2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Verification of a Multi-language Components A case study:

Specman E Environment with SystemVerilog UVM

Eran Lahav, Verification Specialist, Veriest Solutions LTD, Israel (*eranl@veriests.com*)

***Abstract*— Verification of a complex SOC today demands the use of Verification IPs from diverse sources. The ability to utilize available verification components and embed them into an existing Verification Environment, which often consists of different languages, is of great importance . The Accelera UVM-ML Open Architecture [1] provides the ability to assemble and co-simulate components which are written in different languages. Nevertheless, some synchronization aspects - such as sequences alignment and data transport between those components - are left for one's determination. In this paper, we demonstrate a common case for Multi-language necessity: a SOC that is generally verified with a Specman E environment that utilizes an SV UVM Verification Component from an external vendor. In the implementation of this system, we deployed a mechanism for data and bilingual sequence synchronization. In this project, we also deal with a dilemma: In what circumstances is it better to translate (or rewrite) code to another language, rather than combine it in a different language environment.**

Keywords—UVM, Multi Languages, Specman

I. INTRODUCTION

Our work is based on a project where a SOC with Specman E environment utilized a SystemVerilog (SV) UVM (Universal Verification Methodology) Verification Intellectual Property (VIP) from an external vendor. It is a typical case for multi-language necessity: The current verification environment, which is written in one language, should be connected to an IP's verification component, written in another language. The use of UVM-ML Open Architecture (UVM-ML OA) helps to organize and run multi-frame simulations **[**2**]**. It resolves issues of global synchronization, e.g. run-phases, objection mechanism, etc. Nevertheless, some aspects of the multi-language require special consideration: Synchronization between sequences (or tasks in general) and data transfers from one language to another. These aspects include, among others, waiting for events and monitor/scoreboard data, that should be shared between the two code sides. Those aspects are not fully defined and resolved by the standard methodology. In this project, we have been engaged with those issues and tried to cover this gap and suggest ways for synchronization of processes and data transfer between different language components.

Another aspect that had not been widely considered is code translation. In some cases, re-writing part of the code, (either manually or automatically) could be a more reasonable solution. In this paper, we discuss the trade-off between the two alternatives: re-write the code and co-running with original languages. In our project, several sequences were translated from SV to E due to these considerations.

II. THE SYSTEM

The project's SOC Verification Environment (see "Figure 1- System Description") consists of a composite Specman E environment with a multitude of components and functions. It includes a Core with Firmware (FW) sequence library and models for its peripheral's devices and interfaces. A new I3C IP from an external vendor was embedded in the SOC design model, and the verification scheme had been examined for it. As the main purpose of the verification was aimed at integration and connectivity, the VIP from the same vendor was preferred over third-party vendors. Nevertheless, this VIP was an SV-UVM component, and additional consideration had to be taken regarding the bilingual barrier. This VIP utilizes an RTL design (same as the one in the DUT) to employ the I3C

protocol. It contains a series of sequences that carry out I3C by AMBA-AHB access command to the RTL. This structure has a benefit for our further development as will be described in the next section.

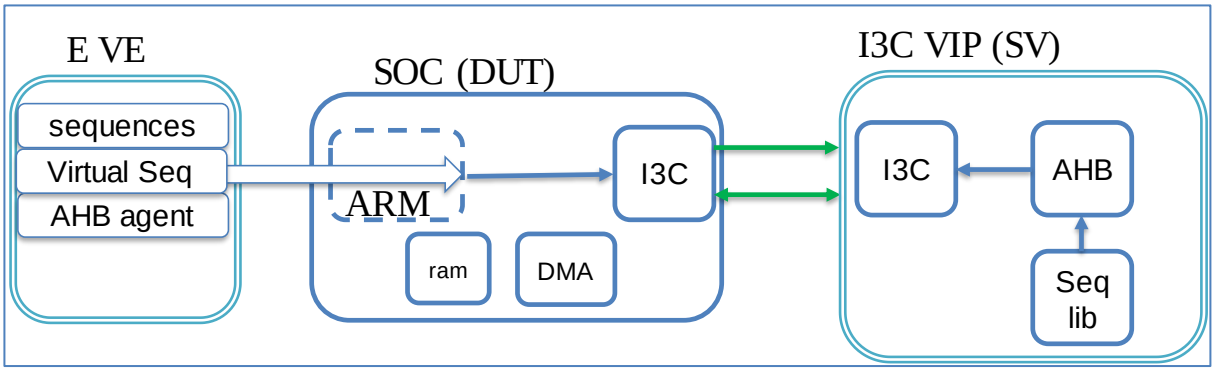

Figure 1- System Description

III. TRANSLATE OR ADJUST?

Multi-language design considerations depend on the situation as the organization of the added component, the way the main environment performs its function, and the ability to optimally re-use both the original environment and the new component. One choice is to translate (or rewrite) the component's code to the other language. This option is highly suitable for sequences that should be run from a sequencer that is written in another language. Although UVM-ML supports this combination (sequence and sequencers from different languages), it is quite complex and cumbersome.

A. DUT Sequences

In our project, the VIP contains sequences of memory accesses that activate the IP. In the DUT we need to add similar sequences for configuration and execute commands. The alternatives for getting those sequences were: (VIP's stand alone in "Figure 2 - Stand Alone Test bench of VIP" is displayed for reference)

1. Run SV-Sequences under E sequencer directly:
   This option is complex and risky, as discussed above.
2. Execute in UVM Test bench side and performing memory access by calling E methods (as shown in Figure 3- Using Memory Access Port):
   The VIP Testbench is based on several agents. The sequences for both Master and Slave can run as stand-alone. The Read/Write access of one side will be transferred to E and be executed in E, while the other side would active VIP as before.
3. Translate SV-UVM VIP sequences to E and use them within the SoC virtual sequences (see "Figure 4- Translation option"):
   Translate SV sequences to E sequences, preferably with a script, and run the E translated in the E environment against the VIP sequences (e.g. E translated Master sequence vs. SV VIP Slave sequence)
4. Writing the sequence in E from scratch (see the top of "Figure 5- Write Sequence from Scratch"):
   Writing sequences from I3C IP specifications that are implementing I3C Master and Slave activity. If the VIP has dependencies between Master and Slave sequences (which the current VIP had), it will require special care.
5. Writing C code and run as a program on Core. (see the bottom of "Figure 5- Write Sequence from Scratch"):

As in 4, but make it more difficult to synchronize with the other side (Master vs. Slave)

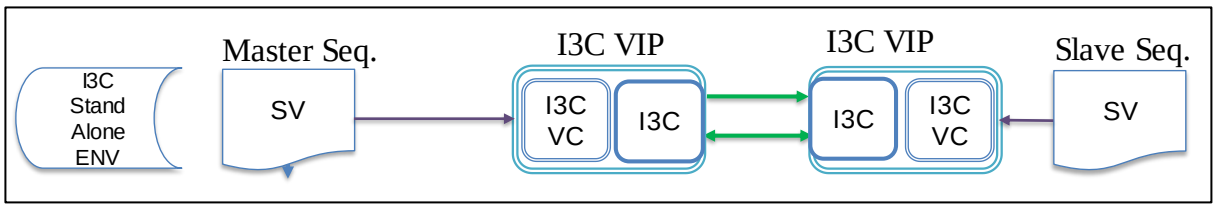

Figure 2 - Stand Alone Test bench of VIP

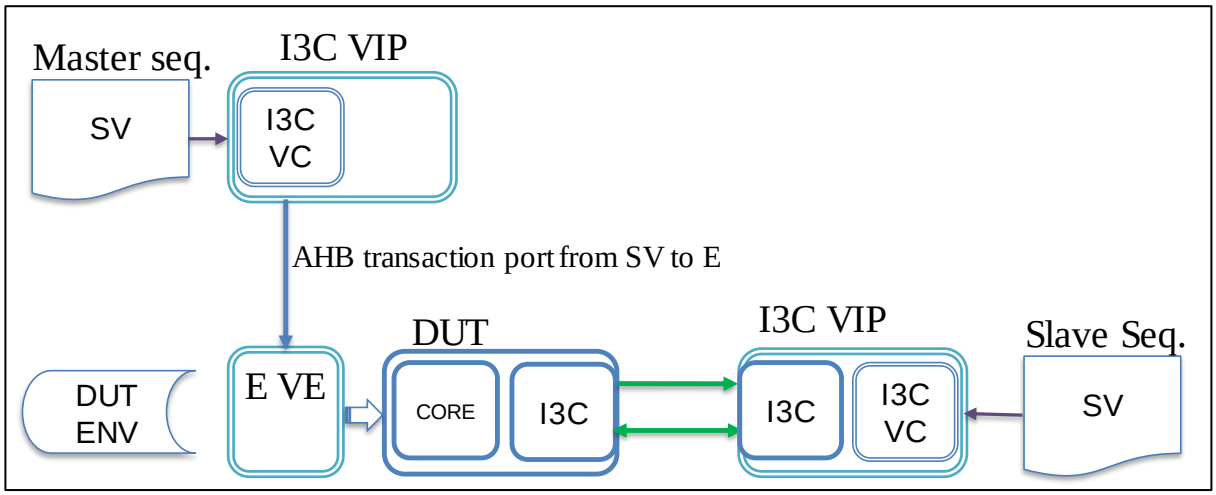

Figure 3- Using Memory Access Port

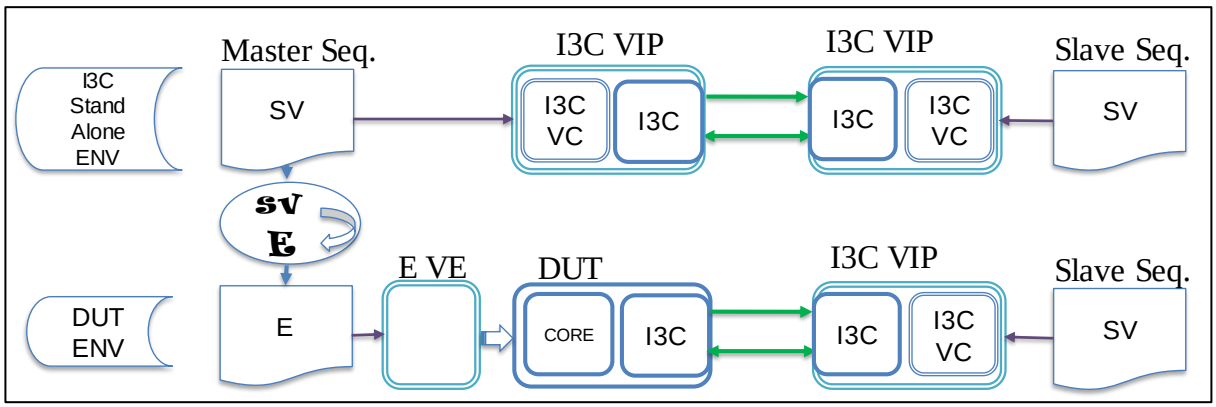

Figure 4- Translation option

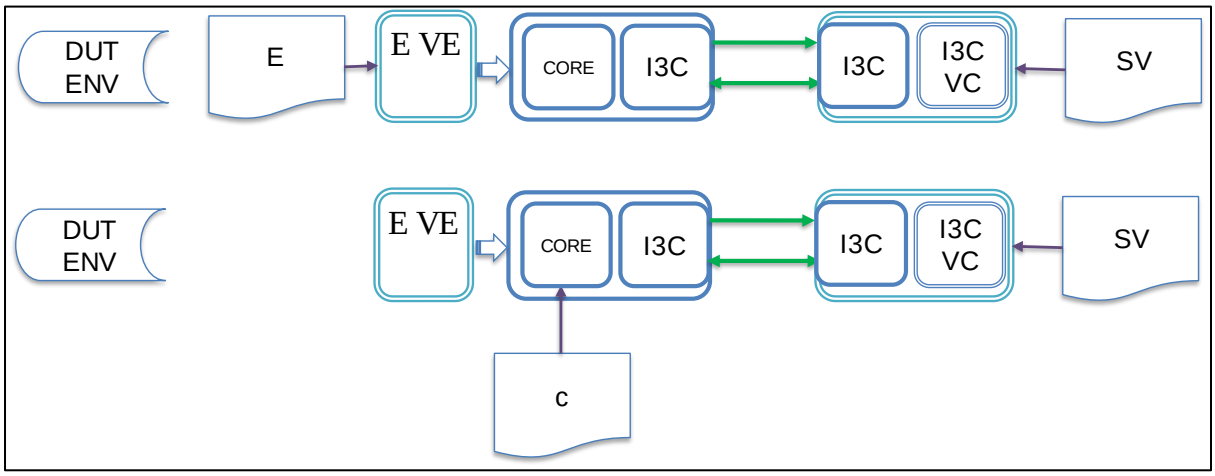

Figure 5- Write Sequence from Scratch

The consideration and consequences of taking each one of the alternatives are summarized in "Table 1 - Translate/Rewrite/Co-languge comparison". As preserving the test set was of high importance for the customer, we had to pick one of the two: Translate (option 3) or Execute E memory access from UVM Sequence.

B. SV to E translator

After a short trial with translation, it was clear that a script can be easily written for SV sequence translation. It was not intended to be a full SV to E translator, though it could handle a subset of SV (and UVM) that was contained within the sequences. The translator was written with Python and it took 3 person-week to complete. It is limited to the command and semantics that were found in the translated code and used some shortcuts (as special SV macro that could be covert directly). We estimate a full SV to E Translator to require about one person-year.

IV. SYNCHRONIZATION BETWEEN LANGUAGE WORLDS

A major issue in combining two languages is the need for synchronization between sequences and processes. Using UVM-ML resolves several general issues such as phase synchronization, end of the test and objection. Nevertheless, there are many places where communication between the components must be established, i.e. scoreboards, monitors and sequences. In our current project, we used the scoreboard in the VIP (SV) to verify message arrival. This scoreboard was fed by the sequences (Comparing messages in Master and Slave). As mention above, the DUT side runs E sequences and the other side runs SV sequences. An additional synchronization is needed to delay one before starting to send traffic.

Synchronization can be done either by E methods port or by TLM ports. The two methods use the C interface of the simulator, to access different objects. We used E methods mainly for legacy reasons, but TLM is can be used just as well.

Table 1 - Translate/Rewrite/Co-languge comparison

#	Aspect	Notes	Translate	Execute Memory Port	Re-write from spec	Write in C
1	Coding Effort	Of Sequences	MED	LOW	HIGH	HIGH
2	Env Effort	Test flow	LOW	HIGH	LOW	LOW
3	Tests Debug Effort		MED	LOW	HIGH	HIGH
4	Preserve Test Suit	Run the exact Vendor scenario and checks	HIGH	FULL	LOW	LOW
5	Flexibility	Running a different scenario. Post silicon debug	LOW	LOW	HIGH	MED
6	SW reusability	Using sequences as a reference to SW	LOW	LOW	MED	HIGH

A. Using Method port for synchronization

Here is a code example of a method that is executed in SV after being called in Specman E :

E Code:

```
method_type wb_project_set_name(name: string);       // Definition
…..
i3c_set_uvm_test_name: out method_port of wb_project_set_name is instance;
keep i3c_set_uvm_test_name.hdl_path()
=="~.dte_board_taa0.get_uvm_test_name_from_e";    // linked to SV

Test_name: string;
i3c_set_uvm_test_name$(test_name);                    // Called in E
```

SV Code:

```
function void get_uvm_test_name_from_e(string name); // Executed in SV
        sv_test_name = name;
        $display($sformatf("I3C DTE Test is %s",sv_test_name));
endfunction
```

“Figure 6 - Synchronization of events and data” shows the usage of methods port for synchronization and data delivery between modules of different languages.

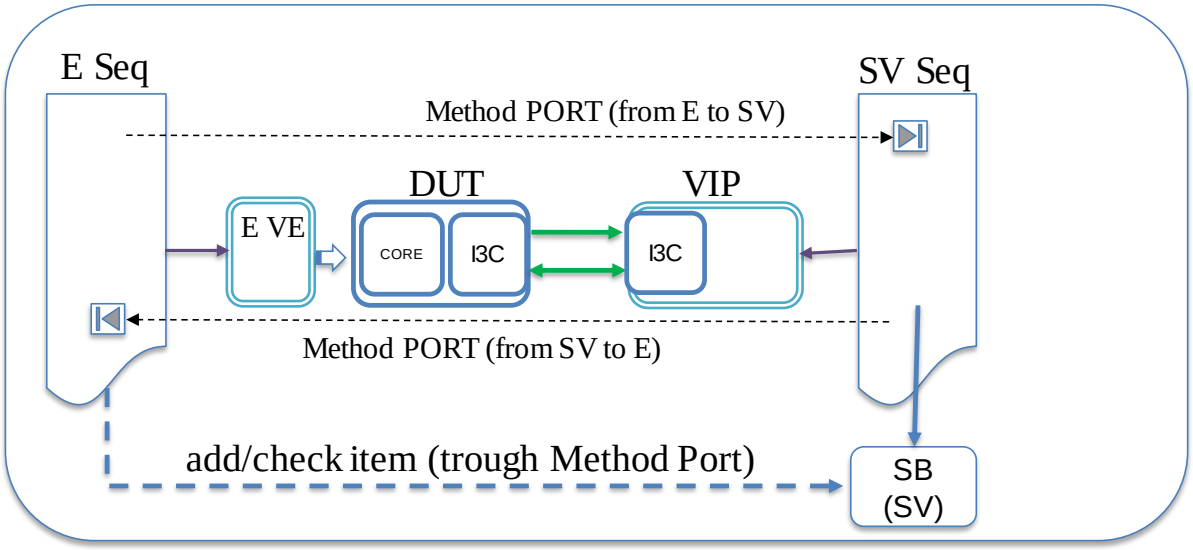

Figure 6 - Synchronization of events and data

V. CONCLUSIONS

Combining a component with a different language into an existing verification environment depends on the particular characteristics of the system and the verification requirement. In some cases, partially translation can be more efficient and time/cost-effective than applying a standard language-to-language porting. We found that for sequences, particularly for those which produce memory access command, translation of the code is beneficial.

Using UVM-ML OA is a valuable way to connect and run together components with different languages. Yet, the implementation of these kinds of systems requires special attention for synchronization. A TLM between the components should close this gap and enable effective verification.

REFERENCES

[1] Accellera, UVN-ML Open Architecture version 1.11

[2] B. Sniderman, Vitaly Yankelevich;" Multi-Language Verification: Solutions for Real-World Problems" (2014); DVCon, San Jose,CA