A UNIVERSAL VERIFICATION METHODOLOGY FOR AN LPDDR3 MEMORY

WILMER DANIEL RAMIREZ VERA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FISICO-MECÁNICAS
ESCUELA DE INGENIERIA ELECTRICA, ELECTRONICA Y TELECOMUNICACIONES
BUCARAMANGA
2016

A UNIVERSAL VERIFICATION METHODOLOGY FOR AN LPDDR3 MEMORY

WILMER DANIEL RAMIREZ VERA

Trabajo de grado para optar al título de Ingeniero Electrónico

Director
ELKIM FELIPE ROA FUENTES
Ingeniero Electricista, Ph.D

Co-Director
HECTOR IVAN GOMEZ ORTIZ
Ingeniero Electrónico, MSc.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FISICO-MECÁNICAS
ESCUELA DE INGENIERIA ELECTRICA, ELECTRONICA Y TELECOMUNICACIONES
BUCARAMANGA
2016

AGRADECIMIENTOS

A Dios, por darme la salud y sabiduría para lograr culminar mi carrera.

A mis padres, Juan y Carmen por apoyarme incondicionalmente durante mis estudios y formarme como persona.

A mis hermanas, Lorena y Sarith, por su ayuda, motivación y apoyo en los tiempos difíciles.

A mis tíos, Ricardo y Teresa, por su apoyo durante toda mi formación universitaria.

A mis asesores durante el trabajo de grado, Elkim Roa y Héctor Gómez por sus consejos y sugerencias.

A mis amigos y compañeros, que siempre me han prestado ayuda y consejo cuando los necesito.

A mis profesores, que con sus enseñanzas y consejos me formaron como profesional y me han ayudado a seguir adelante.

Wilmer Daniel Ramírez Vera

CONTENTS

Page.

LIST OF FIGURES

LIST OF TABLES

RESUMEN

Título Una metodología de verificación universal para una LPDDR3 memory[*]

Autores Wilmer Daniel Ramírez Vera[**]

Palabras clave Microelectrónica, LPDDR3, UVM, SoC verification.

DESCRIPCIÓN

Este proyecto presenta un sistema de verificación digital que utiliza la Metodología Universal de Verificación (UVM) para una memoria síncrona y dinámica de acceso aleatorio (SDRAM) de 32 bits con una velocidad de reloj 800 MHz que varía en tamaños de 4, 6 u 8 Gb. Aunque la industria ha estado adoptando rápidamente a UVM como metodología de verificación para sistemas sobre chips (SoCs), la literatura académica carece de ejemplos detallados de la arquitectura de una verificación de propiedad intelectual (VIP) basada en UVM. El trabajo propuesto presenta arquitectura y operaciones no encontradas en la literatura para la verificación de una memoria LPDDR3. El sistema de verificación que se presenta comprueba la transferencia de datos de una SDRAM de alta velocidad descrita en Verilog, mediante técnicas aleatorias de generación de estímulos, herramientas de cobertura guiada y análisis automático utilizando UVM-SystemVerilog. El sistema de verificación propuesto se describe utilizando características de programación orientada a objetos (OOP), el modelado de nivel de transacción (TLM) y SystemVerilog. Además, dentro del sistema propuesto, se presenta una interfaz de memoria adaptada para funcionar como una interfaz virtual que cumple el protocolo de comunicación de la LPDDR3 y permite una comunicación estable con la memoria. El VIP basado en UVM implementado presenta características de código reutilizables y modulares que se pueden adaptar para una futura implementación de un VIP para una memoria LPDDR4. Los resultados presentados se obtuvieron a partir de pruebas realizadas en una LPDDR3 de 4GB.

[*] Trabajo de grado

[**] Facultad de ingenierías Físico-mecánicas. Escuela de Ingeniería Electrica, Electronica y Telecomunicaciones. Director: Elkim Felipe Roa Fuentes. Co-Director: Hector Ivan Gomez Ortiz

ABSTRACT

Title A Universal Verification Methodology for an LPDDR3 memory*

Authors Wilmer Daniel Ramirez Vera**.

Keywords Microelectronics, LPDDR3, UVM, SoC verification.

DESCRIPTION

This project presents a digital verification system using the Universal Verification Methodology (UVM) for a 32- bit LPDDR3 Synchronous Dynamic Random-Access Memory (SDRAM) clocked at 800MHz that ranges in sizes of 4, 6, or 8 Gb. Although, industry has been rapidly adopting UVM as a verification methodology for System-on-chip (SoC), academic literature lacks detailed examples of the architecture of a UVM-based Verification Intellectual Property (VIP). The proposed work presents architecture and operations not found in the literature for the verification of an LPDDR3 memory. The verification system to be presented checks the data transfer of a high-speed SDRAM described in Verilog, by random stimulus generation techniques, coverage-driven and automate analysis tools written using UVM-SystemVerilog. The proposed verification system is described using objects oriented programming (OOP), transaction-level modeling (TLM), and SystemVerilog features. In addition, within the proposed system, a memory interface adapted to work as a virtual interface that complies the LPDDR3 handshaking and allows a stable communication with the memory is presented, showing its architecture and operation. The implemented UVM-based VIP presents reusable and modular code features that can be adapted for a future LPDDR4 VIP implementation. The presented results were obtained from tests performed on a 4 GB LPDDR3 version.

* Bachelor degree

** Faculty of Physico-Mechanical Engineering. School of Electronics and Electrical engineering and telecommunications. Advisor: Elkim Felipe Roa Fuentes. Co-Advisor: Hector Ivan Gomez Ortiz

INTRODUCCION

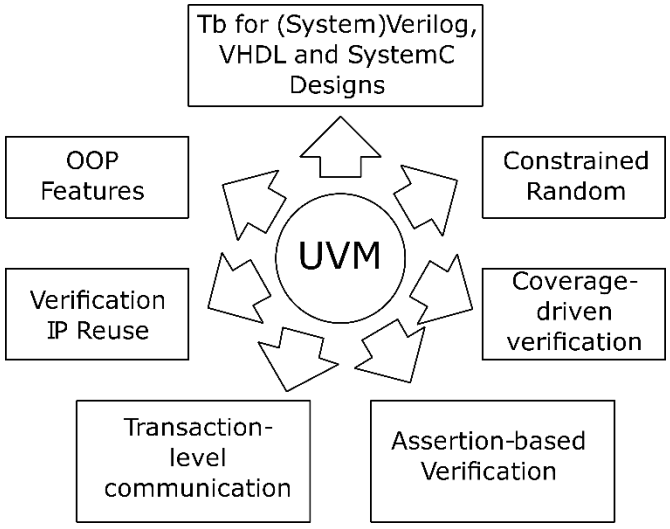
The electronics industry has had a strong hit in the society because of the facilities and innovation that it represents in the people's daily life. A big part of this technological innovation is due to the embedded systems which are present in the most of the devices, from a cell phone to the most advanced supercomputer. Most of these systems are designed using hardware description languages (HDL) [1], which are tools to create and verify the designs before sending to manufacture. Verilog is one of the most popular HDLs which can model digital systems and support many modeling and verification requirements of the industry and has been an indispensable HDL for designers [2]. Although Verilog is very useful to create digital designs, it is a poor language to verify big systems in an efficient manner. To support the complexities of SoC designs, SystemVerilog was developed combining HDLs and hardware verification languages such as Vera and e, and programming languages such as C and C++. Hence, SystemVerilog is considered the first Hardware Description and Verification Language (HDVL) [3]. Although SystemVerilog is a strong HDVL, designers are forced to make a different verification system as the designs grow and become more complex owing to an incapacity of reusing code. To solve this problem, standard verification methodologies were created to design reusable verification environments and supply the requirements of the industry [4]. Currently, there is a methodology that joins SystemVerilog tools and the best features of previous verification methodologies such as the Verification Methodology Manual(VMM), the first implemented set of practices reusable verification environments in SystemVerilog, and the Open Verification Methodology(OVM), an objects library that provides many facilities for constructing testbenches, both methodologies, and other ones not mentioned helped to create one of the best verification tools currently: the Universal Verification Methodology (UVM) [4]. This report presents a study of the

Universal Verification Methodology to be used as a digital circuits verification tool. Moreover, a partial functionality verification of a Low-Power Double Data Rate 3 (LPDDR3) memory is presented. This memory will be used in a research project with the University of Cambridge.

1. THE UNIVERSAL VERIFICATION METHODOLOGY

SoC designs verification is a necessary process to be sure of the proper behavior of the design before sending to manufacture. For widespread and complex designs, SystemVerilog has strong tools such as constrained random techniques for stimulus generation that helps in finding corner case problem making the testing process easier, and assertion-based verifi- cation that helps to detect functional bugs earlier and closer to their original cause. Also, SystemVerilog has coverage-driven verification to be sure that all possible cases of operation are performed, so we can be sure that the testing process is successful. Since the Universal Verification Methodology is a SystemVerilog library class, all the above SystemVerilog tools are also part of UVM. UVM solves the incapacity of reusing code abovementioned due to Object-oriented programming (OOP) and Transaction-level modeling (TLM) features were added to SystemVerilog to create better and reusable verification systems. The OOP adds the class creation that lets to pass features and methods to objects through the testbench. When an object associated with a particular class is created, this object inherits all data members and functions of the class, this is an advantage because an object can take the features of the class in any part of the testbench and thus multiple objects can be created wherever without copying code to other parts of the testbench, so we avoid the hardcode creation, one of the most significant problems for code reuse [5]. On the other hand, TLM is a high-level approach to modeling digital systems that separates the communication among modules from the implementation of functional units and the architecture. Therefore, adding TLM features to the testbench helps to create reusable and maintainability code due to each module has specific functions which are different to the other module functions, so different components of the system can be reused in future works. The *Figure 1* shows a summary of the UVM features that have been mentioned.

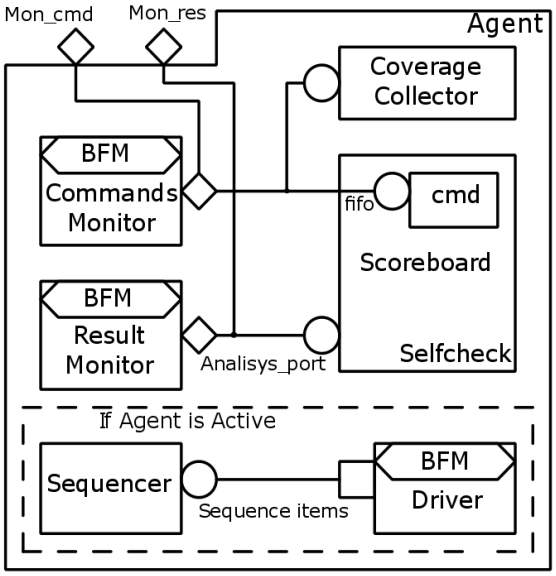
Figure 1 Verifications features and tools of UVM.



The UVM testbench structure is based on a hierarchy of modules (one within another) and each module has a different function. The best way to create reusable and maintainable code is that each module performs a single function in the testbench. Then, not only many parts of the code can be reused for futures designs but also finding an error is easier due to the designer can know what function is failing and consequently, what module has the error [5]. Modules, classes, and objects work together thus: The virtual interface, also called Bus Functional Model (BFM), is a module written using SystemVerilog which is connected to the input and output ports of the Device Under Test (DUT) working like a communication bridge between the DUT and the package of all UVM classes. Therefore, a handle of the BFM is passed through certain UVM components that need to establish communication with the DUT due to this interface controls the sent and the received data to and from the DUT. The virtual interface communicates with the Agents which are the verification components that handle the sending and receiving of data. These components are shaped by analysis components as coverage collectors and scoreboards (called self-checkers too), components that monitor the activity from/to the DUT, and finally, if the agent is the active type then the testbench has a stimulus maker component called

sequencer, and a component that is responsible for carrying the sequence item to the DUT, this last one is called driver. In the Figure 2 is shown the agent and its interconnecting subcomponents, this figure shows that only monitors and the driver communicates with the BFM.

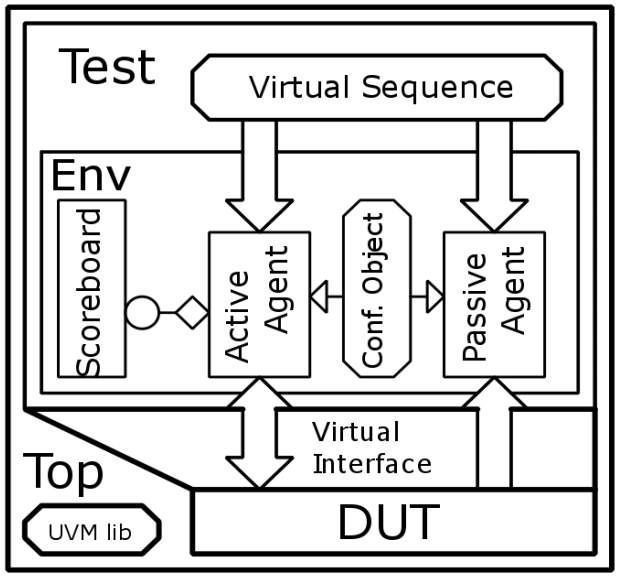
Figure 2 Verification components of a UVM Agent.



The analysis ports and FIFOs are tools of UVM that let pass transactions among components. In the next level of the hierarchy, we find the environment which not only creates and encapsulates one or multiple agents but also is responsible for configuring each one of them as an active or passive agent and other possible configurations. This configuration process is performed passing to the agent, the configuration objects which contains the agent features that are used to create it, the environment also can have analysis components such as scoreboards. The environment is encapsulated in the test that configures the number of virtual sequences to know how many agents will be created. The test special function is to select the sequences of stimulus and the order to be sent to the DUT passing this selection to the sequencer which is in the agent. Finally, the top-level module

encapsulates the DUT, the virtual interface, and connects them, also the top-level selects the test to be executed. In this module, the package that contains all the UVM libraries to run the testbench and the UVM class package are called. The top level is shown in Figure 3.

Figure 3 Hierarchy of classes in a UVM testbench.



2. UVM TESTBENCH FOR AN LPDDR3 MEMORY

LPDDR3 Memories are dynamic RAMs that work at twice the clock frequency due to this type of memory uses both clock edges to transfer data. The verified memory in this project is a 32-bit mobile low-power DDR3 in 4 Gb, 6 Gb, and 8 Gb synchronous dynamic random-access memory versions and internally configured as an 8-bank DRAM. The LPDDR3 to be verified has an 800MHz frequency clock, so the memory can transfer data at 1.6GHz. The LPDDR3 operates according to behavior of the signals shown in the Table 1[6].

Table 1 LPDDR3 signals description

Symbol	Type	Description
CA [9:0]	Input	Provide the memory's command and address
ck, ck n	Input	Differential clock. Both clock edges sampled the command/address inputs.
CKE	Input	The clock enable actives and deactivates the input clock signals and buffers.
cs n	Input	Chip select, part of the command code.
DM[3:0]	Input	4-bit Data mask for each of the four data bytes respectively in a writing process.
ODT	Input	On-die termination enables and disables termination on the DQ bus.
DQ	I/O	Bidirectional data bus
DQS, DQS n	I/O	The Data strobe is a bidirectional and complementary clock used for read and write data. 4-bit data strobe for each of the four data bytes, respectively.

The above signals must be sent to the LPDDR3 in the correct state, order and in different intervals of time, to perform a particular operation mode. The Table 2 shows some of the most significant LPDDR3 operation modes.

Table 2 LPDDR3 Operation Modes [6]

Operation Mode	Characteristics
Active	This mode actives a selected memory bank in order to perform an operation on this bank.
Writing & Reading	The burst write and burst read operations are performed in the respective operation mode. To access to any of these modes, first the LPDDR3 must pass by the Active mode.
Precharge	The clock enable actives and deactivates the input clock signals and buffers.
MR Writing & MR Reading	These modes are used to write and read respectively configurations data in the mode register
Power Down	This mode deactivates input and output buffers. Maximum duration in this mode is only limited by the refresh requirements. The LPDDR3 can pass to this mode from any other operation mode
Deep Power Down	This mode presents a low power function of this mobile SDRAM, allowing reducing the power consumption by shutting down the internal power supply and suspending refresh operations when the device is not accessed for a long period.
Self-Refresh	This mode is used to retain data in the LPDDR3 overriding this data in the memory banks. This operation is performed even if the rest of the system is powered down
CA-Training	This mode is used to improve timing margins of the Command/Address bus.

This project is focused on the data transfer verification in the active mode of the LPDDR3 memory. Therefore, the operation modes used are Active, Writing, Reading and Precharge. The Figure 4 and Figure 5 show the writing and reading process, respectively. The first step to request any operation is to active a memory bank and indicate a row number through the signal CA sampled in both clock edges. Then, Nop commands must be issued during an interval of time that varies depending on the operation required, when the minimum time is met, write or read commands can be issued indicating the column number. Finally, the writing or

reading process starts an interval of time later, 8 words are transferred in both cases (writing or reading). The data strobe samples the data in a writing process and is in line with the data in a reading operation; in both cases, the clock signal samples the data strobe, only Nop commands are allowed in this process. Also, the DM signal controls the data writing, masking the data to be not written.

Figure 4 LPDDR3 writing process

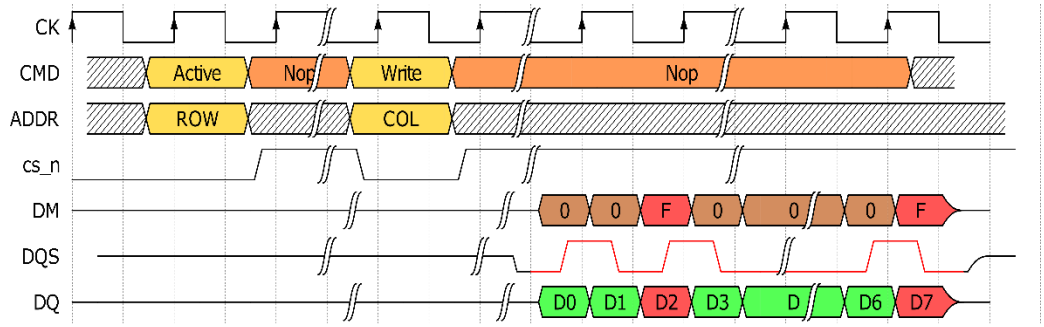
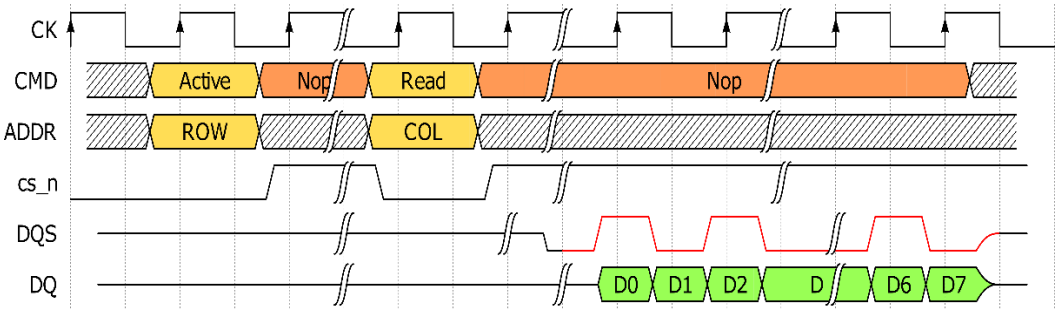


Figure 5 LPDDR3 Reading process



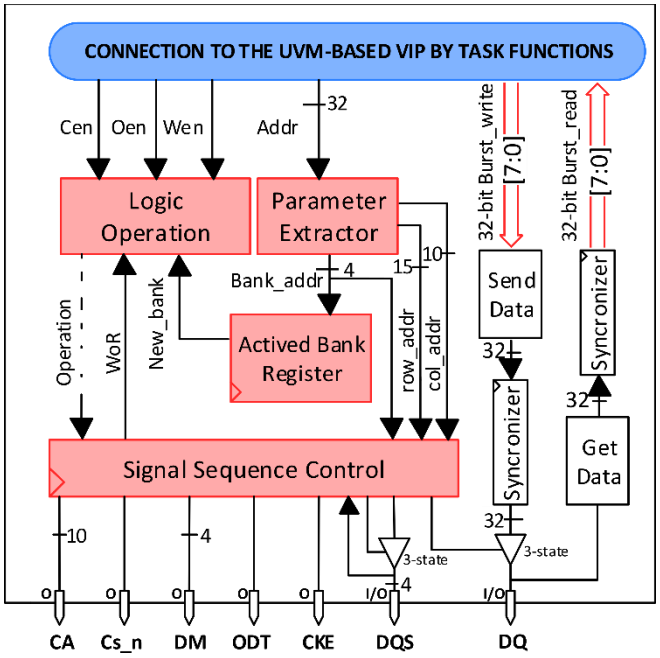
To check the correct behavior of the memory, a verification system using UVM is implemented. Next, the process to create the digital verification system is described.

A. Virtual Interface Description

The first step toward creating a verification system for the LPDDR3 is to describe the virtual interface or bus functional model to convert our stimulus to the memory

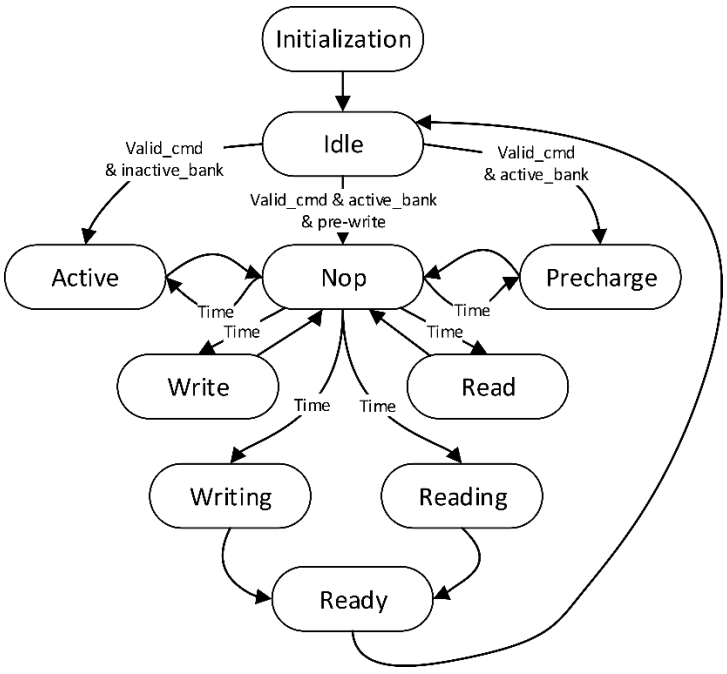
language. Hence, the BFM is a translator between the operations that we want the memory performs and the signals sequence that the LPDDR3 needs to perform a writing or reading process. The *Figure 6* shows a scheme of the virtual interface implemented in this project. There are three input signals that indicate to the BFM that a writing or reading operation is required, these signals are *cen*, *oen* and wen which pass through a logic process to recognize the operation type, just two combinations are valid; First, *cen* and *oen* has to be in high mode while *wen* stays in low mode to indicate to the BFM that a reading process is required, the contrary combination indicates a writing operation. In the logic process, feedback signals from a banks-process-control intervene too. LPDDR3 memories are configured in eight banks and each one has to be active before performing any writing or reading process. If a process is required in an already active bank, this bank has to be precharged. Furthermore, depending on the above process (write or read), the bank activation time varies.

Figure 6 LPDDR3 virtual interface diagram blocks



Therefore, a banks-control system is implemented into the virtual interface. For writing and reading process, in *Figure 6* two data burst are shown, each one is a 32-bit and 8-positions array for the LPDDR3 data burst. All stimulus are sent from the UVM testbench by task functions. After the operation is recognized, several signals sequences start to run to comply the LPDDR3 operations protocol, then certain changes in the state signals are needed as enabling and disabling output buffers. The Figure *7* presents the state machine implemented in the virtual interface to send all needed transactions to the LPDDR3 and perform continuous writing and reading operations. Concluding the description of the virtual interface, this interface is synchronized using two shifted clock signals, one signal clock is the same of the LPDDR3, and the other one is a clock signal shifted 312ps with the same frequency, this signal is used to change the command/address, data strobe and DQ buffers state in the correct time in such a way that setup and hold times of the LPDDR3 are complied with. Now that the system has a translator for the LPDDR3, the next step is to create the stimulus and check components.

Figure 7 Virtual interface state machine



B. Transactions, Sequences, and Generation of Stimulus

In this section, the UVM testbench elaboration starts, and the first step is the transaction creation. The term ”transaction” refers to classes and objects that store data and operations, and the proposed system uses two transactions. The first transaction is called ***lpddr3_sequence_item*** which contains all signals to be sent to the virtual interface; this transaction extends from a UVM library called *uvm_sequence_item* that is made for working especially with a sequencer. The second transaction is called ***read_transaction*** which contains only a 32-bit and 8-positions array to store the read data burst by the virtual interface in a reading process, this transaction also contains a basic operation called do compare whose function is to compare two objects of the same class, in this case, the read transaction class.

Next step, the building of stimulus is created in a special class which has been called ***write_&_read_sequence***, this class generates constrained random data that will be sent to the LPDDR3. The sequence described in this class is a configurable number of writing operations followed by reading operations with random address and data. The write & read sequence is only responsible for creating the sequences, so after creating the sequence to be used, the testbench needs a component that selects the sequence and puts this in another component that carries it. The first component is the ***lpddr3_test*** which selects the ***write_&_read_sequence***, and a lower component of ***lpddr3_test*** receives the sequence. The ***lpddr3_sequencer*** which is specially created to work with *uvm_sequence_items*, a UVM class library which is the parent class of the *lpddr3_sequence_item* that have been already created. The ***lpddr3_sequencer*** extends from the *uvm_sequencer*. This class library has a special communication port to carry data to the lpddr3 driver which is responsible for sending data to the virtual interface through a handle of this interface that is got calling the UVM function *uvm_config_db*. This function makes available the BFM through all the

testbench and got the handle of the interface for the component that calls the function.

Finally, the ***lpddr3_driver*** sends data using the task functions described in the virtual interface, this task functions handle all input and output data connections. So far, the verification system described only sends random stimulus but this still does not have any component that checks that the writing process was completed successfully. In other words, the testbench needs to check that the sent data to a certain address in a writing operation can be read subsequently in a reading operation for that same address.

C. Monitoring, Analysis, and Results

The verification system needs to add components that checks that all writing operations done can be verified by reading operations, the verification system implemented has two monitoring components, the first one is called ***lpddr3_command_monitor*** which observes the sent commands to the LPDDR3 such as the type of operation, the address and the data to be written. The second monitor is called ***lpddr3_data_monitor*** which observes the input data burst in a reading operation; both monitors are responsible for observing data and pass them into a transaction (a different transaction for each) to another component that handles all analysis process to print a result later, this last component in this system verification is called ***lpddr3_scoreboard***.

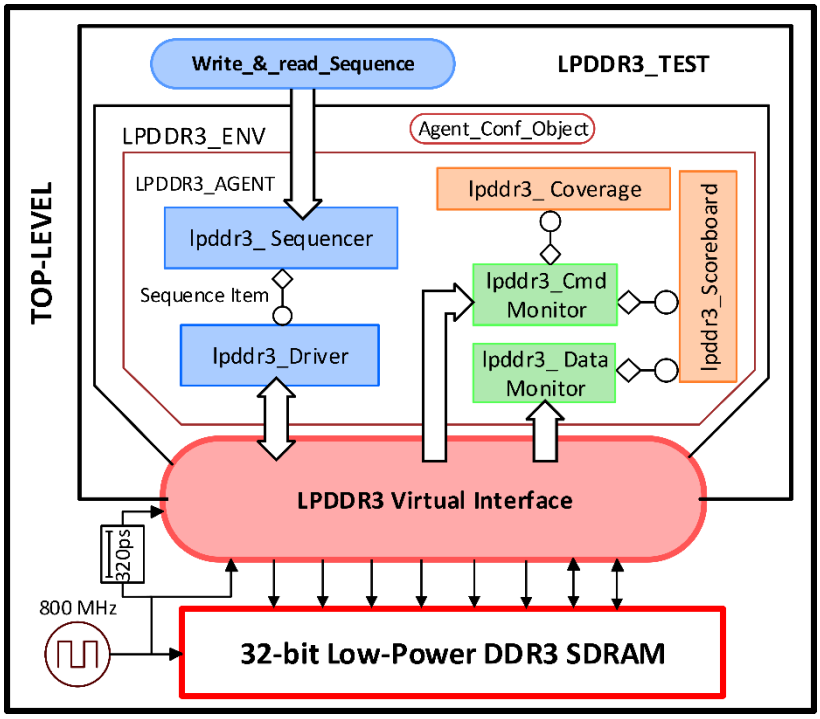
The connection between the scoreboard and the monitors is made through *uvm_analysis_port* and *uvm_tlm_analysis_fifo* that are TLM tools made especially to pass transactions to other components, the monitors put the transactions in these ports and the scoreboard extracts them to do its respective analysis. Finally, the ***lpddr3_scoreboard*** makes a comparison between the read burst data in an actual reading operation and the written burst data in a previous writing operation

for the same addresses. To do this comparison, the read burst data and the written burst data are saved in two different objects of *read_transaction* type which contain a 32-bit and 8-positions array, the verification system uses the do compare operation of the read transaction class to do the comparison between each of the eight pairs of 32-bit data and get a result. Then, the scoreboard prints the result of all comparisons.

In addition, the analysis process of the verification system has a component which must check the functional coverage in terms of stimulus. The lpddr3 coverage also receives an *lpddr3_sequence_item* using an Analysis port and uses covergroups to sample rows, columns, and banks that are used in all verification process. Hence, the system shows the percentage of all possible memory addresses that were verified.

To finish the verification system construction, all above components are created into an ***lpddr3_agent***. The proposed system presents eight agents that are created automatically to verify in parallel each one of the eight memory banks, and another agent more to verify operations (read and write) using all banks. Each agent is instanced in the ***lpddr3_env*** and configured through a configuration object that indicates which bank the agent has to verify. Finally, the top level module is built, the virtual interface described in SystemVerilog and the LPDDR3 described in Verilog are instanced in a SystemVerilog module called top, in this module the LPDDR3 and its interface are connected and the clock signals are defined for both. The package of classes that were created for the testbench is also imported in this module. The *Figure 8* shows the final verification system using UVM.

Figure 8 Blocks diagram of the LPDDR3 verification system using UVM.



3. RESULTS

The first step in the verification system creation process is to verify the virtual interface which was first designed as a communication interface for the LPDDR3 and later was adjusted to be a virtual interface in the verification system. To check that the communication interface was designed successfully, certain simulations were performed to get the results shown in *Figure 9* and *Figure 10*.

Figure 9 Writing operation simulation using the Interface.

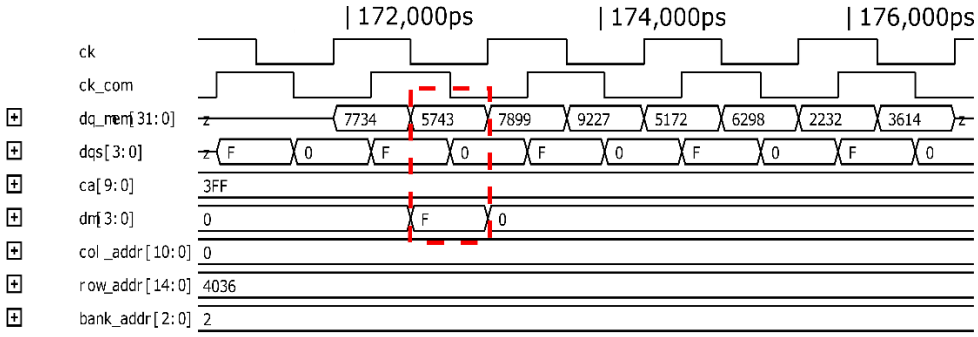
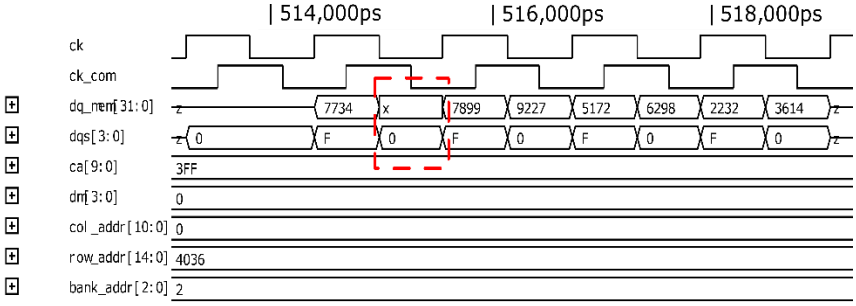


Figure 10 Reading operation simulation using the Interface



The simulations show successfully writing and reading operations respectively, showing initially that the virtual interface design is correct. Also, both simulations show that the data mask signal fulfills its function because the memory does not write the second data masked by the data mask signal in the writing process.

Therefore, the designed interface complies with the requirements for communication with the LPDDR3. The next step is to run the designed UVM testbench and check the LPDDR3. First, the sending of stimulus must be checked to know whether there is a connection or not from the UVM testbench to the LPDDR3.

The checking process made by the scoreboard is shown in the *Figure 11*, sent data and captured data are compared in order to the scoreboard shows a result, in this case, the result is the word CORRECT. Also, the scoreboard prints the written and the read data in a UVM INFO. In a different case, if the compared data are not the same, the scoreboard shows the word FAIL and prints a UVM ERROR.

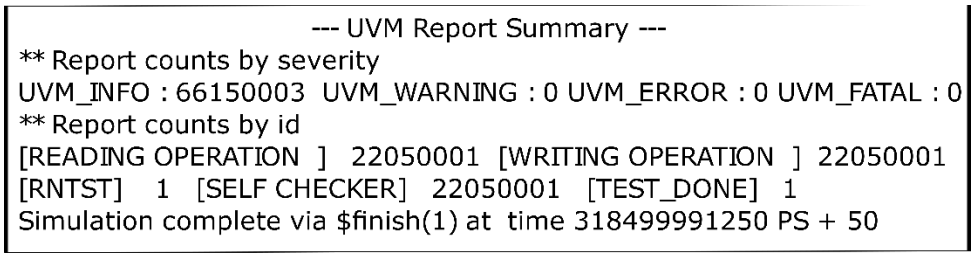
Figure 11 Single sentence of the Scoreboard Information

```
UVM_INFO testbench.sv(341) @ 12599.375000[ns]: uvm_test
_top.env_h.agent_b1.scoreboard_h [SELF CHECKER] [CORRECT]:
Address[Bank: 2 Row: 7724 Col: 917] ***Current Read Data:
Data_0: 37372635 Data_1:  72808459 Data_2:  45368420
Data_3: 47087597 Data_4: 92293077 Data_5:  19062527
Data_6: 19062527 Data_7: 97286636/*Written data in previous
operation:Data_0:  37372635 Data_1:   72808459 Data_2: 45368
420 Data_3:  47087597 Data_4:   92293077 Data_5:  19062527
Data_6: 19062527 Data_7: 97286636
```

The verification process was performed defining in the object ***write_&_read_sequence*** a number of writing operations followed by reading operations. For a 4Gb LPDDR3, the defined number was selected large enough to cover the total number of addresses for each memory bank (16.777.216 addresses). Ideally, the 2.097.152 writing-reading operations would be enough due to the burst data length (eight data), but the selected number also must cover possible repeated memory addresses. Performed tests using the random function showed about 14% of repeated numbers. Based on these tests, the UVM test was run configuring 2.450.000 writing operations, followed by reading operations. When the test is finished, the verification process result is shown in the command

terminal. In the *Figure 12* is shown the UVM report summary of the test which presents the total number of writing and reading operations, scoreboard reports (self-checker), printed information by components, warnings, errors, fatal errors, and the simulation time are shown.

Figure 12 Verification process result



Finally, when the verification process has ended, the last step is to check the total coverage that it had. Table 3 shows the obtained coverage results in the performed verification process. This results show 100% coverage for rows and columns in all banks but not for their intersections. Due to software limitations, the intersection of the total number of rows and columns in each memory bank can not be shown in the coverage results, since the total number of crossings is 16.777.216 per bank, and the software supports a maximum of 1 million.

Table 3 Coverage Results

Item		Coverage (%)
Parameter	Description	
Use of all Banks used in a sequence	8 used banks	100
Bank0 to Bank7	Columns	100
	Rows	100

4. CONCLUSIONS AND FUTURE WORK

This report presented a research work about the Universal Verification Methodology as a tool for digital circuits verification. The implemented verification system has a virtual interface complying the LPDDR3 handshaking, achieving a stable communication with the memory, and allowing to performing the data transfer between the designed UVM test and the LPDDR3 memory. Thereby, the proposed system using UVM verifies successfully LPDDR3 memories described in Verilog that range in sizes 4, 6 or 8Gb easily setting a number of sequences for the verification process and the rows and columns number depending on the memory size. Due to the capacity of reusing code of the Universal verification methodology, a big part of this verification system can be reused in future works with the LPDDR3 operation modes not worked in this project which are indispensable for the operation, and good memory performance. Also, the done code has characteristics of adaptability, whence parts of this code can be used in futures works with LPDDR4 and different type of designs.

REFERENCES

[1] R. J. Duckworth, “Embedded System Design with FPGA Using HDL (lessons learned and Pitfalls to be Avoided),” in 2005 IEEE International Conference on Microelectronic Systems Education (MSE’05), June 2005, pp. 35–36.

[2] S. Palnitkar, Verilog HDL - A Guide to Digital Design and Synthesis. Sun Microsystems Inc, 2003.

[3] “SystemVerilog Unified Hardware Design, Specification, and Verification Language,” IEC 62530 Edition 2.0 2011-05 IEEE Std 1800, 2011.

[4] J. Bromley, “If SystemVerilog is so Good, Why Do We Need the UVM? Sharing responsibilities between libraries and the core language,” in Specification Design Languages (FDL), 2013 Forum on, Sept 2013.

[5] R. Salemi, The UVM Primer. Boston Light Press, 2013.

[6] JEDEC, Low Power Double Data Rate 3. JEDEC Solid State Technology Association, 2015.

BIBLIOGRAPHY

J. Bromley, “If SystemVerilog is so Good, Why Do We Need the UVM? Sharing responsibilities between libraries and the core language,” in Specification Design Languages (FDL), 2013 Forum on, Sept 2013.

JEDEC, Low Power Double Data Rate 3. JEDEC Solid State Technology Association, 2015.

R. J. Duckworth, “Embedded System Design with FPGA Using HDL (lessons learned and Pitfalls to be Avoided),” in 2005 IEEE International Conference on Microelectronic Systems Education (MSE’05), June 2005, pp. 35–36.

R. Salemi, The UVM Primer. Boston Light Press, 2013.

S. Palnitkar, Verilog HDL - A Guide to Digital Design and Synthesis. Sun Microsystems Inc, 2003.

SystemVerilog Unified Hardware Design, Specification, and Verification Language,” IEC 62530 Edition 2.0 2011-05 IEEE Std 1800, 2011.