Programming Model Inheritance and Sequence Reuse

Aji Varghese
Texas Instruments Incorporated,
12500 TI Blvd, Dallas, TX 75093
aji@ti.com

As the systems become more and more complex, the importance on reusability of DV test cases and library were never so prominent than now. This paper describes one such methodology that allows easier and faster porting of DV sequences and libraries between IP to SoC to Software to SiVal. This methodology automates the spec capture in C language at the base level and builds higher level abstraction sequences based upon the functional requirement of the IP. These higher level sequences can be used seamlessly by SoC or Software teams to create real time use case scenarios without going in detail of the IP functionalities. Since most of the library is built without manual intervention, the libraries are correct by construct. This methodology is used widely in IPs and SoCs resulting in estimated 5X effort reduction.

I. Introduction

Programming sequences form the layers above basic register read/write operations in the SW stack (Figure 1). They are split into register programming sequences, functional sequences and user sequences. The register programming sequences primarily groups several register/bit-field operations together to provide atomic functions to configure IP to desired operation modes. The functional sequences combine register sequences and/or register accesses into functional operation – this is the layer which exhaustively abstracts the functionality of the HW. User layer/sequences which may be present on top of functional sequences provides simplified interface to the HW defaulting most of the parameters

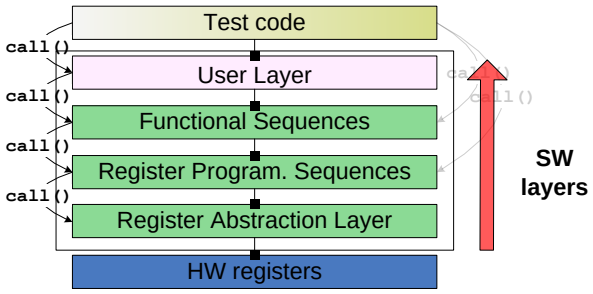

Figure 1: IP SW Stack - layers

This paper proposes multiple ways about how the programming sequences description can be utilized to help reuse the DV collateral across multiple platforms. In the process of creating function libraries from Architecture teams for various groups within a particular IP or SOC, it becomes necessary to produce a means of description that is simultaneously concise yet readable. A combination of internal and third-party tool is used to extract the information from the spec to sequence abstraction layer.

II. Prior Art – Why We Need This

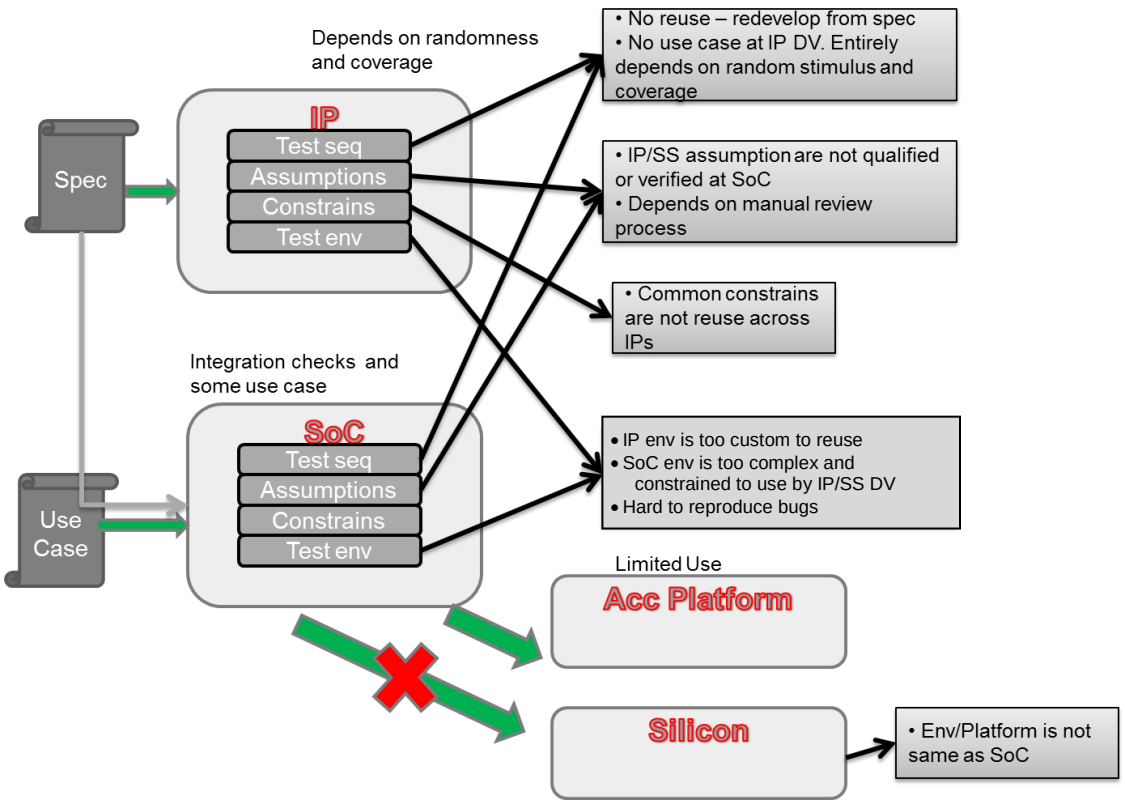

Figure 2: Existing methodology

Current situation - redundancy: As shown in Fig 2 each team starts with reading functional spec to find how to program registers to configure IP/SoC device to certain state. The teams are:

- IP verification
- SoC verification
- SoC validation
- SW development

Can we avoid the redundancy?
Can we gain time needed to develop SW in each team?
Can we avoid misinterpretation in each team?
Can we decrease load to architects to clarify programming model?
The answer: Yes we can! … but How?

III. Proposed Solution - Overview

The scope of this paper is to describe the definition of programming model from IP functional spec and allows reuse of the SW across multiple teams.

The programming model is specified by a set of formally defined programming sequences in functional specification. Each configuration is represented by unique register programming sequence - sequence of register writes, reads, polls and assertions.

The sequences are used and refined by

- IP verification
- SoC verification and validation
- SW development

The programming sequence IS sequence of register read/write operations

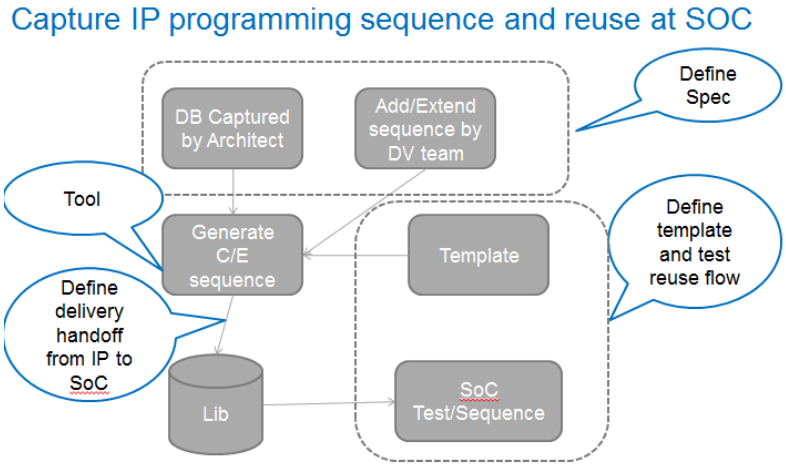

Figure 3: Captue IP programming sequence flow

The sequences are created from the database captured by the architect. These sequences are then reused across multiple IP, SoC and SiVal teams.

IV. Programming Model and Sequence Reuse

The IP SW stack can be categorized as:

- Component based
- Can use component parameters defined on instantiation.

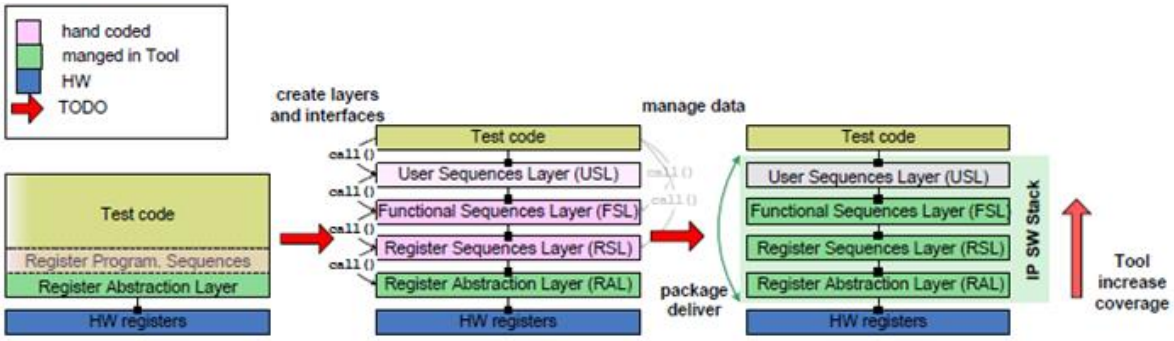

Figure 4: IP SW stack mapping with SW library component.

As shown in Fig 4 the IP SW stack consists of multiple layers such as:

- **Register abstraction layer**: Abstract register to allow various approaches to access the registers.
- **Register sequences**: These are a 'small' size sequences providing atomic configurations allowing easier HW changes.
- **Functional sequences**: These sequences combine multiple register sequences to provide access to IP functionality.
- **User Layer sequence**: This layer further abstracts the functional sequences by hardcoding a certain sequence, hiding the configuration and providing typical or default value.

The register abstraction layer sequences and register sequences are auto generated from a mix or internal and third-party tool hence there is no manual effort in creating. This also helps in removing any manual mistakes that might occur in hand-coded sequences.
There are certain guidelines to be followed to extract the sequences from register definition in IPXACT using internal and third-party tool.

The sequence generation and reuse responsibilities are spread across multiple teams which is shown in figure 5

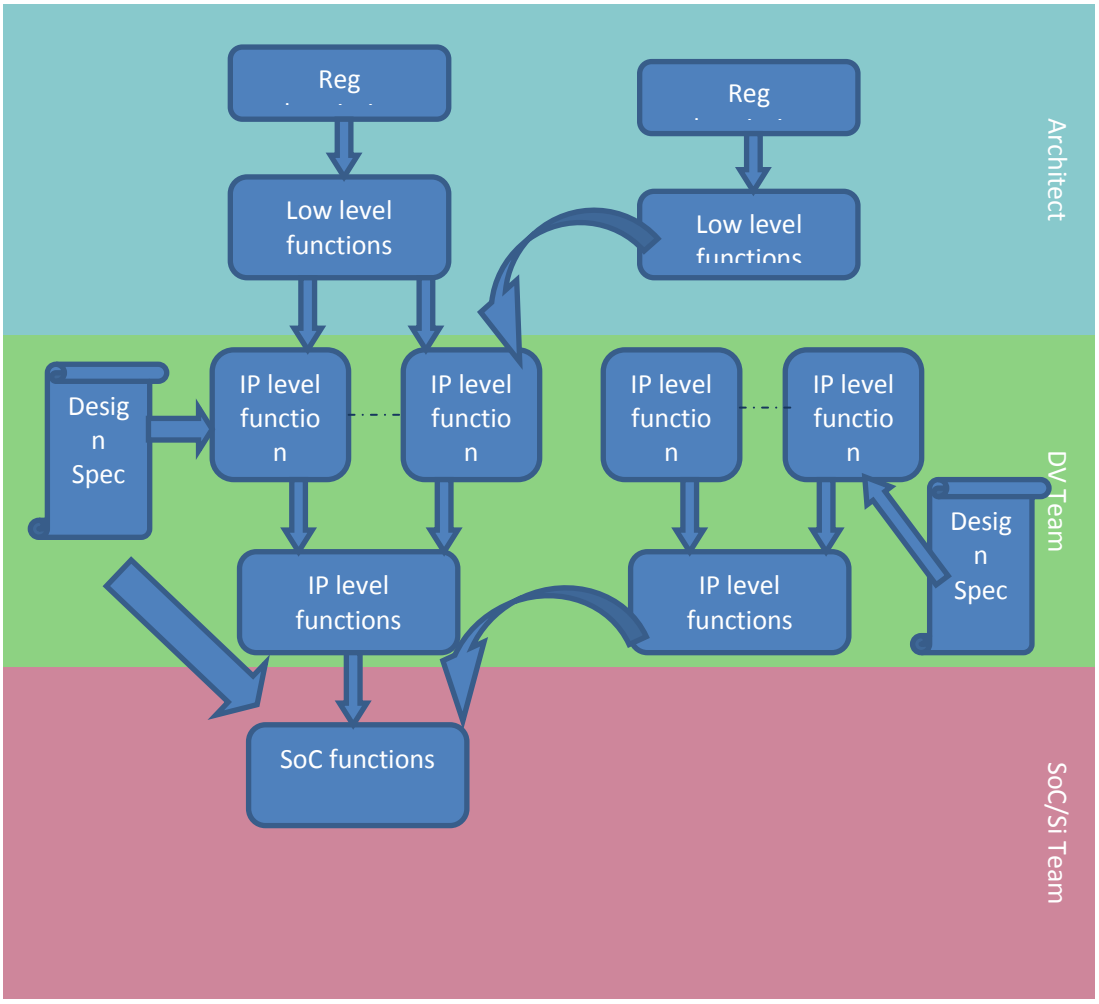

Figure 5: Sequence Reuse Flow across multiple teams

Responsibilities of:

Architect

- Use the register function to build high level functions
- Capture use case sequence.

IP DV Team

- Create higher level functions using the functions defined by Architects.
- Verify and qualify the generated IP DV team sequences

SOC DV Team:

- Create tests using higher level functions defined by Architect and IP DV team. This could include functions from multiple IPs

- Define any additional functions that are specific to an implementation. Such as ordering requirement, PLL, interrupt controller etc.

V. SEQUENCE CAPTURE

Combination on internal and third-party tool is used for writing the Programming Sequences. Figure 6 shows the GUI interface which helps to code the Sequences without errors. It is used to view register values and program fields accordingly (description of memory maps/registers/bitfields).

IP_CONF_REGSET_0_7

Ol	name	addressOffset	dataWidth	accessType	absoluteAddress	description	dim	displa	readT	resetMask	resetValue	value
	dim											8
	IP_REG1	0x0	32	read-write	0x180	The bank of reg It is used for AF Shadow registe	1 (parameter)			0xffffffff	0x0	
	IP_REG2	0x4	32	read-write	0x184	The bank of reg It is used for AF Shadow registe	1 (parameter)			0xffffffff	0x0	

Figure 6: Register information capture

The output generated from this tool is the C function for a particular functionality. This C function uses the register definition to program the IP in a particular sequence.

The C function looks like as shown in figure 7:

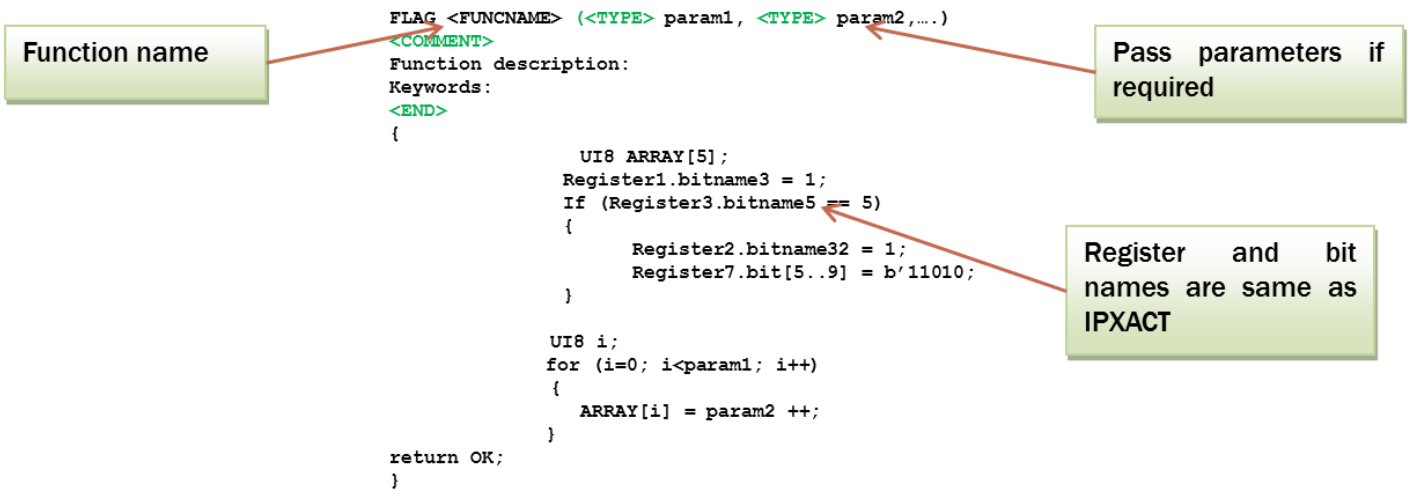

Figure 7: C function sequence example.

The sequence capture flow is shown in Figure 8. The registers are captured in a third party tool. These registers information is converted into IPXACT format [1]. The IPXACT formats are then exported to another tool from which sequences are generated.

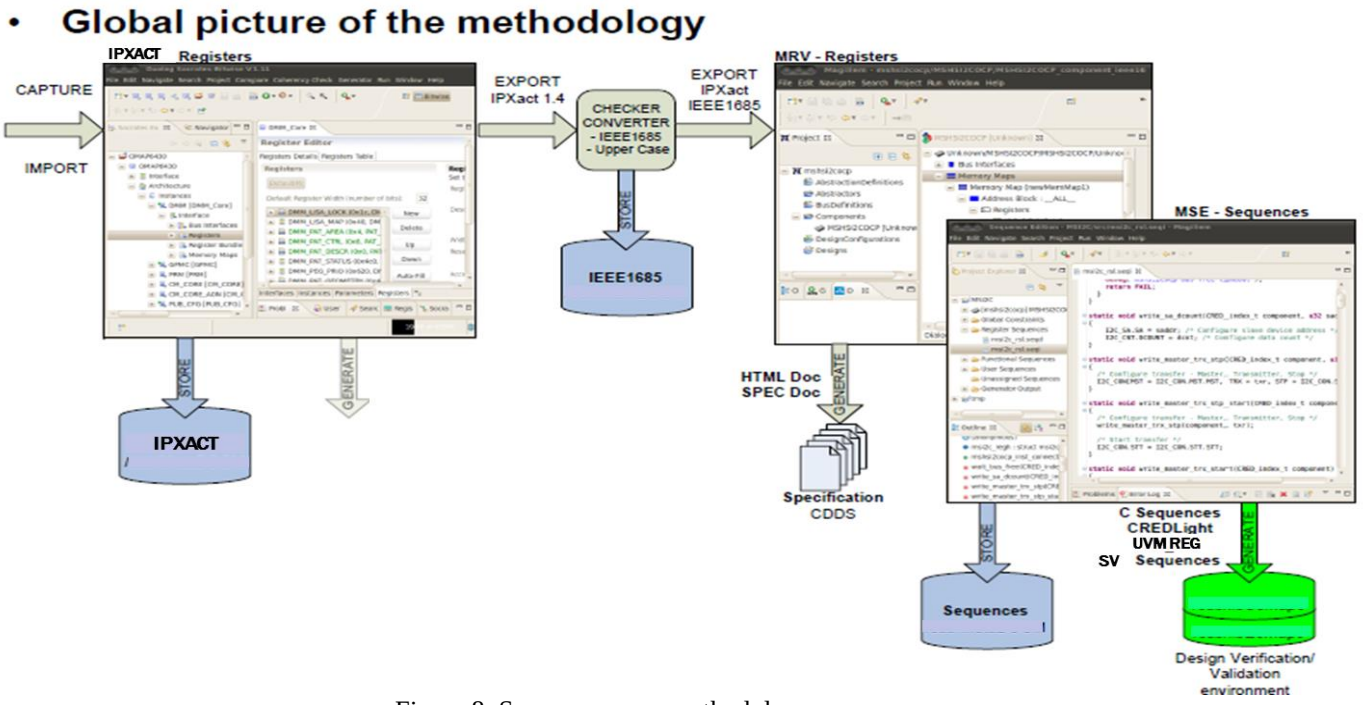

Figure 8: Sequence reuse methodology

VI. Sequence Usage Flow

Architects use the internal tool to generate the various sequences of a particular IP. The tool then dumps out two types of sequence.

1. Systemverilog (SV) sequences [2]:

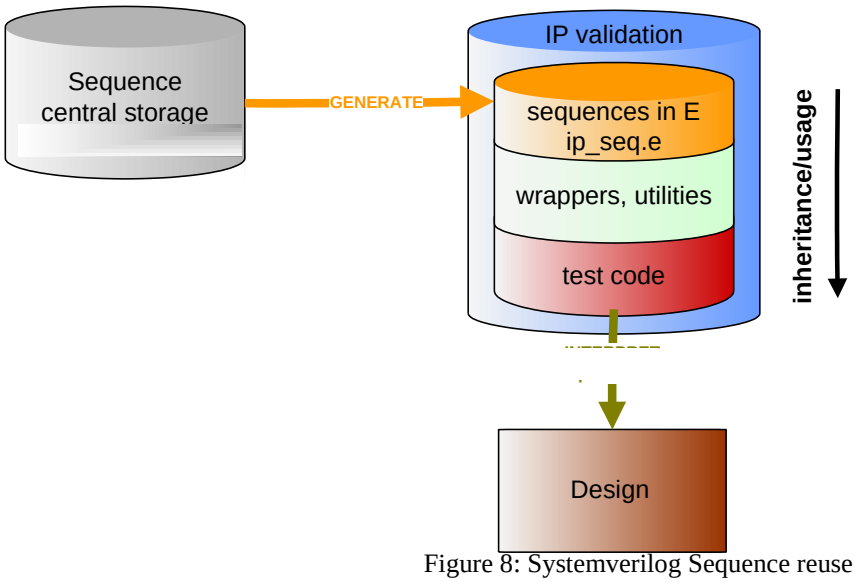

Figure 8: Systemverilog Sequence reuse

The SV sequences are used by IP team for complete IP level functional verification as shown in figure 8. These sequences are used in the UVM environment for 100% functional and code coverage also.

2. C sequences.

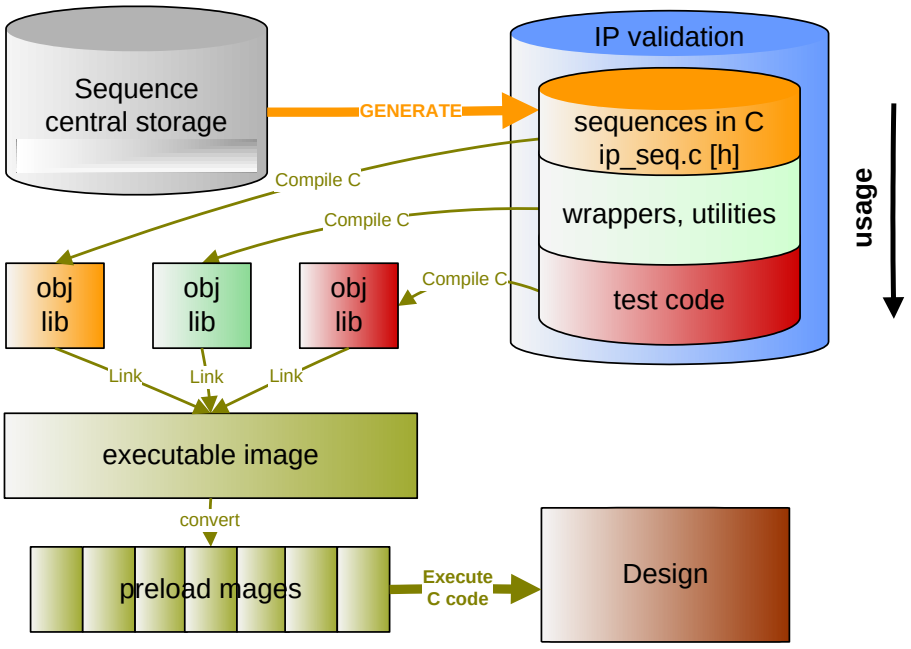

Figure 9: C Sequence reuse

The C sequences are first used by IP team to qualify and refine all the functional sequences. As shown in figure 9 the C sequences are compiled and linked using the processor compiler and converted to images that could be loaded onto processor.

The DV environment for C based verification at IP level verification is shown in Figure 10.

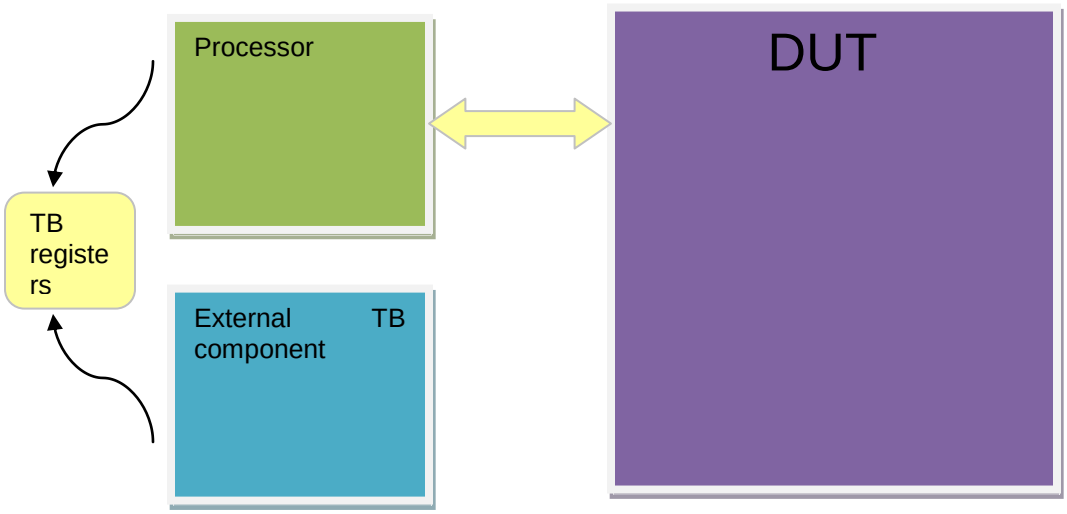

Figure 10: IP level verification environment.

IP DV test bench is created with two masters: processor or master BFM. The SV sequences are run on the master BFM and C tests are run on the processor. The type of master is auto selected by the environment based on the test cases configuration.

VII. RESULTS

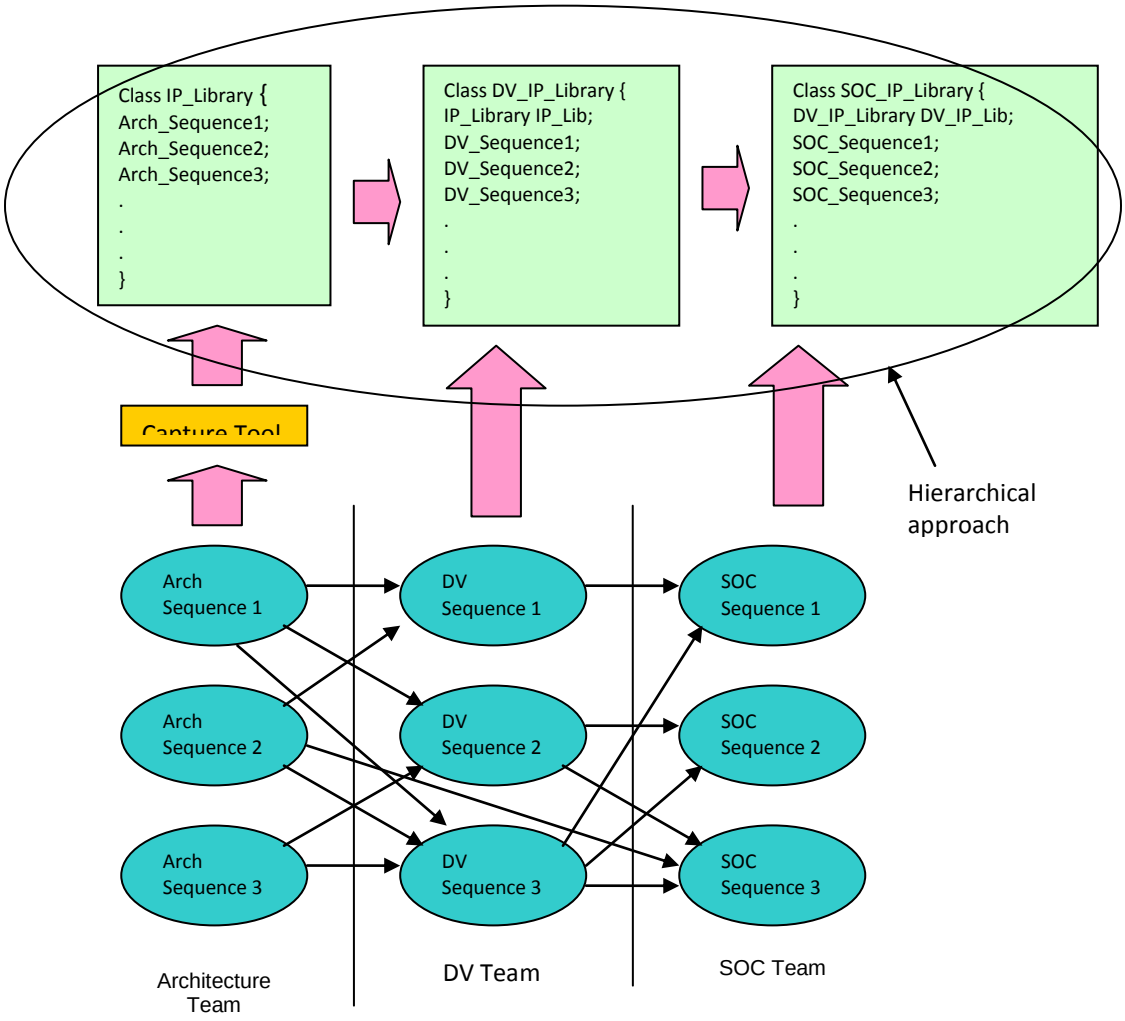

Figure 11: Sequences used across multiple layers.

- As shown in figure 11, the sequences developed at IP level by the architects are verified by IP team and used by SoC V&V team.
- SoC team need not understand the detail of each IPs programming sequence to create the tests.
- Application level tests are created with ease as the tests imports all base level sequences from multiple IPs.

VII. CONCLUSION

- Cost reduction - manpower to read and understand specs and implement code by multiple teams is eliminated.
- No redundancy.

- Correct by construct at SOC DV: Sequences are pre verified at IP level.
- Full reuse from bottom-up. Sequences are used from IP-SS-SOC-silicon platforms.
- Less effort for test case development as test is combination of IP level sequences.
- This methodology is fully developed and used in IP DV cycle. There is an estimated 5X DV cycle time reduction using this methodology.

VIII. Limitations

- This methodology is targeted towards register-centric functional verification where multiple registers are initialized before starting data transfer.
- There is a small learning curve to understand the third party tools that capture and dumps out register information required to be used for this methodology.

IX. Abbreviations and Acronyms

- BFM – Bus Functional Model
- DV – Design Verification
- IP – Intellectual Property
- SiVal – Silicon Validation
- SOC – System on Chip
- SS – Sub System
- SV – System Verilog
- VnV or V&V – Verification and Validation

IX. References

[1] http://accellera.org/downloads/standards/ip-xact
[2] SystemVerilog 3.1a Language Reference Manual.