Integration of HDL Logic inside SystemVerilog UVM based Verification IP

Aleksandra Panajotu
Elsys Eastern Europe
Belgrade, Serbia
(aleksandra.panajotu@elsys-eastern.com)

***Abstract*- Level of complexity of the Verification IPs which need to be developed during the verification process is dictated by the complexity of the system that is surrounding the Device under Test. In cases where it is required of the Verification IP to behave as similarly to the real system as possible, development of such Verification IP can be difficult. This paper will propose a method of integration of HDL Design Logic inside a VIP Wrapper which would simplify the process of VIP development in those cases and increase the precision with which VIP will model expected Design behavior. It will discuss use-cases for this method as well as its advantages and disadvantages.**

I. INTRODUCTION

Development of Verification IPs (VIPs) is common practice for UVM based Verification. Implementation of the logic inside every VIP is dictated by the behavior of the HDL Design that needs to be modeled. As complexity of communication between the Design under Test (DUT) and the system rises, implementation of the required VIPs becomes more demanding. In most cases, the behavior of VIP is not directly dictated by the Design implementation that it is modelling, but by the specification used to develop that Design. On the other hand, sometimes the nature of the DUT requires that the VIP behaves as similarly to the already implemented Design as possible. That requirement usually does not represent an issue to the Verification team, but if the surrounding Design uses complex logic that is hard to reproduce in any other way than implementing the same logic in the VIP, the development of such VIP can require great effort.

This paper will focus on the integration of HDL Design Logic inside a VIP Wrapper which simplifies the process of VIP development, increases the precision with which VIP will model the expected Design behavior and decreases the effort needed for updating/upgrading the VIP due to the changes applied in the Design Logic it models in cases where the randomization of the stimuli that VIP should provide is not needed and the exact replication of behavior of surrounding logic is expected.

This approach would provide more precise VIP behavior considering that the most complex part of the neighboring IP is being reused and not modeled, which reduces the work needed to develop the complex logic. Focus of this paper will be the implementation of a hat models the behavior of surrounding HDL Design that is processing some inputs coming from the system and providing stimuli to the DUT.

It is important to note that Design that is being used in this approach needs to be already verified and available for reuse. Also, this approach is recommended only for cases where the DUT requires the behavior of the VIP in question not to differ from the behavior of one part of the logic it models due to the requirements of the DUT.

The paper will go into details about VIP Wrapper development and implementation, examples of the implementation code, integration in the Testbench and use-cases.

A. VIP Wrapper overview

VIP Wrapper contains three distinct parts: SystemVerilog UVM based VIP, HDL Design logic and the Wrapper Interface, hereafter referred to as the VIP, the Design and the Wrapper. The idea is to integrate the Design and the interface module of the VIP into the Wrapper. The VIP will keep all its SystemVerilog and UVM features and will represent a part of the system that is driving the input signals of the Design that will process them and use them to stimulate the DUT.

II. VIP WRAPPER IMPLEMENTATION AND DEVELOPMENT

A. *Implementation overview*

VIP Wrapper implementation will need following steps to complete:

- Implementation of the Wrapper module that will contain a VIP interface module and Top Module of the Design
- Implementation of the VIP that will provide input stimuli to the Design in the VIP Wrapper
- Instantiation of the Design inside the Wrapper module
- Instantiation of the connections between the Design and the VIP
- Integration of the VIP Wrapper inside the Testbench module

B. *Development of the VIP Wrapper*

The VIP should be developed according to UVM methodology. Since it is being used as a surrounding logic for the Design, it has an interface that is matching the part of the Design interface that is dependent on the surrounding system. This interface will be only internally visible, and it only serves for communication between the Design, the VIP and the Wrapper. The VIP simulates the rest of the system that Design needs in order to function properly. This allows the Verification team to develop randomized sequence items and sequences which introduce randomization to the VIP's behavior, even though the Design does not support randomization.

The Wrapper will act as an interface to the rest of the UVM environment where it will be used. SystemVerilog does not allow instantiation of a module inside an interface instance, so the Wrapper will be declared as a module. This does not affect its integration or its behavior. The Wrapper has the Design instance and the VIP interface instance inside it and the connections to the DUT in the Testbench. It will also have internal connections between the VIP interface and the Design.

The overview of the VIP can be seen in Figure 1 and the example code for the Wrapper module can be seen in Figure 2.

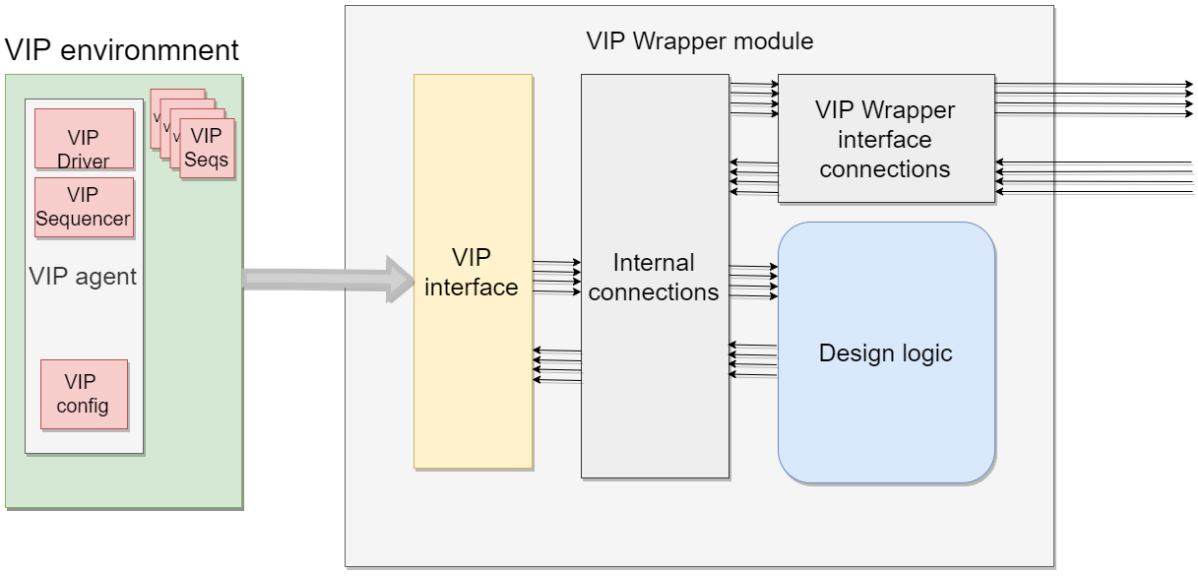

Figure 1. VIP Wrapper overview.

```
module wrapper_if_module
(
//input and output signals of the VIP Wrapper
);

        //internal signals used for connections between the Design and the VIP
        interface

        design_module          design_instance
        (
        //input and output connections of the Design
        );

        vip_interface          vip_interface_instance
        (
        //input and output connections of the VIP
        );

                        //assignments of the internal signals

endmodule : wrapper_if_module
```

Figure 2. VIP Wrapper code example.

The VIP doesn't have to be a single component, it can be an environment with a virtual sequencer and several VIPs instantiated inside the environment that are working independently (i.e. independent request-acknowledge modules) or in coordination (simulating some more complex part of the system for which multiple VIPs can be used instead of using one complex VIP). If multiple VIPs are being used, their interfaces will be connected to the Design and the Testbench in the same manner and their environments or agents will be instantiated inside the environment of the VIP. Overview of the VIP Wrapper with multiple VIPs instantiated inside can be seen in Figure 3.

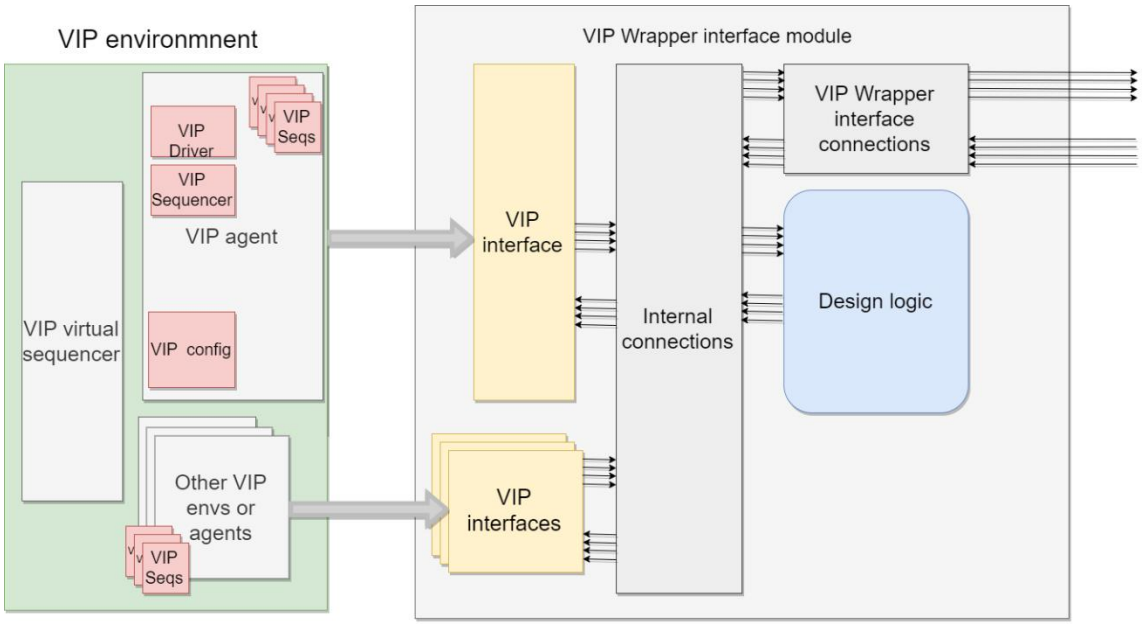

Figure 3. Overview of the VIP Wrapper with multiple internal VIPs.

Although implementation of checkers and functional coverage is not in the focus of this paper, it is worth mentioning that there is a possibility of implementation of protocol checkers in the Wrapper and VIP interface instances. For more advanced checkers, it is possible to develop UVM monitor object that will monitor the Wrapper signals, as the VIP is only aware of its interface which does not have to contain all the signals from the Wrapper. It is also possible to implement functional coverage inside the VIP or to create a new UVM monitor and subscriber that could collect coverage.

III. INTEGRATION IN TESTBENCH

VIP Wrapper integration very slightly differs from the regular VIP integration. The Wrapper is instantiated in the testbench as a module and is connected to the signals like any other module or interface. Signals of the VIP interface should not be connected to the Testbench, only the Wrapper interface signals. VIP interface that is instantiated in the Wrapper will be passed through the UVM configuration database from the Testbench and the VIP environment will be able to acquire it. It is important that the VIP interface instance is passed to the UVM configuration database instead of the Wrapper instance as the VIP is not aware that is a part of a wrapper and is expecting the get its own interface through the database. Example code of the Testbench with Wrapper instance is shown in Figure 4. Overview of the Testbench with VIP Wrapper instance is shown in Figure 5.

```
module testbench;

        //internal testbench signals

        dut_top_module          dut;
        (
        //input and output connections of the DUT
        );

        wrapper_if_module       vip_wrapper_instance
        (
        //input and output connections of the VIP Wrapper
        );

        initial begin
                uvm_config_db#(virtual vip_interface)::set(null, "*.vip_wrapper_instance*",
                        "vif", vip_wrapper_instance.vip_interface_instance);

                run_test();
        end

endmodule : testbench
```

Figure 4. Example of integration in Testbench.

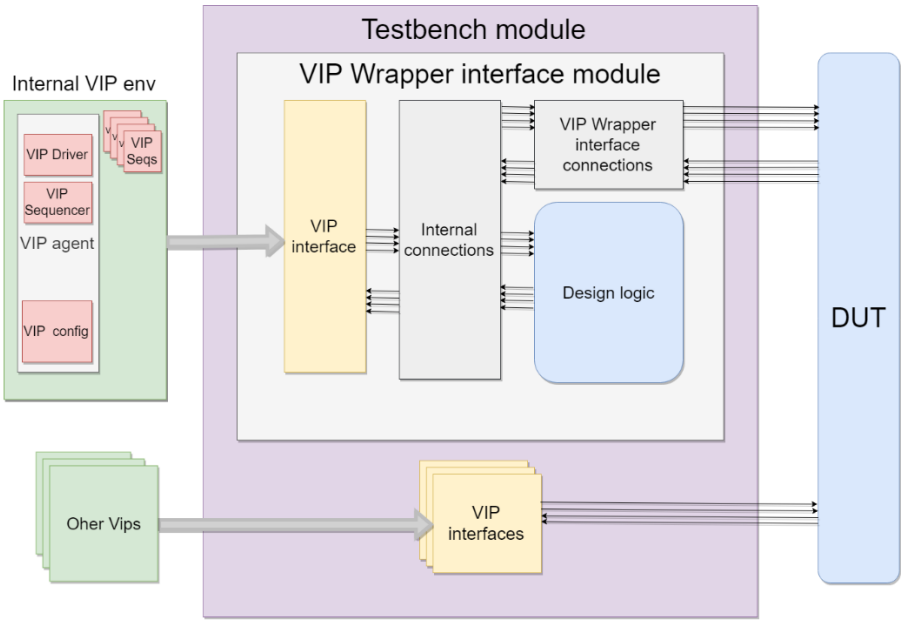

Figure 5. Integration of the VIP Wrapper inside a Testbench.

IV. USE-CASES

This method is most applicable in cases where the DUT should be connected to a part of the system that can be split into two parts: (1) module with complex logic that is difficult to model with high precision but is available for reuse and (2) the surrounding logic, which is generating inputs to that module. The logic in the module would then model the behavior and drive input signals of the DUT. Figure 6 describes the layout of the DUT, surrounding design logic and the System. The surrounding logic in Figure 5 is the part of the system that the Verification team needs to model by developing the VIP. The regular approach would consider the implementation of a regular VIP which would contain very complex driver logic and large sequence library. The approach this paper proposes enables simplification of the VIP implementation and provides the module that generates the stimuli to the DUT that will completely match the surrounding logic.

A. Effective use of this approach

This approach can be used in situations where the exact behavior of a part of the surrounding logic must be reproduced. On projects where multiple IPs that are closely connected are being developed at the same time but by different Verification teams, especially when the IPs are not communicating using a standard protocol, or if one large IP is being verified as several smaller IPs and for every part of the IP a UVM testbench has to be developed, it can be very hard to create VIPs that simulate the behavior of the surrounding system. Situations where System that needs to be modeled is large and consist of multiple modules that can work independently, as mini-SOCs, or the modeled IP has a high number of synchronizers and its behavior must be perfectly modeled.

An additional problem can introduce multiple IPs that are being developed at the same time and the synchronization mechanism between them slightly changes with every development iteration. In this situation, there are multiple issues with pure UVM VIP approach:

- It is very hard to constantly change the VIP that is being used for verification
- There are always differences in synchronization between the modeled version and the actual Design
- Multiple IPs are using the logic from this Design and every Verification team developed its own VIP

The solution is to integrate the module of the Design that is responsible for all the calculations and synchronizations and create a VIP that models the feedback the Design needed from the system. This way every time the Design gets updated, the Wrapper would also be updated with the new version of the Design. The changes needed from the Verification team are greatly reduced in case the rest of the system is not changed and the accuracy of the modeling was improved.

This approach proved very effective on a project where a large IP that needed to be modeled had a module that processed multiple request signals that are coming from the whole system and generated feedback to the DUT. It was of great importance for the testbench development to have the processed stimuli from the IP behaving precisely the same as it is implemented in the actual Design due to various synchronization mechanism. The VIP Wrapper reduced the effort needed to recreate the same behavior that was already present in the Design to the development of the VIPs that generated requests to the Design that in return used its complex logic and provided stimuli the DUT needed.

It is highly preferable that the Design that is being used inside the Wrapper is fully verified. In case the verification is not fully done, cooperation between Verification teams that are using the Wrapper and the one that is working on the Verification of the Design that is being used is very important. In this case, it is also recommended that the same Verification team that is doing the verification of the Design that is being reused is developing the VIP Wrapper as it could support eventual changes in the Design implementation. Positive aspects of this kind of cooperation are that in cases where some corner case behavior was not accurately defined in the specification or was not even discovered, it can be revealed by the verification teams that are using the VIP Wrapper. If wrapped Design is not fully verified the risk of having a bug in it is very high and it is recommended to use this approach with great caution or not use it at all.

Using an RTL Design for purposes of verification is not a new approach, but the possibility of wrapping it inside a UVM VIP can bring benefits in form of more control over it by using the other UVM VIP instances inside the Wrapper that can be controlled using randomized sequences. Practical implementation differs from one project to another, but the guidelines provided in this paper can ease the development of such VIP in cases where it can be used.

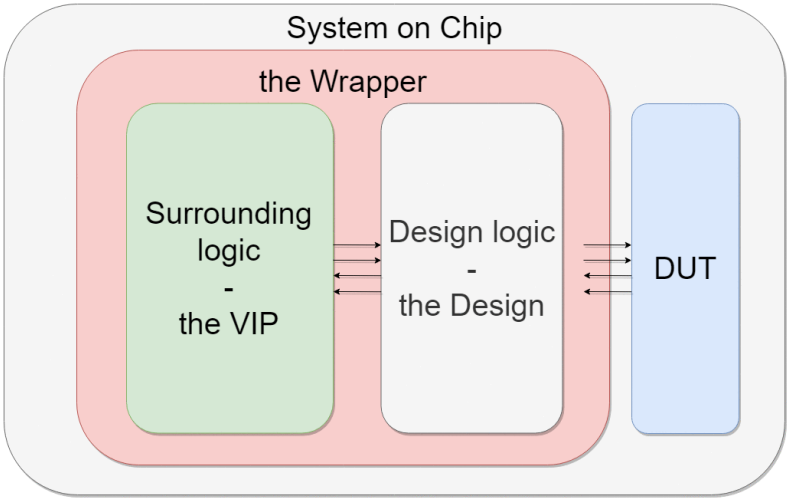

Figure 6. Layout of the System that is of interest for this paper.

B. Reusability

The extent of reusability of VIP developed using this approach depends on the actual implementation. If there is no boundary difference between the regular VIP instance and VIP Wrapper instance that is being used as a replacement, the integration would require much effort. The biggest impact the change would introduce is related to sequences that are being called in test cases.

This approach shows good results on the projects on which the same VIP is be used in multiple Verification teams, as it reduces the work needed to implement the VIP multiple times by different Verification teams and improves the precision with which the surrounding design is modeled.

V. CONCLUSION

Development of a VIP Wrapper can be beneficial when exact replication of the surrounding logic is needed. Even in those situations, development of the standard UVM VIP can be done, but the approach proposed in this paper can reduce the effort needed for development. The VIP Wrapper developed using this method has the same core behavior as the design logic that it should model with added possibility of randomization using UVM VIPs wrapped inside. It provides an easier implementation of stimuli generation and an easier VIP adaptation to an eventual change in the System providing stimuli to the DUT. It allows Verification team to develop less complicated VIPs and combine them with already available RTL Design, but also introduces a risk of a bug being present inside the Design that is being reused, and for that reason, the Design for integration must be verified. The randomization of the stimuli that Design inside the VIP Wrapper provides is reduced and comes down to the randomization of the inputs to the Wrapped Design. For this reason, this approach is recommended for generating stimuli that should not be greatly randomized, as it can provide greater precision and efficiency in VIP development.

REFERENCES

[1] IEEE Standard for SystemVerilog: Unified Hardware Design, Specification and Verification Language, IEEE Std. 1800-2017
[2] Universal Verification Methodology (UVM) - http://www.accellera.org/activities/working-groups/uvm