Development of a Generic and a Reconfigurable UVM-Based Verification Environment for SoC Buses

Alaa Hussien[1], Samar Mohamed[1], Mohamed Soliman[1], Hager Mostafa[1], Khaled Salah[2], Mohamed Dessouky[2], Hassan Mostafa[3]
[1]Department of electronics and communications, faculty of engineering, Ain Shams University, Cairo, Egypt.
[2]Mentor Graphics, Cairo, Egypt.
[3]Electronics and Communications Engineering Department, Cairo University, Giza
Email: Khaled_mohamed@mentor.com, mohamed_dessouky@mentor.com, hmostafa@uwaterloo.ca

Abstract – The similarities between SoC buses depends partially but not totally on domain. Generic universal verification methodology (UVM) architectures can be used to reduce effort and time to market. Generic UVM allows focusing on test cases rather than building the UVM. Although there are common features between SoC buses, but some properties and test cases must be customized. This paper presents a generic and reusable verification environment for SoC buses to accelerate verification process. To evaluate the efficiency of the proposed methodology, we apply it to three different SoC buses. The results are very promising in terms of high reusability and reducing of verification time.

KEYWORDS - Universal Verification Methodology (UVM), SoC buses, Bus Functional Model (BFM), Generic, Unified, Reuse.

I. INTRODUCTION

SoC buses are vital components in any SoC. Due to rapidly increasing operation frequencies, the performance of the SoC design heavily depends upon the efficiency of its bus structure [1].

ASIC/SoC verification is one of the most important tasks in digital design world. A fact tells that 60 to 70 % of total design time is consumed by verification only. Different companies adopt different verification methodology till universal verification methodology (UVM) comes into the picture, which is the best solution to overcome most of the drawbacks reported by the previously used methodologies [2].

Reusable verification environment is required to reduce verification efforts. The idea is nothing but "plug and play" for DUT/DUV with some minor changes in the testing environment with each new protocol.

"Reuse" is a term that is frequently associated with verification productivity. When a verification environment is needed for a new design, or for a design revision with significant changes, it is important to highly reuse what you have. In our previous work, we presented generic UVM for DRAM and flash-based memory controllers [3]-[4].

In this work, we present a generic and reusable verification environment for SoC buses. The proposed methodology makes use of the common features between different SoC buses to build generic UVM components. The proposed methodology is applied to some SoC buses such as AMBA APB, AMBA AHB and Avalon.

The rest of this paper is organized as follows: Section II presents the followed methodology including a flowchart of the steps taken. Section III, as a central part of this paper, gives an overview of the implementation methodology. In Section IV, for proofing of concept purpose, waveforms are attached indicating the success of the claimed idea. Concluding remarks and future plans are given in Section V.

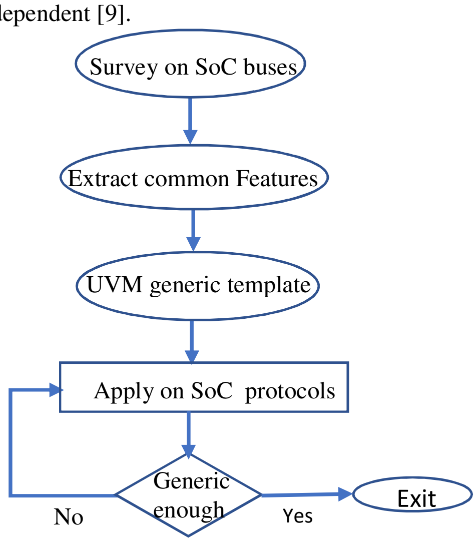
II. METHODOLOGY

The current sections discuss the followed methodology:

1. Firstly, a strategy has been adopted to collect the common features between different SoC protocols by performing a detailed comparative study on their different aspects and domains (Table 1).
2. Secondly after scrutiny and observing all the similarities as the common commands, signals' operations and topology could be obtained to build a generic UVM template.
3. Common features are used as an input to produce a generic UVM and to implement a BFM for the selected protocols to be tested.
4. Finally, the shown flowchart in Fig.1 briefly summarize our methodology.

Main Challenges of Previous Environment/Verification Methodology were as follows:

1. Reusability
    a. Test cases from pre-designed verification environments could not be reused.
2. Significant time was spent in reproducing and tailoring the environment to be generic.
3. Wire level assignments and assertions are protocol dependent [9].



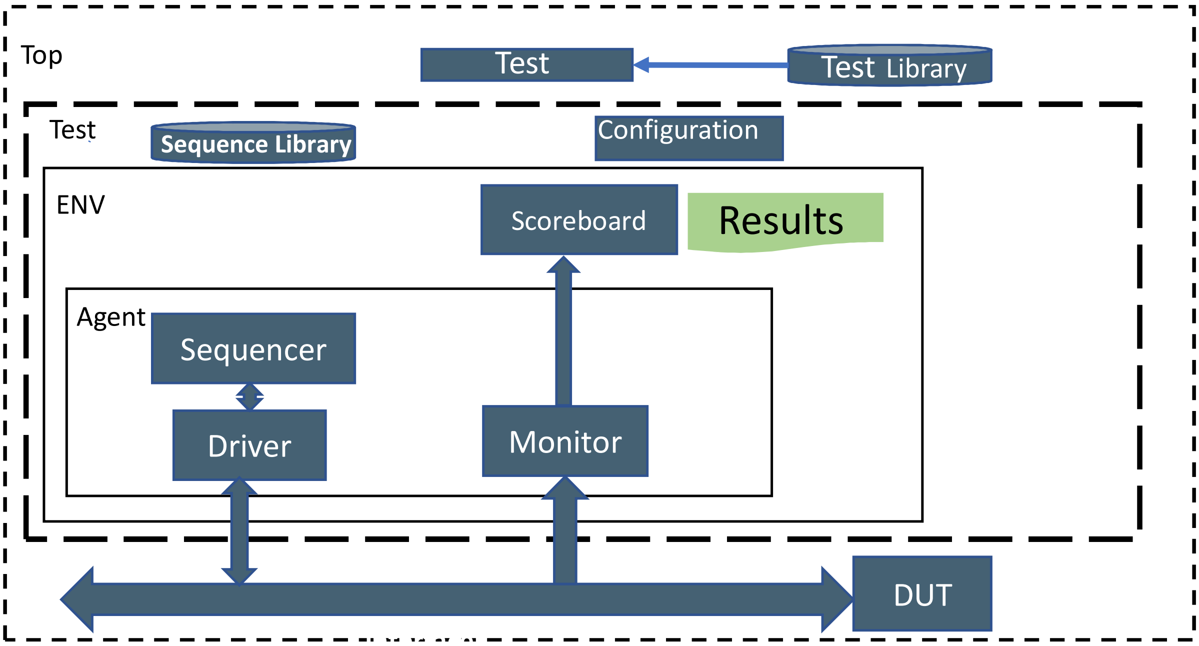


Fig.2 UVM Architecture

III. IMPLEMENTATION AND VERIFICATION

By investigating the most common architectures of the UVM and by investigating the test scenarios and sequences for the selected SoC buses, we came up with the following generic architecture. So, our environment can now be recognized as shown in Fig.2.

The top module contains different test case scenarios, each one of them instantiate our environment then the environment instantiates our scoreboard and master/slave agent which contains the sequencer, driver, monitor. Notice that we developed a master and a slave agent and according to the user's needs we can choose which agent suits his test case. Concerning the test cases, the main focus was pointed to the most generic command such as read, write, write then read, wait then read and wait then write. For the sequence item and interface, data and address widths all are parameterized. The driver is the most challenging block due to the differences between the sequences of operations and signals in each protocol, but thanks to the common functionality of SoC buses, we could successfully choose some scenarios that could be applied to all SoC buses. Moreover, in order to provide full controllability to the designer, there's a function built specially for using generic names and a generic operation flow.

A. Case Study 1: AMBA APB Protocol

First, we applied our UVM environment to APB protocol, all of the APB operations where covered in our test cases, the driver operations were adjusted manually and we used the master agent as we want to test a slave memory, then we changed the data/address size parameter to be 32 bits to fit the size of the APB interface. Listing 1 shows the APB test scenario [10].

B. Case Study 2: AMBA AHB Protocol

Our next case was AHB, we applied our UVM environment to AHB protocol, some blocks where efficiently reused as the sequencer, portion of the driver as shown in Listing 2 and the basic commands as single read and write in the tests [10].

C. Case Study 3: Avalon Protocol

Avalon is very similar to APB protocol, so we didn't change much in our environment, we made minor modifications in the signal's names and in the driver/test cases [13].

Listing 1: Alternate read and write with wait

```
class apb_one_write_one_read_wait_test extends
apb_base_test;

`uvm_component_utils(apb_one_write_one_read_wait_test)

task run_phase (uvm_phase  phase);
        apb_write_al_sequence   apb_wr_al_seq;

        apb_read_al_sequence   apb_rd_al_seq;

        apb_with_wait_sequence apb_wait_seq;
```

Listing 2: Reusable build phase of the driver

```
function void ahb_mdriver::build_phase(uvm_phase
phase);
        if(!uvm_config_db#(ahb_magent_config)::get(this,
"", "ahb_magent_config", magt_cfg))
        begin
            `uvm_fatal(get_full_name(), "Cannot get VIF
from configuration database!")
        end
        super.build_phase(phase);
    endfunction
```

The following figures are samples of operations done on the APB and AHB protocols.

A. APB Bus

For APB a combination of tests was simulated, read and write were alternated with wait feature. The waveform is shown in Fig.3 and Fig.4

B. AHB Bus

For the AHB bus, there are more advanced functions than the APB so an increment test was performed with an error test to check how the slave will respond.

Also, wrap16 read test and Increment 8 write test were implemented and the behavior of the addresses were checked as shown in Fig.5 and fig. 6.

IV. PERFORMANCE EVALUATION

According to a real extracted statistic of the effort done to verify a certain bus (AMBA AXI3 or AMBA AHB), which reveals that one consumes around 1 month to build the whole UVM environment with test scenarios (partial but not full test cases) and consumes around 1 week to build the test cases only (without worrying about building the UVM environment) as shown in Fig. 5 [4]-[9].

Also, Fig. 7 and Fig.8 demonstrates the number of weeks required to build a UVM for each protocol from scratch vs. using a generic template and make use of the reusability.

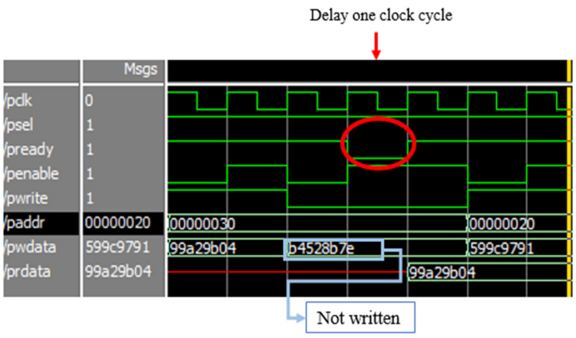


Fig.3 APB Alternate read and write with wait.

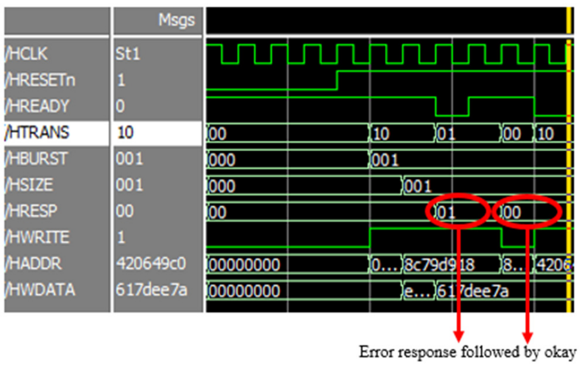


Fig.4 AHB increment write followed by error.

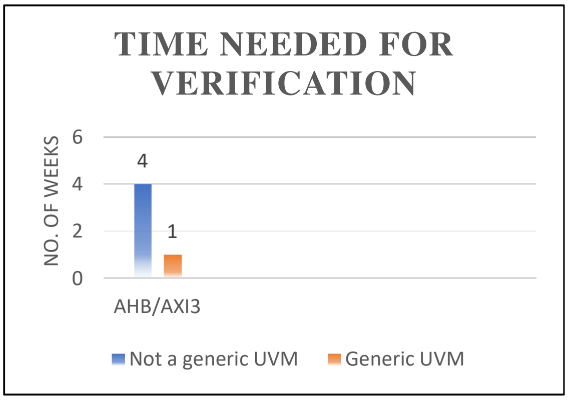


Fig.7 Statistics for the time needed for verification.

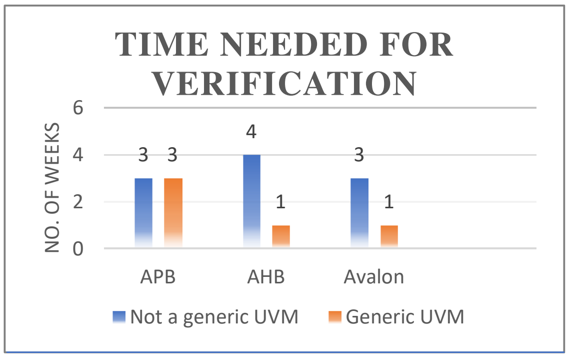


Fig.8 Comparison between generic and a non-generic UVM.

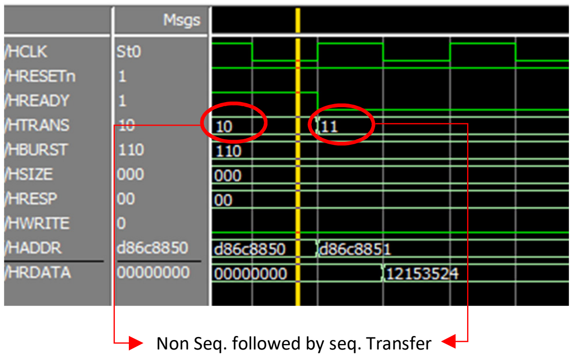


Fig.5 AHB Wrap16 Read.

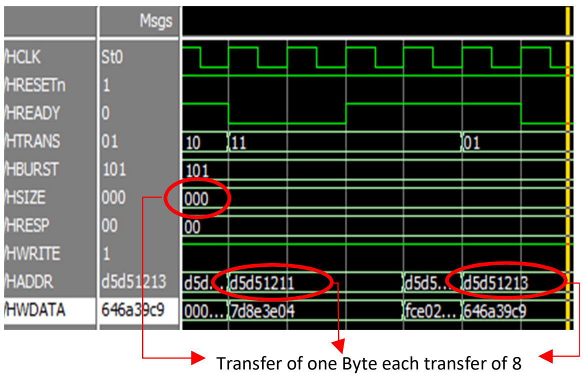


Fig. 6 AHB Incre8 Write.

Table 1 Comparative Study between different SoC Buses

Feature / Bus		Data /Address Size	Burst	Domain	Arbitration type	Bus Topology
Avalon[13]		8, 16, 32, 64, 128	Supported	FPGA & SOPC	Slave - side	Point to point
Wishbone[19]		8,16,32 &64/ 0-64	Supported	FPGA	Defined by the end user.	Shared Bus/Crossbar-switch/Data flow ring
STBus[16]		8,16,32,64	Supported	ATM Networks (Consumer App)	Priority/TDMA	Shared bus/Crossbar
SAS[20]		8,16,32&64 /0-64	Supported	Move data to & from hard disk drives	Priority	point-to point switched
SATA[18]		Up to 2048 byte	Not Supported	Connect to hard disk drivers	No arbitration	Point to point
PCI[14]		2/64-bits	Supported	Chip to chip	TDMA	Point to point
Core frame[20]		400MB/sec	Supported	Bluetooth/Wi-Fi communications	Priority	Point to point
Core connect[11]		16, 32 and 64 byte 256 bits	Supported	SoC	Priority	Hierarchal Shared/crossbar
MIPI (DigRf)[7]		1.5Gb/s	Not Supported	Mobile.	No arbitration.	Point-point/multiplexing
LIN[17]		0 to 8 Bytes/ No address	Not Supported	Automotive Domain	No arbitration.	Shared bus (1 to 16 slaves)
FlexRay[12]		10MbPS/None	Supported (Dynamic slot)		TDMA / FTDMA/ Priority	Flexible
CAN[15]		Up to 64 bits	Supported (dynamic slot)		Priority	Shared bus
MOST[5]		Default is16 bit address	Not specified		TDMA/ Priority	Ring
AMBA[10]	AXI3	32, 64, 128, 256, 512, or 1024 bits wide / 32 bit.	Supported	Designing high-performance embedded microcontrollers	No arbitration	Point to point
	APB	(8,16 or 32 bits) /32 bit. /32 bit.	Supported		No arbitration	Point to point
	ASB	(32,64,128 or 256 bit) / 32 bit. or 256 bit) / 32 bit.	Supported		Priority	Matrix
	AHB	(32,64,128 or 256 bit) / 32 bit. or 256 bit) / 32 bit.	Supported		Priority	Matrix

V. CONCLUSIONS AND FUTUR WORK

In this paper, a generic UVM verification environment for verification of SoC buses is proposed. As compared to earlier methodologies, the proposed methodology helped in saving verification cost and effort with the help of a detailed comparative survey between more than 10 protocols. Although this environment is developed for SoC buses with single interface. The concept could be extended for SoCs with multiple interfaces. So, in future, as an adaptation, a generic UVM template could be generated to verify more buses and develop a verification environment for the whole SoC.

REFERENCES

[1] Bennini L., DeMicheli G., Networks on Chips: A New SoC Paradigm, IEEE Computer, Vol. 35, No. 1, Jan. 2002, pp.70- 78.

[2] K. Salah, “A UVM-Based Smart Functional Verification Platform: Concepts , Pros , Cons , and Opportunities,” In Design & Test Symposium (IDT), 2014 9th International, pp. 94-99. IEEE, 2014.

[3] K. Salah, and H. Mostafa. "Constructing Effective UVM testbench for DRAM Memory Controllers." New Generation of CAS (NGCAS). IEEE, 2018.

[4] K. Salah "A Unified UVM Architecture for Flash-Based Memory." 2017 18th International Workshop on Microprocessor and SOC Test and Verification (MTV). IEEE, 2017.

[5] [2] P. Patil, V. Sangamkar, “A Review of System-On-Chip Bus Protocols”, International Journal of Advanced Research in Electrical, Vol. 4, Issue 1, Jan. 2015, pp. 2.

[6] GRZEMBA, book is based on the MOST Specification, E2, 2010.

[7] MIPI alliance specifications, https://www.mipi.org.

[8] [https://github.com/marcoz001/axi-uvm

[9] generic system verilog universal, https://arxiv.org/ftp/arxiv/papers/1301/1301.2858.pdf

[10]ARM.AMBA Specifications v2.0, 1999.

[11] Core Connect Bus specifications, by IBM,International, 1999.

[12]FlexRay Communications System Protocol Specification.

[13] Altera Avalon, Avalon bus specification: Ref. manual. July, 2003.

[14] Rovin and Sagar, PCI Bus Specifications.

[15] Siemens Corp., Version 2.0, Siemens Microelectronics.

[16] STbus communication, User manual, October 2012.

[17] LIN Protocol and Physical Layer Requirements -TI, Feb. 2018.

[18] Essential Guide to Serial ATA and SATA Express.

[19]Richard Herveille, “WISHBONE SoC Interconnection” .

[20] Serial Attached SCSI Standard.