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

Alaa Hussien¹, Samar Mohamed¹, Mohamed Soliman¹, Hager Mostafa¹, Khaled Salah², Mohamed Dessouky², Hassan Mostafa³

¹Department of electronics and communications, faculty of engineering, Ain Shams University, Cairo, Egypt.
²Mentor Graphics, Cairo, Egypt.
³Electronics 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 depend 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].

![Flowchart of the Steps Taken](image-url)
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].
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.](image)

![Fig.4 AHB increment write followed by error.](image)

![Fig.5 AHB Wrap16 Read.](image)

![Fig. 6 AHB Incre8 Write.](image)

![Fig.7 Statistics for the time needed for verification.](image)

![Fig.8 Comparison between generic and a non-generic UVM.](image)
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.

Table 1 Comparative Study between different SoC Buses

<table>
<thead>
<tr>
<th>Feature Bus</th>
<th>Data / Address Size</th>
<th>Burst</th>
<th>Domain</th>
<th>Arbitration type</th>
<th>Bus Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avalon[10]</td>
<td>8,16, 32, 64, 128</td>
<td>Supported</td>
<td>FPGA &amp; SOPC</td>
<td>Slave - side Point to point</td>
<td></td>
</tr>
<tr>
<td>Wishbone[10]</td>
<td>8,16,32 &amp;64/0-64</td>
<td>Supported</td>
<td>FPGA</td>
<td>Defined by the end user. Shared Bus/Crossbar-switch/Data flow ring</td>
<td></td>
</tr>
<tr>
<td>STBus[10]</td>
<td>8,16,32,64</td>
<td>Supported</td>
<td>ATM Networks</td>
<td>Priority/ TDMA Shared bus/Crossbar</td>
<td></td>
</tr>
<tr>
<td>SAS[10]</td>
<td>8,16,32&amp;64/0-64</td>
<td>Supported</td>
<td>Move data to &amp; from hard disk drives</td>
<td>Priority point-to-point switched</td>
<td></td>
</tr>
<tr>
<td>SATA[10]</td>
<td>Up to 2048 byte</td>
<td>Not Supported</td>
<td>Connect to hard disk drivers</td>
<td>No arbitration Point to point</td>
<td></td>
</tr>
<tr>
<td>PCI[14]</td>
<td>2/64-bits</td>
<td>Supported</td>
<td>Chip to chip</td>
<td>TDMA Point to point</td>
<td></td>
</tr>
<tr>
<td>Core frame[28]</td>
<td>400MB/sec</td>
<td>Supported</td>
<td>Bluetooth/Wi-Fi</td>
<td>Point to point</td>
<td></td>
</tr>
<tr>
<td>Core connect[10]</td>
<td>16, 32 and 64 byte 256 bits</td>
<td>Supported</td>
<td>SoC</td>
<td>Priority Hierarchical Shared/crossbar</td>
<td></td>
</tr>
<tr>
<td>MIPI (DigRF)[9]</td>
<td>1.5Gb/s</td>
<td>Not Supported</td>
<td>Mobile</td>
<td>No arbitration. Point-point/multiplexing</td>
<td></td>
</tr>
<tr>
<td>LIN[17]</td>
<td>0 to 8 Bytes/ No address</td>
<td>Not Supported</td>
<td>Automotive Domain</td>
<td>No arbitration. Shared bus (1 to 16 slaves)</td>
<td></td>
</tr>
<tr>
<td>FlexRay[14]</td>
<td>10MbPS/None</td>
<td>Supported</td>
<td>(Dynamic slot)</td>
<td>TDMA / FTDMA/ Priority Flexible</td>
<td></td>
</tr>
<tr>
<td>CAN[10]</td>
<td>Up to 64 bits</td>
<td>Supported</td>
<td>(Dynamic slot)</td>
<td>Priority Shared bus</td>
<td></td>
</tr>
<tr>
<td>MOST[10]</td>
<td>Default is 16 bit address</td>
<td>Not specified</td>
<td></td>
<td>TDMA/ Priority Ring</td>
<td></td>
</tr>
<tr>
<td>AXI3</td>
<td>32, 64, 128, 256, 512, or 1024 bits wide / 32 bit.</td>
<td>Supported</td>
<td></td>
<td>Designing high-performance embedded microcontrollers</td>
<td></td>
</tr>
<tr>
<td>APB</td>
<td>8,16 or 32 bits /32 bit.</td>
<td>Supported</td>
<td></td>
<td>No arbitration Point to point</td>
<td></td>
</tr>
<tr>
<td>ASB</td>
<td>(32, 64, 128 or 256 bit) / 32 bit. or 256 bit / 32 bit.</td>
<td>Supported</td>
<td></td>
<td>No arbitration Point to point</td>
<td></td>
</tr>
<tr>
<td>AHB</td>
<td>(32, 64, 128 or 256 bit) / 32 bit. or 256 bit / 32 bit.</td>
<td>Supported</td>
<td></td>
<td>Priority Matrix</td>
<td></td>
</tr>
</tbody>
</table>

V. CONCLUSIONS AND FUTURE 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

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