Ultimate Shift Left: Unleash the Power of UVM Virtual LAB Methodology upon SoC Verification

Roman Wang
Advanced Micro Devices, Inc. Shanghai R&D Center
Roman.Wang@amd.com,

***Abstract*- The project execution schedule is the one of the important factors for a project to be successful in the market. The challenges of SoC verification execution typically come from the complexity of design and flow, stability of the database, resources, reuse, simulation performance, debugging turnaround time, and so on. Therefore, how to meet the schedule with high verification quality is always important for verification engineers. In this paper, we promote a new practical Universal Verification Methodology (UVM) virtual laboratory (vLAB) methodology to address these challenges. It supports the requirements of reusability, scalability, and flexibility. Powered by it, the IP/SoC verification teams gain great benefits from finding issue, reporting for fix early and quick bring up. It has been widely adopted in many SoC projects for more than two years, and has successfully demonstrated a 'shift left' to pull in schedules with high quality.**

I. INTRODUCTION

A. Motivation

In general, verification jobs start from IP or block level. The IP team creates the UVM stand-alone environment to qualify the IP features. Eventually, it's expected to reuse the IP level random constraints stimulus and passive testbench without any change at a higher level (say subsystem or SoC level) to qualify the interesting or necessary data flow. Eventually, we can create high level UVM tests and concurrently run multiple different IPs' reusable tests to stress the bus. This can greatly reduce the burden on the SoC verification team. To execute the SoC project, we usually define several important milestones with specific quality criterion to ensure the schedule. For example, at **project phase one**, the initial SoC database can be ready for IP integration. At that time, the new design may have more than 40% features be implemented, or the derived design is almost implemented. IP integration to SoC need to follow a semi-pipeline rule that means a high priority or high layer IP executes the integration earlier than others. This may usually take weeks because of the stability of the SoC database. The IP team can only qualify the SoC database flow when its integration is done, because the upper layer design complex may not be ready. At **project phase two**, the IP is integrated 80% of the implemented features into the SoC. First, the IP team must qualify the SoC database quality again in one or two weeks. When the SoC verification team delivers the workable test bench, the IP team can start to qualify the reset release sequence and initialization for the SoC design complex in three to four weeks. The initialization is the key to ensure the downstream register access and upstream DMA traffic from the high layer to the IP ports through the SoC design complex. By reusing the IP level sequences at a high layer, the first register access can be tested. It is generally difficult to get workable even given several workaround forces because some design features are still not ready yet or buggy at higher levels of complexity. Another reason is the IP team may lack understanding of the higher level design complexity. At **project phase three**, the IP integrates 100% of the featured design into the SoC. The IP team must do the same job again as in **project phase two.** If both the database and the initialization are good with the removal of past workarounds, the IP team can do a real IP feature verification in SoC. The first sanity test usually is workable in one month after the **project phase three**, then the engineer can do more tests for qualification.

B. Challenges

- ✓ Complexity of the SoC design, environment, and flow. It usually takes a long time to clean such issues before integrating the IP into the SoC.

- ✓ Stability of the SoC database. Any bad submission or IP integration may break the SoC database due to dependencies which are challenging to be fully qualified across many teams.
- ✓ IP integration has semi-pipeline rule in the SoC. Before the IP integration is done, you cannot qualify a SoC database for your IP integration. The real IP functional verification will install until the SoC database is qualified.
- ✓ Lots of integration re-spins between the IP and the SoC. For verification, the reuse issue is important. IP-level verification cannot fully consider the reuse for SoC. Re-spins may take days because of merging overhead or simulation time. The challenge is that there is no chance to qualify the IP reuse before its integration in the SoC.
- ✓ The bring-up of IP verification works as the serial mode in the SoC. The IP team must qualify the upper design complex to ensure that the related data path is workable, then start to qualify the IP functional features.
- ✓ Simulation performance and debugging turnaround time. The release of verification changes may take a long simulation time and easily conflict with others. For example, the reset release and initialization sequence for the upper level bridge design complex may take 50% of the whole simulation time (6~8 hours per test).

The big challenge is that we do verification very late in the process due to many constraints. To improve, we raise questions such as:

- □ How to decouple the verification complexity?
- □ How to reduce the verification dependency?
- □ How to bring up verification environment in parallel?
- □ How to reuse design verification (DV) stimulus and test benches without any change?
- □ How to reduce or eliminate the re-spins because of the reuse?

In a word, how can we do things a little earlier and better?
To address the challenges above, we suggest a new practical UVM virtual laboratory (vLAB) methodology and present the detailed concepts and our successful story in the following sections.

II. PROPOSED SOLUTION

A. The Concept of UVM vLAB Methodology

The original idea was inspired by the personal computer (PC). There is a powerful motherboard and several daughter cards attached on specific sockets with plug-play support, and the cards can also be attached on the different motherboard if the sockets are the same. The basic input output system (BIOS) can take self-testing for hardware at power on stage, then the operation system (OS) can move on if everything is fine. In the SoC level, the IP team wants to verify the interesting or necessary data path based on the SoC database. We decouple the selected IPs from the SoC per verification requirement and set other unnecessary IPs as the shell to build several subsystems. We name such subsystem as Combo Whacker (CW) Verification [3]. For example, the register and DMA data path from the IP to high-level masters are bridged with register fabric, data fabric, and bridge design complex. We want to reuse the IP-level UVM stimulus to stress the data path and make the testbench passive to check including coverage collection. Te multiple IPs' reused sequences can be created into multiple IPs' concurrent stress scenarios. The challenge is that the SoC level testbench wants to provide a flexible solution and make our stimulus work on the selected designs, but verification methodology is a little different between the IP and SoC.

Therefore, we define a UVM vLAB methodology to make the SoC CW verification life easier. The vLAB architecture is looking like a complex laboratory which is a highly configurable, scalable UVM framework. It contains several different types of UVM Verification IP (VIP), reusable base testbench and generic UVM abstract layering. It is also powered by our UVM verification solution library (including [3] [4] [5] [6] [7] [8] [9] [10]) to handle the shared verification challenges across multiple IPs and SoCs. The vLAB methodology ensures the IP to SoC reuse without a gap, especially the reuse of UVM, C++ or mixed tests (for example UVM/C++). It also supports multiple production lines by flexible configuration. The Fig.1 shows the plug-play concept for SoC verification. The SoC database, environment, and flow represent the motherboard. Different IP represents the daughter cards, and they are integrated into the SoC motherboard though bus interfaces which represent the sockets.

❖ The concept of plug-play has two categories:

➢ Design Point of View

- Motherboard (SoC RTL database), daughter cards (IPs).
- Configure the IP as RTL view to represent plug-in design under test (DUT).
- Configure the IP as Shell view to represent plug-out DUT.
  - The shell should drive lower strength value by default to avoid the X propagation. The UVM VIP will drive the port of shell in the testbench.

➢ Verification Point of View

- Mother board (the SoC level Verification database and flow).
- Daughter cards (the IP level UVM environment, including Testbench, Register Abstract Model (RAL), UVM memory model, sequence, test, etc.).
- Daughter cards are attached on DUT via System Verilog (SV) interface binding methodology.
- Configure daughter cards as passive mode to represent the plug-out.
- Configure Daughter cards as active mode to represent the plug-in.

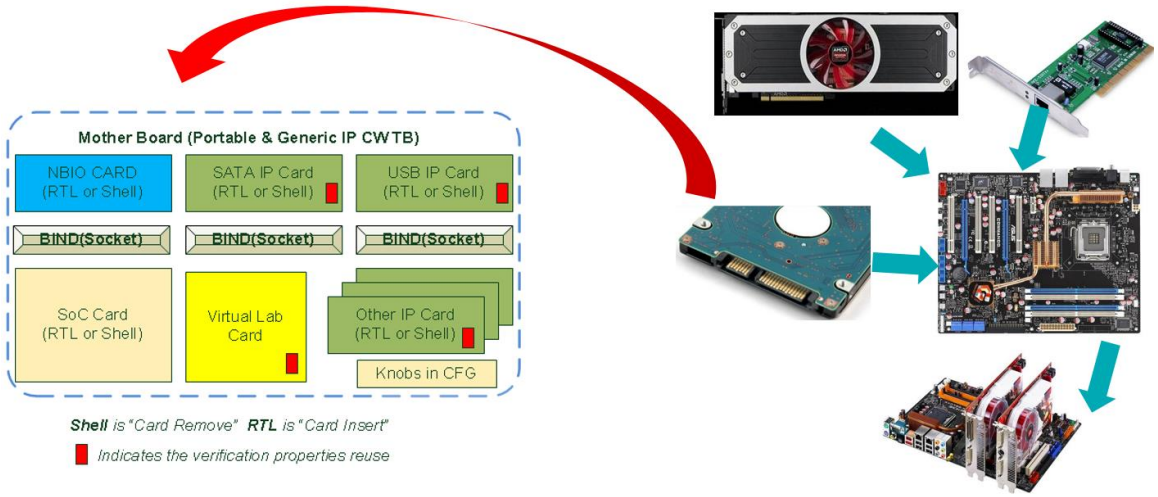

Fig. 1. The different methods to access the registers by using RAL.

❖ Evolve the serial crawl into the parallel fast run

In the past, the first IP sanity test should bring up as serial mode. Firstly, we have to qualify the upper layer design complex in the SoC, then we can start the IP functional verification. The serial mode has a high dependency, complexity and need great effort with many re-spins for a long time. If upper layer design is not fully implemented yet, the IP test must be installed even though the IP is already well integrated in the SoC database. To resolve this challenge, the vLAB defines three bring up modes to make the verification tasks parallel in Fig. 2.

➢ The vLAB Upper Link Mode

The intent is to bring up the data path of the upper layer DUT where the IP is attached. The targeted IP will act in shell view. The vLAB can provide the UVM VIP to drive the high-level design and response on the IP shell. When the SoC database is ready for IP integration, we can use the vLAB built-in smoke test to qualify the SoC database, environment and the UVM test flow. It runs daily to monitor the database quality. If the upper layer design is already integrated in the SoC, we can start to qualify the high level design about the SV interface binds, reset release

sequences and bridge design complex initialization at the project phase one stage. In this way, we can early find potential issue, and early report for fix.

- The vLAB IP Present Mode

The intent is to bring up the IP design and verification quality in the SoC database. Especially, it's for different connection between the IP standalone and the SoC. In this mode, the upper DUT is configured as the shell view. We can run all of the IP level tests similarly on the active IP standalone. We can start to qualify this mode at project phase one. People may get confused about the difference between the vLAB IP present mode and the IP standalone, because they are looking very similar. Here is a comparison:

1. IP Standalone
   - It is to verify comprehensive IP features in the IP UVM standalone environment and flow.
   - It may have a top Verilog wrap to integrate IP sub-modules and bind SV interfaces to an IP module or a top Verilog wrap (even it is not reusable).
2. vLAB IP Present Mode
   - It is based on the SoC design complex, database, flow, and verification environment.
   - The IP design module tile integration may change from the IP to the SoC.
     Ex. we have a top module to wrap all module tiles in the IP level, but they are flattened at the SoC.
   - There is possible standard functional cells inserted between IPs by flow in the SoC level.
   - The IP-level flow may not work at the SoC, it needs to tune and debug.
   - The IP UVM stimulus and testbench need to tune for reuse at the SoC level.
   - The IP level SV interface binding may be different between the IP and SoC.

The vLAB IP Present Mode is an earlier, well-prepared and inevitable step to build the vLAB full link mode.

- The vLAB Full Link Mode

The intent is to bring up the full data path including the upper level DUT and the IP, both of them are in RTL view. It is our final goal. If everything goes well, we can try it at the project phase two.

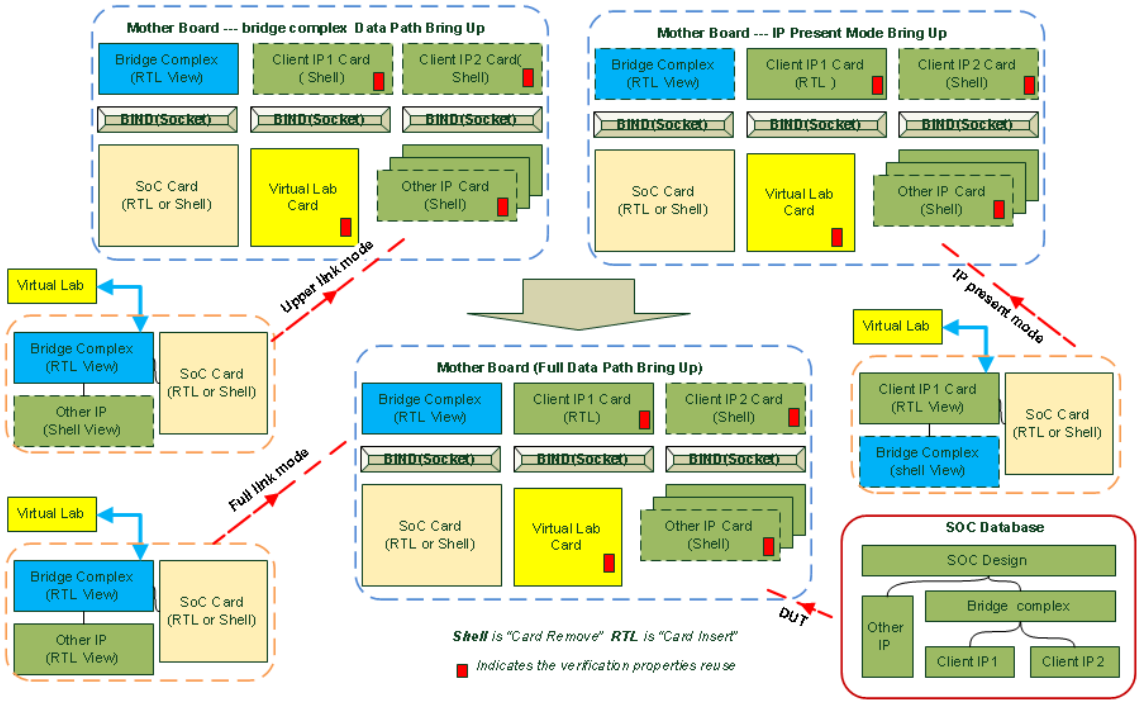

Fig. 2. Different vLAB Modes.

In principle, the vLAB Upper Link Mode can work earlier than the vLAB IP Present Mode, as well as in parallel with the vLAB IP Present Mode when the IP integration is done. At this point of the process, we have improved several issues for bringing up. However, the IP integration re-spins overhead is still out there. The IP mainline always keeps moving forward after branching out for IP integration. If the engineer integrates the changes from the IP branch to the IP mainline, he might really suffer due to many conflicts to be merged and it may take days to carefully resolve. The SoC also keeps moving forward, when the engineer works on re-spins back. He has to re-integrate the updated IP branch to the SoC, and he has to pay for the integration time and debug effort again. To address this challenge, we suggest a practical solution in the next section.

❖ UVM Virtual Combo Whacker (vCW) Prototyping Architecture

When the SoC project kicks off, the SoC and IP teams are starting to build their verification environment. However, the IP team cannot qualify any verification for the SoC-level reuse in advance. To make good use of the time window between the IP- and SoC-level verification, we create several virtual UVM models to support Virtual Combo Whacker (vCW) Prototyping and qualify the SoC-level reuse of the IP early.
We introduce two new bring up modes to support the vCW concept:

➢ The vLAB IP Virtual Full Link Mode

When the SoC database is not ready, the infra team (handles the tool and flow) helps to create a simple environment to mimic the SoC required flow (this takes a low effort to create and maintain, and can be deprecated when the SoC database is ready). We use the SV binding approach to build a dummy DUT integration prototype for the required data path as the same hierarchy of the SoC definition, and most of the IPs are in shell view. We do not want to model the full features of design, but just ensure that the data path is basic workable. The targeted IP is also integrated as RTL view. Of course, you can integrate more IPs. In the UVM vCW Prototyping Architecture in Fig. 3, we model the high-level abstract features of the bridge design complex, system control fabric, IP clients by using UVM model, or VIPs. (e.g., indirect register access and shadow register access). The virtual model is configurable to generate the different architecture layout per project needs, and looking like a 'virtual verification transformer'. In this way, we can qualify the reuse before integrating the IP branches to the SoC. The experience is shown to reduce more than 90% re-spins efforts at the project phase one stage. When we find the reuse issue, we can directly roll the changes in the IP mainline. Eventually, we ensure the IP branch quality for the SoC-level reuse. It meets the goal to implement earlier, and communicates with the other teams earlier.

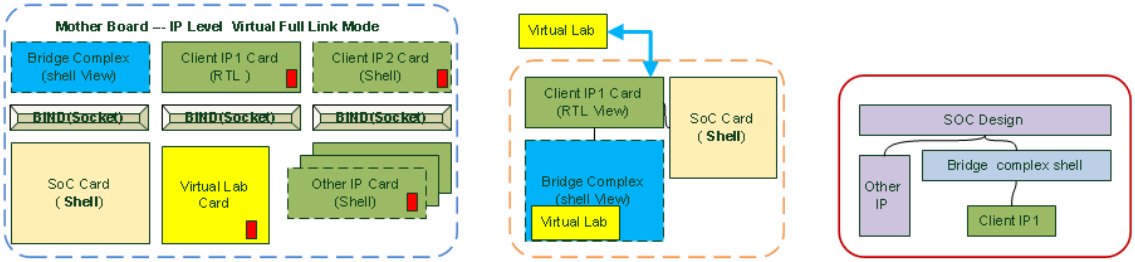

Fig. 3. The vLAB IP Virtual Full Link Mode for IP1.

➢ The vLAB SoC Virtual Full Link Mode

When the SoC database is ready and the upper layer design complex is not fully functional at project phase two, we can replace the DUT with vLAB virtual UVM models as shown in Fig. 4. Then, we can simply reuse the IP stimulus on a high layer to drive an interesting data path. When the upper layer design complex is implemented well, all UVM virtual models are reconfigured as the passive mode for checking and coverage collection.

The only difference between vLAB Full Link Mode and vLAB Virtual Full Link Mode is that we adopt the virtual UVM model to replace DUT. Based on the five vLAB bring up modes, we can decouple the verification complexity and perform the verification work in parallel. In the best case, no one will be stalled by others.

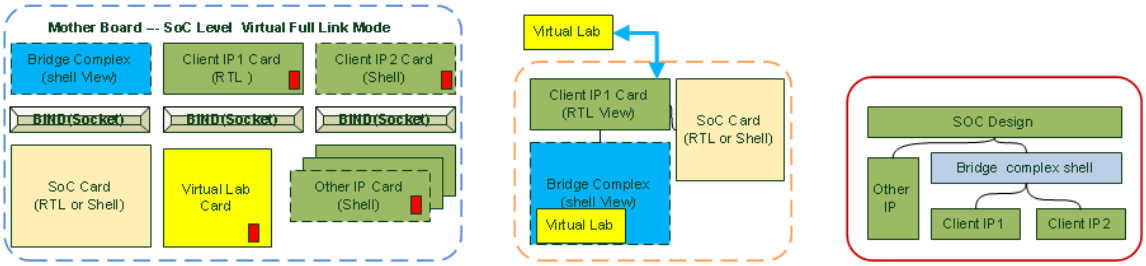

Fig. 4. The vLAB SoC Virtual Full Link Mode for IP1.

❖ The Next Level Shift Left: Efficient Execution Workflow

As the Fig. 5 shows, we define an efficient execution workflow. It is divided into three development phases:

- The IP Development Phase:

  The focus is on the vLAB and IP mainline development in parallel. The vLAB can start earlier than the IP development, because of its common framework.

- The vCW Development Phase:

  The focus is to build a simple environment with the same flow as SoC, integrating the vLAB and IP mainline. The vLAB IP virtual full link mode will start at this phase.

- The SoC CW Development Phase:

  - SoC startup CW stage:

    The focus is to qualify the SoC database, and the DV flow by using vLAB built-in smoke tests.

  - Separated IP CW stage:

    The IP is integrated into the SoC. The focus is to qualify the vLAB upper link mode, vLAB IP present mode, vLAB IP full link mode, and vLAB SoC virtual full link mode.

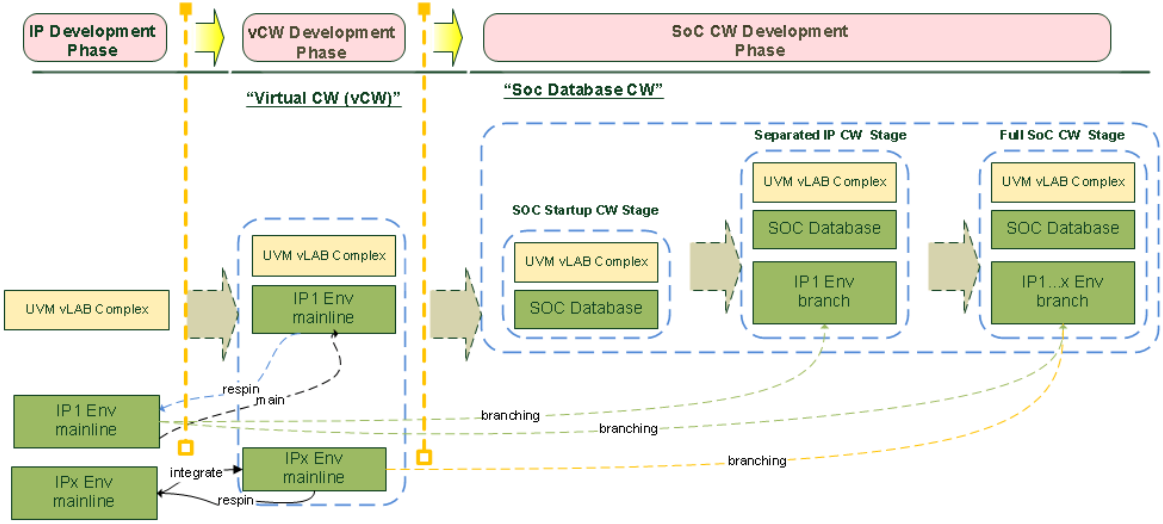

Fig. 5. The vLAB Execution Workflow in Real Projects.

  - Full SoC CW stage:

    The more IPs are integrated into the SoC. The focus is the same as the separated IP CW stage, but we can try the multiple IPs stress test by reusing the IP-level sequences or tests.

❖ *Scalable vLAB UVM Abstract Layering for IP/SoC Reuse*

To ensure the IP level testbench, sequence and test reuse at the CW level without any change, we define a scalable vLAB UVM abstract layering in Fig. 6.
The IP-level tests extend from the IP test base, which integrates the IP module UVC, RAL model, etc. The IP TB configure class controls the IP module UVC.
On the CW-level, The vLAB builds its base test based on our UVM solution library and integartes the UVM complex UVC. We also developed the vLAB smoke test suites inside the vLAB package. The vLAB itself is designed for common requriements and shared by all of the IPs. When a specific IP is connecting to the vLAB, we define an additional layer as the IP CW test base. It allows the user to override the IP-level verification code per CW requirements. As we can see, the IP testbench configure class also extends the IP CW configure. For example, we can override the root path string in the configuration class. In the IP-level, the root path may be "ip" as its module name. In the CW-level, the root path should be changed to the SoC level hierarchy "top.*.*.*.ip_inst_name". The UVM sequence works with the same consideration as the UVM test and configuration class.

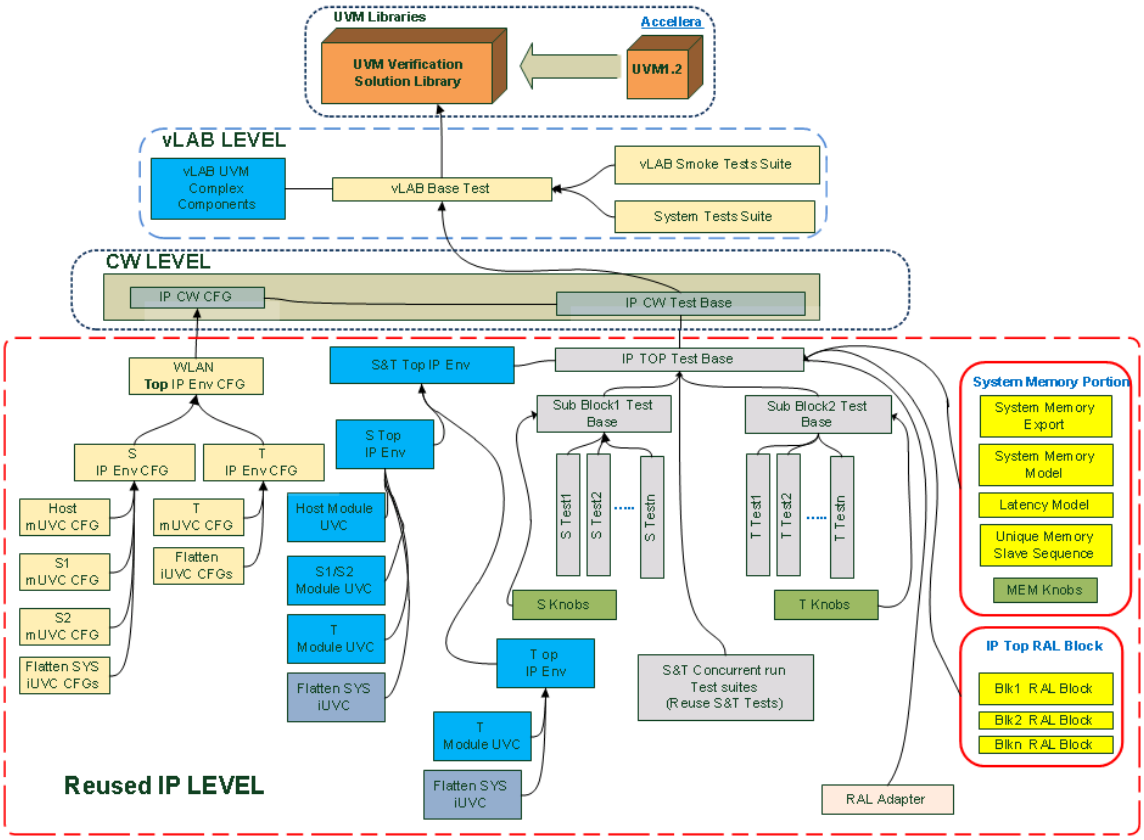

Fig. 6. Scalable vLAB UVM Abstract Layering.

The UVM does not support multiple inherits, so we use the macro to identify the parent class on different verification levels. The code snippet is shown in Fig. 7.

```
1. In IP top test base
`ifdef IS_CW_P
class ip_top_base_test extends ip_cw_base_test;
`else
class ip_top_base_test  extends verf_slib_reset_aware_base_test;  // Verification solution library
`endif

2.  In IP top sequence base
```

```
`ifdef IS_CW_P
class ip_top_base_sequence extends ip_cw_base_sequence;
`else
class ip_top_base_sequence extends verf_slib_phase_aware_base_sequence; // Verification solution library
`endif
```

Fig. 7. Code example.

- *Practices to Accelerate the Simulation and Debug.*

Here are some tips to improve verification and debug productivity.

- ✓ Adopt the partial compile to reduce recompile time.
- ✓ Adopt the save restore technology of EDA tool, it can bypass the reset release and SoC initializing for the test to save almost 30% simulation time.
- ✓ Write the reset release sequence and the SoC initializing register programing in a file, then vLAB will have a free running sequence to read the file line by line and decode into a bus transactions to broadcast. This method is similar as the CPU reads the instruction from memory and decodes for execution. It can greatly save the turnaround debugging time when we tune the reset release and SoC initializing.
- ✓ Disable the CDC and OVL check during bringing up the test at an early stage.

III. CONCLUSION

In this paper, we describe a new practical UVM virtual laboratory (**vLAB**) methodology to address the SoC verification challenges. It supports the requirements of reusability, scalability, and flexibility. Powered by it, verification teams gain great benefits from the 'early to find issue and early to report for fix' and quickly bring up. It has been adopted in multiple SoC projects for more than three years, and has been successful in achieving a 'shift left' to pull in projects' schedules.

Projects/IP	New SoC (First adoption)	1st derived SoC	2nd derived SoC
IP1 vLAB	Tune 1st sanity pass > 4 weeks @ **project phase two**	Tune 1st sanity pass 2 weeks @ **project phase two**	Tune 1st sanity pass 1 week @ **project phase two**
Regression	Regression pass rate > 80% > 4 weeks after sanity pass	Regression pass rate > 80% 2 weeks after sanity pass	Regression pass rate> 80% 1 week after sanity pass
IP2 without vLAB	Tune 1st sanity pass >5 weeks @ **project phase three**	Tune 1st sanity pass 3~4 weeks @ **project phase three**	Tune 1st sanity pass 2~3 weeks @ **project phase three**
Regression	Regression pass rate > 80% > 6 weeks after sanity pass	Regression pass rate > 80% 4~5 weeks after sanity pass	Regression pass rate > 80% 3~4 weeks after sanity pass

ACKNOWLEDGMENT

We would like to thank Roman's wife (Liangliang Li) for her continued support and the AMD team (Michael Jiao, Tengfei Gao, Lifeng Chen, and Zhi Wang) for executing this solution in a real project.

REFERENCES

[1] IEEE 1800.2-2017 UVM.

[2] IEEE 1800-2012 SystemVerilog.

[3] Roman Wang, Thomas Bodmer, "The Art of Portable and Reusable UVM Shared System Memory Model Verification Methodology across Multiple Verification Platforms: UVM IP Stand-Alone, ComboWhacker, Virtual FPGA and SoC Full Chip", DVCon USA 2016.

[4] Roman Wang, Uwe Simm, "Thinking Beyond the Box: Adopt the Reusable UVM Thread Management and Customized UVM Reset Package to Attack Thread Aware Verification Challenges", DVCon India 2015.

[5] Roman Wang, Thomas Bodmer, “Wrapping Verilog Bus Functional Model (BFM) and RTL as Drivers in Customized UVM VIP Using Abstract Classes”, DVCon USA 2015.

[6] Roman Wang, “Practical experience in automatic functional coverage convergence and reusable collection infrastructure in UVM verification”, DVCon Europe 2014.

[7] Roman Wang, Uwe Simm “A New Epoch is beginning: Are You Getting Ready for Stepping into UVM-1.2?”, DVCon India 2014.

[8] Roman Wang, Uwe Simm, “Making UVM Verification life Easier: UVM Debug Capabilities”, CDNLive China 2013.

[9] Roman Wang, “A comprehensive approach to scalable framework for both vertical and horizontal reuse in UVM verification”, CDNLive China 2012.

[10] Roman Wang, “UVM SDMAM technique for system level SoC Verification”, CDNLive Silicon Valley 2012.