig. 5. Simulation Results

Table below shows a brief summary of features supported by ARMX generator comparing with other generators.

Table

Summary comparison of coverage and spend time of the used method

METHOD NAME	COVERAGE	TESTBANCH CREATION TIME
Direct Inputs	40%	VERY LONG
Randomization	60%	LONG
Purposed Method	100%	SHORT

Conclusion. Traditional verification methods often face challenges in addressing stringent timing constraints and achieving comprehensive functional coverage. In contrast, the proposed testbench effectively utilizes UVM's support for constraint randomization to accelerate testing and achieve high functional coverage in validation of FTL memory controller. By employing a combination of random and directed test cases, our methodology ensures, thorough validation, the FTL memory controller's functionality. Our results demonstrate that this approach not only meets verification requirements but also achieves 100% functional coverage for FTL memory controllers. The application of UVM randomization and its comprehensive framework significantly enhances both the reliability and efficiency of the verification process.

REFERENCES

1. Universal Verification Methodology (UVM) Class Reference Manual. - June 2014.
2. **Petrosyan O. and Manukyan A.** Functional Verification of Multiport SRAM Memories Based on UVM //2023 IEEE East-West Design & Test Symposium (EWDTS). – Batumi, Georgia, 2023.-P. 1-4.
3. **Melikyan V., Harutyunyan S., Kirakosyan A. and Kaplanyan T.** Uvm verification ip for axi // 2021 IEEE East-West Design & Test Symposium (EWDTS). - 2021.-P. 1-4.
4. Constructing Effective UVM Testbench by Using DRAM Memory Controllers / **R. Kabilan, R. Ravi, J.M. Esther, U. Muthuraman, et al** // 2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS).-Coimbatore, India, 2022. – P. 1034-1038.
5. **Khalifa K. and Salah K.** Implementation and verification of a generic universal memory controller based on UVM // 2015 10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS).- Napoli, Italy, 2015. P. 1-2.
6. Supradeep Narayana. On-chip communication hardware resources for globally asynchronous and locally synchronous systems // 8th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'05). - Las Vegas, NV, USA, 2005. - P. 6.
7. "Taming heterogeneity - the Ptolemy approach" / **Johan Eker, Jorn Janneck, Edward A. Lee, Jie Liu, et al //** Proceedings of the IEEE.- 91(1). - 2003. - P. 127-144.
8. **Nagy L., Koscelánsky J. and Stopjaková V.** Design of a globally asynchronous locally synchronous digital system // 2014 IEEE 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA).- Stary Smokovec, Slovakia, 2014. – P. 529-533.
9. Optimization of GALS CMP architecture with DCT as case study / **A.S. Menon, J.R. Gini, B. Aishwarya, C.C.G. Balaji, et al** // 2011 3rd International Conference on Electronics Computer Technology. - Kanyakumari, India, 2011. - P. 330-333.

National Polytechnical University of Armenia. "Synopsys Armenia" CJSC. The material is received on 07.10.2024.

Ա.Գ. ՄԱՆՈՒԿՅԱՆ

FTL ՀԻՇԱՍԱՐՔԻ ԿԱՐԳԱՎՈՐԻՉԻ ՆԱԽԱԳԾՈՒՄԸ ԵՎ ՖՈՒՆԿՑԻՈՆԱԼ ՍՏՈՒԳՈՒՄԸ UVM ՄԵԹՈԴԱԲԱՆՈՒԹՅԱՄԲ

Ներկայացվում են հերթային բուֆերացվող տարրերի փոխանցման մակարդակով (ՀԲՏՓՄ) նախագծման մեթոդաբանության վրա հիմնված հիշողության կարգավորիչի նախագծումը և իրականացումը, որը բարձրացնում է ռեգիստրների փոխանցման մակարդակից (ՌՓՄ) FIFO մոդուլների վրա հիմնված փոխանցման վերացականության մակարդակը, որտեղ գլոբալ հաորդակցությունը կատարվում է հերթային բուֆերացնող տարրերով, իսկ տեղական մոդուլներում՝ ՌՓՄ-ով: Այսպիսի համակարգերի ֆունկցիոնալ ստուգման համար անհրաժեշտ է հատուկ թեստավորման միջավայր, որտեղ հաշվարկված կլինեն տվյալների տեղափոխությունները հերթային բուֆերացնող մոդուլներով: Այդպիսի թեստավորման միջավայր ստեղծելու համար օգտագործվել է UVM մեթոդաբանությունը, որը հնարավորութիյուն է տալիս կատարել թվային սխեմաների արդիական պահաջներին համապատասխանող թեստավորում: Առաջարկված մեթոդը հնարավորություն է տալիս ստուգելու ՀԲՏՓՄ հիշողության կարգավորիչը՝ թեստավորման ծածկույթը հասցնելով 100%:

Առանցքային բառեր. Ֆունկցիոնալ ստուգում, UVM թեստավորման միջավայր, հիշողության կարգավորիչ, օպերացիոն հիշողություն, FIFO-FIFO փոխանցման մակարդակ:

А.Г. МАНУКЯН

ПРОЕКТИРОВАНИЕ И ВЕРИФИКАЦИЯ КОНТРОЛЛЕРА ПАМЯТИ FTL НА ОСНОВЕ UVM

Представлены проект и реализация контроллера памяти на основе методологии проектирования FTL (уровень передачи FIFO-to-FIFO), которая переводит уровень абстракции с RTL до семантики передачи значений транзакций FIFO-FIFO или смешанной модели, где глобальная связь осуществляется с помощью FIFO, а локальные модули могут быть реализованы с помощью абстракций более низкого уровня, таких как RTL. Хотя архитектуры FTL обеспечивают мощную структуру для обработки сложных транзакций, проверка этих конструкций создает значительные проблемы из-за их сложных схем связи и проблем своевременных транзакций. Для функциональной проверки таких систем необходима специальная тестовая среда, в которой будут рассчитываться передачи данных с помощью модулей FIFO. Для создания такой тестовой среды была использована методология UVM, которая позволяет выполнять тестирование, отвечающее современным требованиям цифровых схем. Предлагаемый метод позволяет тестировать контроллер памяти FTL, увеличивая покрытие до 100%.

Ключевые слова: функциональная верификация, тестовая среда UVM, контроллер памяти, оперативная память, FTL.