Design scheme for Emulator-friendly Memory Verification IP to Accelerate Simulation Performance

Sunghyeon Kang, Munsik Bae[1] Jinsung Song, Minsung Kang, Sangkyoo Jeong[2] Seokho Hong[3]

[1]({sunghyeon.kang, munsik.bae}@sk.com)
SK hynix Inc., Seongnam, Korea

[2]({jinsung.song, minsung.kang, Sangkyoo.Jeong}@sk.com)
SK hynix Inc., Seongnam, Korea

[3]david.hong@siemens.com
Siemens EDA, Seongnam, Korea

***Abstract*-As the design size increases, System on Chip (SoC) level simulation consumes a lot of Turn-Around-Time (TAT). One of the solutions to address this TAT consumption is to speed up verification through Co-emulation. However, while emulators can accelerate the Design Under Test (DUT), the non-accelerated testbench, especially the frequent communication over the memory interface between the Memory Verification Intellectual Property (VIP) and the design, acts as a bottleneck for simulation performance, thereby diminishing the effect of reducing TAT.**

To address this issue, we propose a method to accelerate the frequently communicated Memory VIP on an emulator. We redesigned the primary operations of the Memory that operated within the Universal Verification Methodology (UVM) Component into a synthesizable form that connects the DUT and the testbench. Experimental results showed that the redesigned approach achieved approximately 27 times faster simulation performance compared to the simulation time before redesign. This approach significantly reduces the simulation bottleneck, effectively lowering the TAT and enhancing the overall verification efficiency of large SoC designs.

I. INTRODUCTION

As the complexity and size of logic in SoC design verification continue to increase, the verification process has become progressively more intricate. One solution to improve the quality and efficiency of verification is to utilize Co-emulation with an emulator, which can enhance the speed of verification. However, when directly using the testbench employed in simulation within this context, it can sometimes be challenging to achieve satisfactory simulation run performance.

This paper presents a method for redesigning memory VIP, which involves data transmission, to be more emulator-friendly. This study examines the bottlenecks inherent to conventional Memory VIPs during UVM testing on an emulator. Through a redesign process, we have identified methods that not only allow for reuse in typical simulations but also maximize emulator performance.

The remainder of this paper is organized as follows: Section 2 introduces the methodology for performing UVM tests within an emulator and explains why the reuse of UVM Memory VIP from simulation environments fails to yield satisfactory performance in such an emulator setting. Section 3 presents the redesign methodology for the Memory VIP aimed at maximizing performance within the emulator. Section 4 provides experimental results that compare the simulation performance of the redesigned Memory VIP with that of the original. Finally, concluding remarks are given in Section 5.

II. ENHANCING UVM TEST PERFORMANCE: CAN WE BOOST EXECUTION IN EMULATOR?

A. Background - UVM test emulation

The basic principle of testing in an emulator involves accelerating a synthesizable DUT in the emulator, while not a synthesizable UVM testbench operates in a simulator. The testing is conducted through communication between the emulator and the simulator, a process known as Co-emulation [1]. To facilitate relatively easier testing of UVM testbenches in an emulator, Electronic Design Automation (EDA) companies provide various emulator options such as Veloce SvTbBringup (SystemVerilog Bring-up) mode [2].

[1]Sunghyeon Kang and Munsik Bae equally contributed to this paper.

B. *Challenges of Memory VIP in Co-emulation*

In Co-emulation, one of the most critical issues that make it challenging to maximize simulation performance when running UVM tests used in simulation is the frequent interaction between the accelerated DUT in the emulator and the non-accelerated Memory VIP in the simulator. These interactions can prevent the testbench from effectively capturing the performance benefits of the emulator [1]. As a result, although improved test performance can be achieved compared to simulation, it is difficult to attain satisfactory speeds when compared to verifying with actual Memory device boards connected to the emulator. This consequently makes it challenging to pre-validate the operation of the SoC before the actual Memory is developed.

C. *The Need for Customized VIP instead of Pre-packaged VIP Solutions from EDA companies*

However, EDA vendors who actively advocate for the Co-emulation methodology may already provide VIP suited for their emulators. Consequently, it is natural to pose the following question: 'Should we not simply use the VIP that incorporates the Co-emulation methodology provided by the IP vendor?' The answer to this question can vary—'yes' or 'no'—depending on the type of project. From the perspective of memory vendors, it is essential to develop next-generation memories and corresponding SoCs to quickly respond to industry trends. The specifications of these memories are subject to frequent changes, and each memory vendor may employ a variety of own special operations that are not defined in the specifications. For instance, NAND flash memory uses several vendor-specific commands shared with the Memory Controller to enhance performance, which are not disclosed for confidentiality reasons. While EDA companies can provide VIP solutions that are sufficiently accelerated on emulators, it is challenging for them to support these undisclosed special commands promptly. Therefore, there is a timely need for customized models that can incorporate new features, offer superior debugging capabilities, and flexibly respond to unstable new functionalities.

III. Key Idea - Emulator-Friendly Re-Modeled Memory VIP

The core principle of Co-emulation acceleration is to improve simulation performance by delegating the intrinsic roles of the UVM Driver/Monitor to synthesizable and be able to accelerate interfaces in the form of functions or tasks. Based on this fundamental concept, we propose a VIP design scheme that is tailored to the performance characteristics of memory operations. Fig. 1 depicts a schematic representation of the main idea of the Re-modeled Memory VIP and Interface proposed in this paper.

A. *Moving the main operation from the driver to the interface with an appropriately sized array*

As mentioned earlier in section 2, the biggest bottleneck of an emulator is frequent communication with non-accelerated regions. This leads to performance degradation in simulations, especially in cases of high-speed data transfers like NAND and DRAM.

To address this issue, you can reduce iterations by declaring an array of transactions such that the size of each transaction fits within the bandwidth permitted by the emulator per unit time when transmitting transactions between the emulator (Interface) and the simulator (VIP Driver). By considering the size of transactions that the memory specification can handle and the efficiency of the bandwidth between the emulator and simulator, you can declare an adequately sized array within the interface and design an FSM to control it (see Fig. 3).

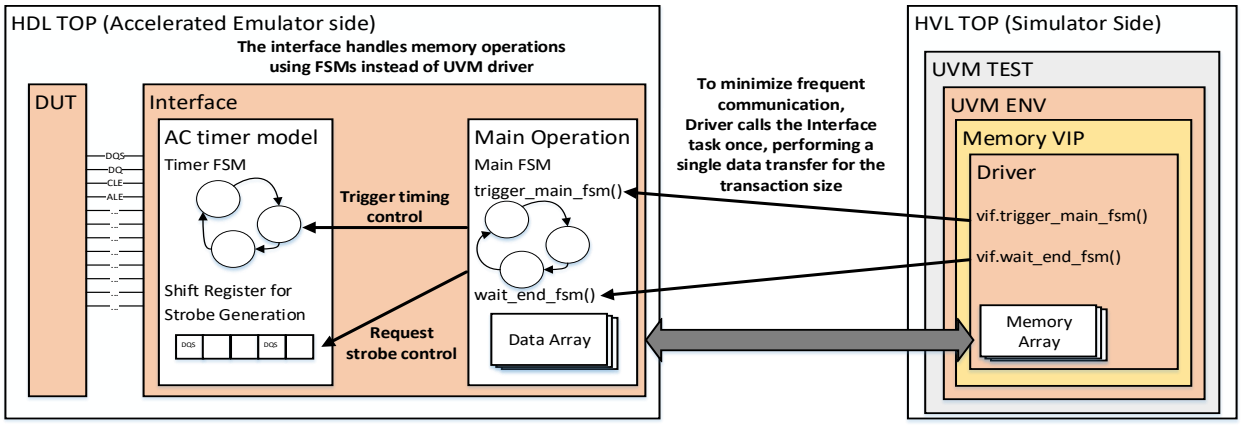

Figure 1. Key Idea of Design Scheme for Emulator-friendly Memory Verification IP

Fig. 2 illustrates the configuration of the Main Operation Finite State Machine (FSM). Detailed descriptions of each state within the Main Operation FSM are as follows:

IDLE: The system is in an initialized state with no ongoing operations. The Driver transitions the state to DRIVE by invoking *trigger_main_fsm()* on the Interface. In the case of a read operation, the Driver transfers its Data Array to the Interface. Subsequently, the Driver calls *wait_end_fsm()* via the virtual interface.

WAIT: In this state, the system awaits the Data Transfer process corresponding to Read/Write commands.

DRIVE: This state manages the Data Transfer between Interface and DUT. If a situation arises that necessitates halting the Data Transfer, such as encountering back-pressure, the system transitions to the PAUSE state. Upon completion of the Data Transfer, the system transitions to the EXIT state.

PAUSE: In this state, the system waits when the host, such as a memory controller, cannot receive data. This pause persists until data transmission becomes feasible again.

EXIT: All transmissions have been completed, or the transaction has been terminated due to a command abort or similar event. The system transitions to an IDLE state in the subsequent cycle. During this phase, the *wait_end_fsm()* task, called by the Driver through the virtual interface, completes. For write operations, upon the termination of *wait_end_fsm()*, the Driver acquires the Data Array from the interface.

Fig. 3 explains the data transfer mechanism governed by the Main Operation FSM. As previously mentioned, the key to accelerating simulation is to minimize the number of communications between Hardware Description Language (HDL) and Hardware Verification Language (HVL) side. The most straightforward method to achieve this reduction is to maximize the data transferred within the bandwidth limits permitted by the emulator. In the case of a write operation, all necessary data can be transmitted once at the EXIT stage. For a read operation, the data transfer occurs once at the IDLE state when the driver triggers the *trigger_main_fsm()* task. The data transfer operations between DUT, conforming to the memory specifications, are executed in the DRIVE state according to Alternating Current (AC) timing requirements.

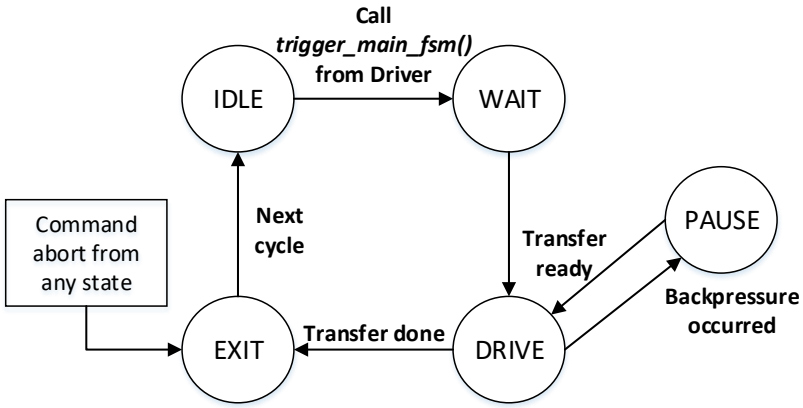

Figure 2. Main Operation FSM

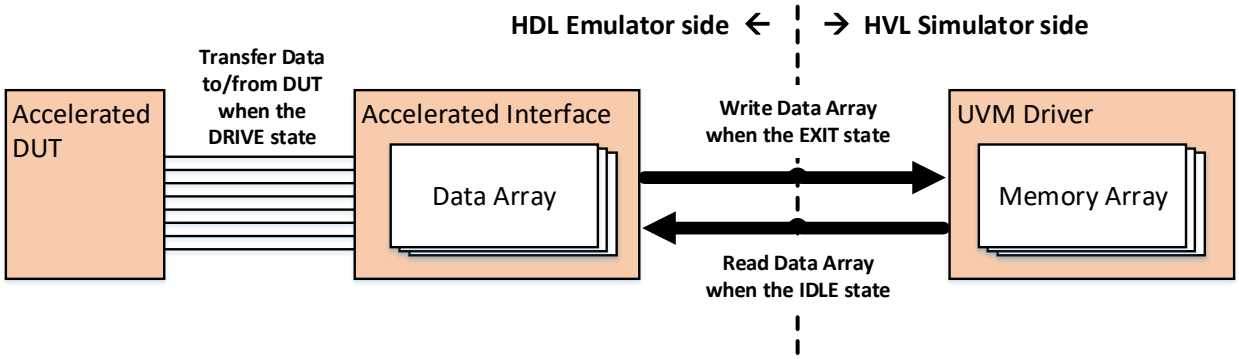

Figure 3. Data Transfer Controlled by Main Operation FSM

B. Overview: AC timing modeling based on clock cycles

Another principal strategy for Co-emulation optimization involves eliminating time-consuming regions in simulation and retaining only the untimed areas. Although, as mentioned in Section 2, it is possible to run tests even if there are time-consuming constructs in the testbench using options provided by emulators like Veloce SvTbBringup mode, one must tolerate a performance drop. This is because the time-consuming regions in the simulation are monitored as events by the emulator.

A key characteristic of memory operation is signaling that meets with the AC parameters defined by device specifications. Implementing this often requires time-consuming methods, typically realized through Verilog delay statements (#delay) in UVM drivers and monitors. We propose an alternative method by embedding two specific AC timing models directly into the interface. The first model features an AC timer FSM for AC parameters that are activated once when a memory command is issued. The second model uses a shift register to handle AC parameters that are continuously applied during data transmission, such as those associated with strobe signals.

C. AC Timer FSM

Fig. 4 illustrates the structure of the AC timer FSM. In all states of the Main Operation FSM, except for IDLE, AC timing corresponding to each state can be defined, thereby allowing the AC timer FSM to be triggered. After the Main FSM sets the desired AC timing and waits for it to elapse, it informs the Main Operation FSM that the delay has ended. This notification serves as a fundamental condition for transitioning to the next state within the Main Operation FSM.

The description of each state of the AC timer FSM is as follows:

IDLE: This is the initialization stage. The FSM can transition to the SET state through a trigger from the Main Operation FSM.

SET: Upon receiving the AC timing setting from the Main FSM, this state updates the counter to wait for the specified delay.

COUNT: In this state, the FSM counts according to the specified AC timing.

CHECK: This state checks for the presence of a linked AC timing. By referencing the AC timing list (see Fig. 5), if a linked AC timing exists, the FSM transitions to the SET state to wait for the new AC timing. If there are no further linked AC timings, the FSM transitions to the DONE state.

DONE: This state notifies the Main Operation FSM that the AC timing has been awaited. In the next cycle, the FSM transitions back to the IDLE state.

Fig. 5 illustrates a simplified example aimed at enhancing understanding by depicting the management method for the AC timing list required for NAND Read Data transfer. Initially, the necessary timings for the read transfer in each state of the Main Operation FSM—specifically, tDQSQ (DQS-DQ skew; DQS to last DQ valid, per access), tQH (DQ-DQS hold; DQS to first DQ becoming non-valid, per access), and tCHz (CE_n high to output Hi-Z)—are sequentially registered in the AC timing list. In the SET state, the system is configured to count the first parameter, tDQSQ. Once the counting of tDQSQ is completed in the COUNT state, tDQSQ is removed from the AC timing list. Subsequently, in the CHECK state, tQH is checked, and the system transitions back to the SET state to await the next timing, repeating this process iteratively.

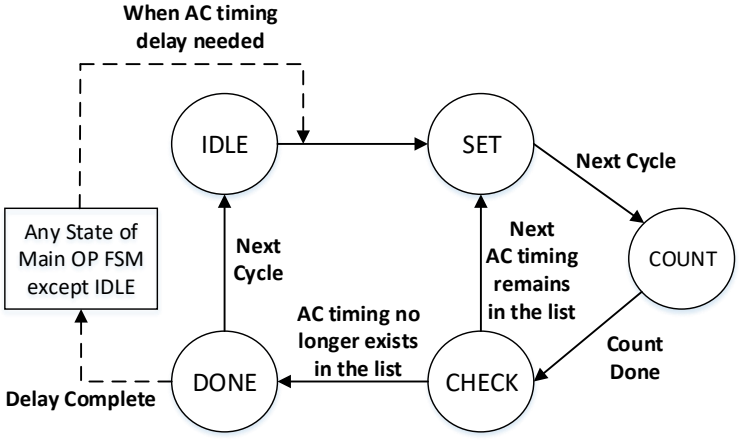

Figure 4. AC Timing FSM

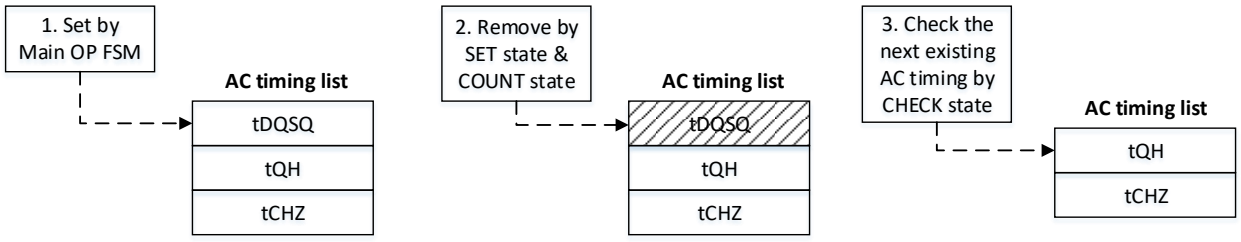

Figure 5. Example of Managing the AC Timing List

D. Shift Register for Strobe Generation

In a series of read operations, some AC timers are used on a one-time basis, while other AC timings, such as those related to the DQS signal, need to be applied every time the signal toggles. For instance, according to the JEDEC-defined NAND specification, the DQ strobe signal must be toggled after a certain interval in response to the Read Enable (RE) signal. This is defined in the NAND Specification as tDQSRE (Data output RE_n to DQS latency) [3]. Such signals must be applied to every bit of data transmitted within a single command. This specific AC timing delay was implemented using a simple shift register rather than a FSM.

Fig. 6 illustrates the operational principle of a shift register designed for signals such as the Strobe signal. By shifting 1 bit per cycle, the signal is toggled upon reaching the desired target. To better understand this, consider the tDQSRE example previously discussed. Initially, the register point correlating with the tDQSRE interval is set as the target. Upon the occurrence of the RE signal from the host, it is sampled by the emulator clock and recorded in the first bit of the shift register. When the shifted RE signal reaches the predefined target, the DQS signal is generated. This shifting takes place every cycle, and the described process is repeated with each occurrence of the RE signal.

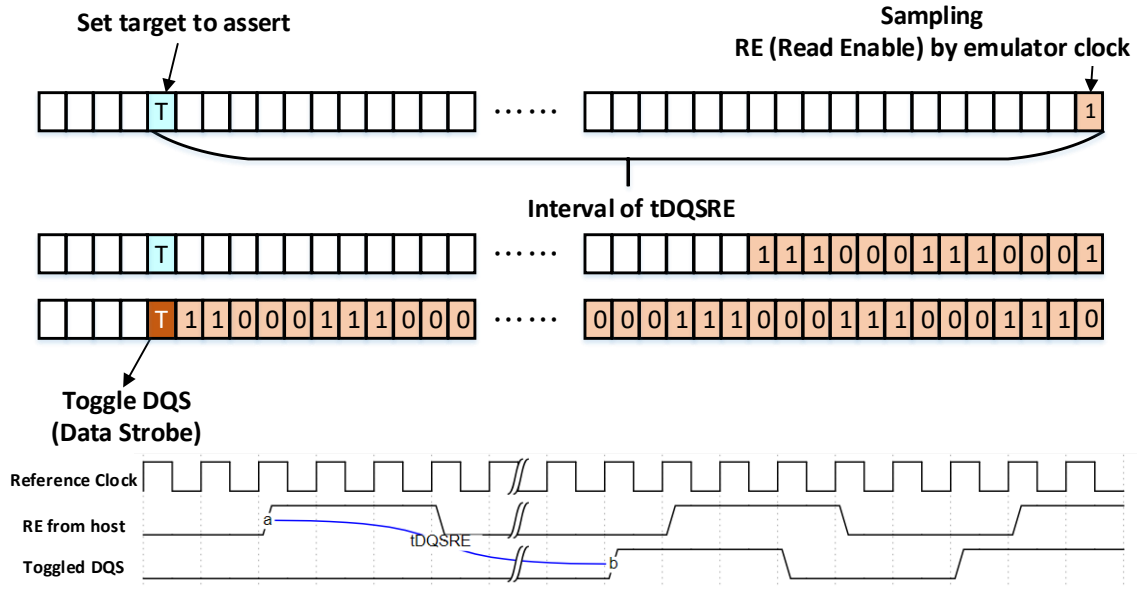

Figure 6. Example of Strobe Generation by Shift Register

IV. EXPERIMENT

A. *Experimental Conditions*

To measure the simulation performance of the test, two metrics were used: 1) the simulation time elapsed per hour in the period when the memory operation peaks, and 2) the data transfer rate between the NAND Controller SoC and the NAND using IO meter, measured per second.

Table 1 shows a description of the experimental setup. The emulator used in the experiment is the Veloce StratoM from Siemens, with Veloce OS version v23.0.2. The simulation on Veloce was conducted using the Questa simulator, version 2023.3. For the DUT, which is a Peripheral Component Interconnect Express (PCIe) Solid State Drive (SSD) controller, the VirtuaLab PCIe version 1.9.0 was utilized to function as the host interface solution.

The test scenario is as follows: For one of the basic operational scenarios of NAND performance—Sequential Write-Read operations—comparisons were made among 1) Conventional NAND VIP and 2) a Re-modeled scheme applied to the NAND VIP. In an additional experiment, we connected a physical NAND device board, which had already been developed, to an external port of the emulator instead of using a VIP. The purpose of this experiment was to measure the performance under the same scenario. The reason behind using a physical NAND device for this experiment was that it is considered the most accurate indicator of maximum test performance when evaluating the design with an emulator.

TABLE I
EXPERIMENT DETAIL

Component	Description
Design Under Test	PCIe 4.0 SSD controller SoC
Emulator	Veloce StratoM
SW version - Emulator OS	Veloce v23.0.2
Simulator Version	Questa 2023.3
Host Interface	VirtuaLab PCIe 1.9.0
NAND device topology	4 Channels, 4 Chip Enables, 2 Logical Units

B. *Experimental Result*

Based on the experimental results presented in Fig. 7, the conventional NAND VIP completed 7.38 milliseconds of simulation per hour, whereas the redesigned NAND VIP achieved a throughput of 204 milliseconds per hour, thereby demonstrating an approximately 27-fold improvement in simulation performance. The IO meter measurement also exhibited a similar trend to the observed simulation performance. When compared to the performance of an already developed physical NAND, the redesigned NAND VIP attained approximately 67% of the actual NAND's performance. Considering the advantages of debug capabilities such as Checkers and Debug signals inherent to VIP, which are difficult to quantify numerically, it can be expected that during the initial development of SoCs associated with the latest memory specifications described in Section 2, rapid verification can be achieved using VIP instead of prototype memory, which has yet to be manufactured.

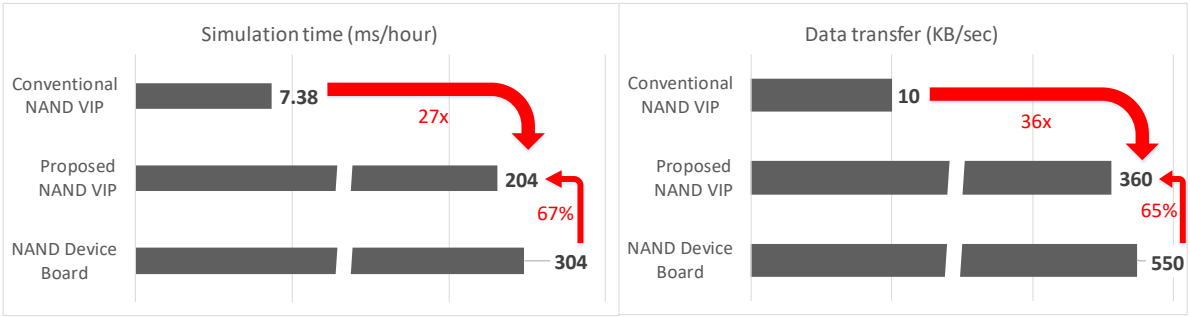

Figure 7. Experimental Result of Simulation Performance

V. Conclusion

This paper presents a design approach to accelerate the emulation performance of SoCs requiring frequent data interaction verification, such as NAND controllers. To achieve this, we propose the declaration of appropriate data types to minimize repetitive communication between the emulator and the simulator, and a FSM that can operate on interfaces capable of acceleration on the emulator. Additionally, to minimize the time-consuming areas on the test side, we rewrote the AC timer function using synthesizable syntax. Experimental results demonstrated that the redesigned approach achieved 27 times faster simulation performance compared to the VIP prior to redesign. This suggests the potential for efficient early verification of memory controller functions.

References

[1] Universal Verification Methodology (UVM) Cookbook, Siemens.
[2] Veloce Questa Flow User Guide v23.0.2, Siemens.
[3] JEDEC Standard NAND Flash Interface Interoperability JESD230G, JEDEC Solid State Technology Association.