Scalable Integration,Verification flow for faster Soc Build Up

Major Project Report

Submitted in fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)

By

Bhumi Dave

(17MECE04)

Electronics & Communication Engineering Department
School of Technology
Nirma University
Ahmedabad-382 481

May 2019

Scalable Integration,Verification flow for faster Soc Build Up

Major Project Report

Submitted in fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

By

Bhumi Dave

(17MECE04)

Under the guidance of

External Project Guide:

Mr. Revanuru Murthy

Engineering Manager,

CIG,

Intel Technology India Pvt. Ltd.-BLR.

Internal Project Guide:

Dr. Akash Mecwan

Assistant Professor

E&C Engineering Department,

School of Technology, Nirma University

Electronics & Communication Engineering Department

School of Technology-Nirma University

Ahmedabad-382 481

May 2019

Certificate

This is to certify that the Major Project entitled **"Scalable Integration,Verification flow for faster Soc Build Up"** submitted by **Bhumi Dave (17MECE04)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Dr. Akash Mecwan
Assistant Professor,
EC Department,
School of Technology,
Nirma University, Ahmedabad.

Dr. N.P.Gajjar
Professor,
Coordinator M.Tech - Embedded Systems
School of Technology,
Nirma University, Ahmedabad

Dr. D.K.Kothari
Professor and Head,
EC Department,
School of Technology,
Nirma University, Ahmedabad.

Dr Alka Mahajan
Director,
School of Technology,
Nirma University, Ahmedabad

Certificate

Statement of Originality

I, **Bhumi Dave**, Roll. No. **17MECE04**, give undertaking that the Project Report on "**Scalable Integration,Verification flow for faster Soc Build Up**" submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Electronics and communication (Embedded System)** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me as part of on-going research work in Intel Technology India Pvt. Ltd. and I give assurance that no attempt of plagiarism has been made. It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

_____

Date:

Place:

Endorsed by

Dr. Akash Mecwan

Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr. Akash Mecwan**, Assistant Professor, EC Engineering Department,School of Technology, Nirma University and Internal Guide for guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Dr. N. P. Gajjar**, for his exemplary guidance, monitoring and constant encouragement.

I would also like to thank **Mr. Revanuru Murthy**, **Mr. Sagar PuttaSwamy** from **Intel Technology India Pvt. Ltd.**, for guidance, monitoring and encouragement regarding the project.

- Bhumi Dave

17MECE04

Abstract

VLSI technology is advancing day by day, transistors have been scaled down a lot incorporating more complex design in single System on Chip (SoC). Now a days the complexity of chip is rapidly increasing, verification plays a dominant role concerned with time and price in the enhancement of a Soc. Increased design complexity mandates the need for functional verification. The bug that is found at early level of abstraction will reduce the total cost incurred on a single chip so 70 percent of the time is devoted in verifying the design.

Even though SOC requires multi-instances of IP integration and verification collateral, the IP development methodology is built for a single instance and this introduces unique challenges at SOC. Numerous challenges arise when the IP environment is stamped multiple times with regards to RTL, Test bench, test Island collateral integration and IP validation, resulting in long integration and debug cycles. This report addresses the SoC problem of integration of IPs by providing a mini SoC framework, which delivers scalable integration (collage and testisland) collateral capability, flexible reconfiguration of the verification environment. This report also presents key learnings and challenges encountered during development of the Multi-Instance (MI) SoC-Subsystem. This proof-of-concept subsystem was successfully deployed for PCIE Multiple Virtual Channel (MVC) IP and reused for PCIE Single Virtual channel (SVC).

Abbreviation Notation and Nomenclature

IP ..... Intellectual Property
SoC ..... System on Chip
PCIE ..... Peripheral Component Interconnect Express
I2C ..... Inter-IC
USB ..... Universal Serial Bus
HDL ..... Hardware Description Language
RTL ..... Register Transfer Level
FSM ..... Finite State Machine
OVM ..... Open Verification Methodology
DUT ..... Device Under Test
FPV ..... Formal Property Verification
GLN ..... Gate level Net list
EDA ..... Electronic Design Automation
IEEE ..... Institute of Electrical and Electronics Engineers
VLSI ..... Very Large Scale Integration
IC ..... Integrated Circuit
BFM ..... Bus Functional Model

Contents

List of Figures

Chapter 1

Introduction

This chapter discusses about verification goals and also about motivation, objective and problem statement.

1.1 Organization Profile

Intel is one of world's leading semiconductor company and the developed x86 series of processors, processors which were found in the computers. Intel was founded in 1968 to develop semiconductor based memory products. Intel has developed the worlds first processor in the year 1971. Today, Intel provides industries with processors and software blocks that are the main parts of computers and networking and servers and communications products.

Intel began its operations in India in the year 1988. Initially it was sales and marketing office was started in India, Intel India was expanded in faster rate due to the India's I.T and engineering talent pool. At present work done at Intel India is hardware and software engineering.

1.2 Verification

The hardware design verification goal is to check design implementation matching with high level specification. Hardware design should perform a particular task for which it has implemented some IPs such as PCIE, I2C and USB etc. based on the specifications of the design. The task of validation engineer is to make sure that design should function correctly and complete that desired task i.e. the design is an exact replica of the design. The verification process runs in parallels with the RTL design process. Designers reads the hardware specification and understand the human language description and creates the register transfer level (RTL) description in machine readable format generally called as RTL code. To write RTL description one needs to get control on overall architecture of the design and input format, transform function and the desired output. Verification process checks the design for agreement with the specifications and reports any discrepancies / bugs. Further prepares a verification plan along with the test list is created. The validation plan is close to the design and has explanation of what features need to be added and what techniques can be used. The digital model is exercised in an OVM based verification environment using automatically generated constrained random test sequences. The progress of verification is tracked using the code and functional coverage data. The coverage reports are analyzed and to address the gaps to reach 100 percent verification closure, the test set is supplemented by adding direct tests which exercises the functionality to be tested.

One of the significant components in the verification environment is the Bus Functional Model (BFM). BFM is nothing but an agent in the test environment consisting of driver, monitor and sequencer. BFM model initiates bus transactions as if the transactions were available from other external systems and evaluates the output produced by the DUT. It converts from high level transactions into pin activities at the DUT interface. Verification engineer also need to study the architecture, build the validation plan, then implement design to create tests so as to exercise RTL

code. There are different ways to test the design. While performing verification it is easier to find error at block level of modules designed by verification engineer. Bugs at block level can be easily found as they are within the single block by writing the directed tests. After the completion of block level verification, the further plan is to search for the errors at corners of the different blocks. Mismatch in design will be encountered when more than two designers follow the same design specification but have different ideology. The Engineers job is to finds the more error prone areas of design and this may even help to reconcile the two views of the designer. When the DUT is complex, then the directed test will become a tedious job. So it demands a specific preplanned verification plan for the testing all the scenarios in the DUT.[1]

1.3 Motivation

Scalability is newly growing term in industries to automate the entire design verification environment according to design configurations. Since we are getting project configurations and according to that we are creating our verification environment and its different components. It is difficult for verification engineer to re-create an environment based on new configurations with required time constrained. Many times we copy previous environment and try to build environment and its components based on specs. This method leads to many manual error and it becomes cumbersome when debugging them.

Earlier projects took 4 to 8 weeks at SoC level to integrate subsystems. With scalable platform performing complete Integration, time reduces to 1 to 2 weeks to integrate subsystems.

Less than 2 weeks to integrate and run L0 content

Over 75% reuse reuse of validation content

Ensure changes to IP (in the boot path), reconfiguration of the chassis and integration of the Chassis and IPs do not break the Boot flow

Reduce TTM for new SoCs

The majority of bugs are solved which helps SoC team and reduces time to market for SoC .

Maximum coverage of validation

Parallel and faster runs against subsystems enables quick debug.

1.4 Problem statement

Previous SoC projects took 6 to 8 weeks for SoC team, to integrate an IP delievered by IP team and complete the validation. Moreover, verification collateral provided by IP team was also not of right quality. This delayed the SoC projects, increasing the time to market.For multi instance environment, the problem was more significant to bring the validation collateral up in given deadlines.

A novel approach was invented consisting of mini SoC environment having essential components to bring up the right quality of verification collateral. Scalable environment for multiple instances is developed. Complete integration of IP's provided by IP Team is performed at mini SoC level, providing this integrated IPs to SoC team. Solving the majority of bugs at the mini SoC level, helps the SoC team to complete the project in 1 week and this reduces the time to market for SoCs.

1.5 Objective

Major objectives of thesis are:

- Understand functionality of IP and its verification environment if already exist or create verification environment according to test plan and functional requirement of IP.
- Learn System Verilog and OVM which are the fundamentals required for creating a complex reusable, constrained random stimulus and coverage driven verification environment.

- Analyze different OVM component and find the scope of scalability.

1.6 Flow Of Report

This thesis is divided into six chapters as follows:

- **Chapter 1** This chapter presents basic theory on Verification, Motivation, Problem Statement and Objective for the Thesis.
- **Chapter 2** This chapter describes Literature Survey on System Verliog,VLSI process flow and Verification.
- **Chapter 3** This chapter describes an overview on Pre-Verification Process, Functionality of Test Bench and Types of Testing.
- **Chapter 4** This chapter presents discussion on Scalable Integration Platform,Implementation details and challenges faced during Multi Instance SS Verification Enablement
- **Chapter 5** This chapter presents results of the project.
- **Chapter 6** This chapter closes the thesis with concluding remarks

1.7 Summary

As discussed in this chapter, the major objective is to reduce the time to market for SoCs by mini SoC environment having essential components to bring up the right quality of verification collateral.

Chapter 2

Literature Survey

In the previous chapter, the motivation, objective and problem statement were discussed. This chapter explains VLSI process flow, types of verification and features of System Verilog.

2.1 System Verilog

System Verilog combines the features of Hardware Description Languages like Verilog and VHDL and with Hardware Verification languages like System C along with features from C and C++.System Verilog was made an IEEE standard in 2005 (IEEE1800) and was updated in 2009 and later in 2012.It's wide variety of options and flexibility makes it as choice of industry standard. System Verilog applications includes RTL design, Assertion Based verification, and coverage driven verification environment using constraint random techniques.In a verification perspective, System Verilog uses object oriented techniques consisting of classes and is closely related to Java. Most of the EDA software vendors support System Verilog in their products.

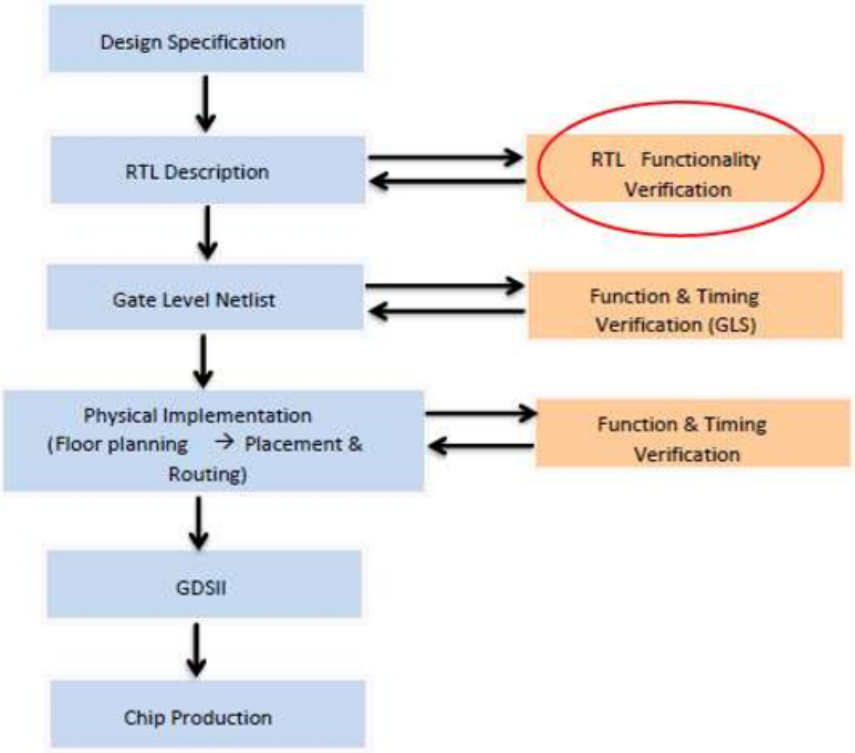

Figure 2.1: VLSI Flow

2.2 VLSI Flow Process

Below are the steps in VLSI flow:

2.2.1 Design Specifications

- Goals and design constraints.
- Functionality.
- Speed and power performance metrics.
- Technology constraints (Fabrication technology and design techniques).

2.2.2 Structural and Functional Description

Kind of architecture (structure) to be used for design, example ALU, pipelining, RISC/CISC, etc. A complex system design problem can be normally divided into smaller sub design. The functionality of this sub block should work according to the specifications.

2.2.3 Logic Design/Register Transfer Level (RTL)

Once defined, the top level system and subsystems need to be implemented. They are implemented using schematics, combinatorial and sequential logic, logic expression, finite state machines, etc. RTL should match the functional description of the subsystems. RTL is usually described with Hardware Description Languages as Verilog or VHDL. At this stage, Functional/Logical Verification is performed to ensure the RTL designed achieves the required functionality.

2.2.4 Gate Level Net list

On completing the functional Verification an optimized Gate Level Net list is generated from RTL through Logic/RTL synthesis. Synthesis Tools like as Design Compiler (Synopsys), RTL Compiler (Cadence), Magma (Blast Create), etc are used at this step. A design RTL description and standard cell library are inputs to the synthesis tool and at the output it produces a gate-level net list. Constraints such as area, timing, testability, power, etc. are considered. Synthesis tools calculate the costs of different possible implementations to meet these constraints

2.2.5 Physical Implementation

Physical implementation of the Gate Level Net list is the next step in the ASIC flow. The Gate level Net list (GLN) is converted into geometric structure which represents the layout of design. The layout is done according to the design rules

specified in technology library. The design rules represent limitations of the fabrication process. Sub-steps in Physical Implementation:

Floor planning, Placement Routing. Physical Implementation produces a GDSII file to be used by the foundry for fabrication of chip. Several tools such as Blast Fusion (Magma), IC Compiler (Synopsys), and Encounter (Cadence) etc. are used at this step. Physical Verification verifies to ensure that the layout meets the fabrication rules.[2]

2.3 Verification

The hardware design verification goal is to check implementation matching with high level specification. Hardware design should perform a particular task for which it has implemented such as DVD (digital video disc) player, router network and signal processor in radar, based on a design specification. The task of validation engineer is to make sure that device should function correctly and accomplish that desired task successfully that is, the design is a correct representation of the design specification. The verification process runs in parallels with the RTL design process. Designers reads the hardware specification and understand the human language description and creates the register transfer level (RTL) description in machine readable format generally called as RTL code. To write RTL description he/she needs to understand the overall architecture of the design and the input format, transform function and the desired output. Verification engineer also need to read the architecture specification, build the verification plan, and then implement it to create tests so as to exercise RTL code. There are different ways to test the design. While performing verification it is easier to detect error at the block level, in the modules designed by a verification engineer. The Verification Engineers job is to find the more error prone areas of logic and maybe even helps to reconcile these two different views of the designer. When the DUT is complex, then the directed test will become a tedious job. So it demands a specific pre-planned verification

plan for the testing all the scenarios in the DUT. [3]

2.3.1 Types of Verification

- **Block Level/Standalone Verification**

In this type of Verification, a standalone System Verilog based test bench is created for each block. The directed tests are created to check the required functionality.

- **IP Level Verification**

Here in this type of verification, all the blocks are integrated and verification environment is created through standard methodology OVM and respective sequences are driven to check the each block response and also the IP Level output response.

2.3.2 Simulation based verification

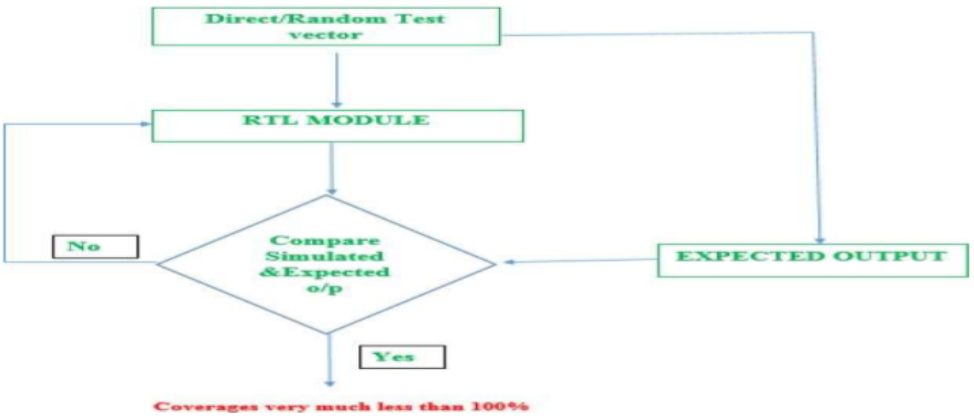

Figure 2.2: Traditional Simulation Based Verification

Coverage Based Verification

Coverage is metric used by verification methodology for reference for selection of test.

- **Code coverage:**

  Code coverage is calculated by tool itself. This feature is there in all the HDL it gives the number for how many lines of RTL code are executed, how many times corresponding expressions is covered, branches executed.

- **Functional coverage:**

Functional coverage gives idea about how much functionality is covered by the applied tests. Functional coverage observes the execution of a test plan. Functional coverage is feature of system Verilog which helps to determine how testing of the design has done. If the functional coverage is 100 % then it indicates that all features of design has been tested. Combination of 100 % functional coverage and code coverage indicates that testing is done.

2.3.3 Assertion based verification

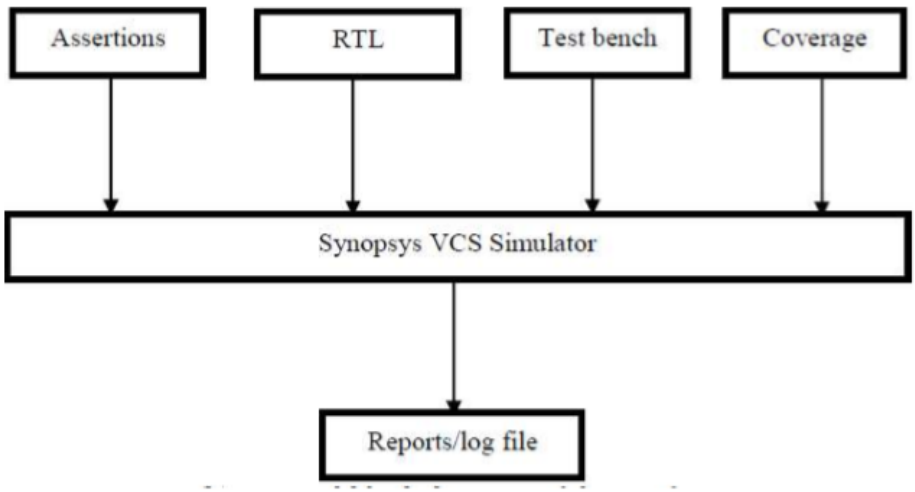

Figure 2.3: Typical block diagram of the verification environment

Assertions are the conditional statement that is used to check the specific behavior of the design and it will display a message on occurrence of it. Assertion helps us to improve observability and catches error earlier in the design phase.

Mostly, we will add assertions during verification process to observe the conditions that are hard to check by using simulation process. And most of time, they are used to simplify the debugging process of the complex system design. Assertion monitors wait for a particular condition to occur and then it alert the designer when it occurs. Assertion monitors can be treated as internal test point. Assertion is used to improve the observe ability of the complex design. If assertions are not used then applied test vector has to be long enough to ensure the bug triggering and will get propagated to the observable outputs, otherwise we will not able to detect the bug. Assertion allows us to check bad behavior within the design and help us to remove the bugs in the design.

2.3.4 Formal Property verification (FPV)

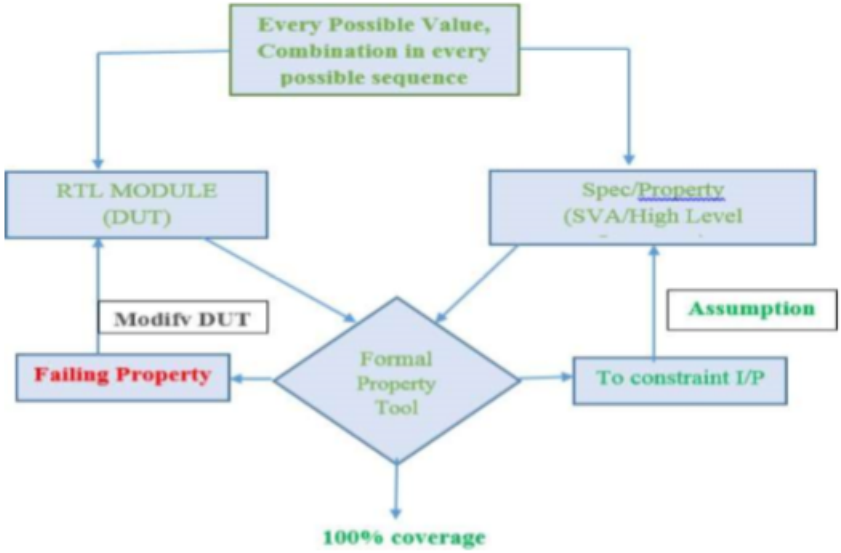

Figure 2.4: Formal Verification

FPV (Formal Property Verification) is a static verification technique which is capable to reduce the verification time and hunts the bug which dynamic verification is not capable to find. Formal Property Verification technique analyzes the VLSI design mathematically and proves against the set off properties which are meant to

be true for all test cases. Formal Property Verification ensures us 100 % coverage. Many design teams have reported finding important bugs using FPV techniques. The major input to FPV is a design description with a set of properties, in our case Verilog RTL models with embedded OVL properties. (It is also possible to run FPV on more abstract model with externally specified properties.) Some of the properties must be designated by the user as assertions, targets that need to be proven. The rest will be assumptions, or constraints, properties that the tool should take as a given. Usually the assumptions are properties on inputs, or properties whose proof would depend on logic external to the model. The level of hierarchy where FPV is run has to be chosen carefully the tools can quickly run out of memory on complex models, and we had to run numerous designs at unit level instead of cluster level. Often it will take a few attempts (in which the tool crashes due to excessive memory consumption) in order to identify a reasonable level to run. Once the proper hierarchy is selected, the user can effectively run the FPV tool and analyze the result. The formal verification activity was carried out on the DUT at the block level first and then IP level. [4]

2.4 Summary

This chapter describes Literature Survey on System Verliog,VLSI process flow and Verification.

Chapter 3

Pre -Verification Process

In the previous chapter, VLSI process flow, types of verification and features of System Verilog were discussed. This chapter explains Pre-Verification Process, Functionality of Test Bench and Types of Testing.

3.1 The Verification Process

Detecting bugs is not the entire goal of verification. The main intention of design in hardware is to develop a device that can perform the specific task, for example a mobile phone, a TV, or a laptop, based on the specification. The main job of a verification engineer is to ensure that the device/hardware can successfully accomplish the dedicated task, that is, the design is a proper representation of the given specification. The bugs are just a result of discrepancies in the design. The verification process starts in parallel with the design development process. Both the designer and verification engineer read the given hardware specification. The designer creates a corresponding logic for the hardware specification in a machine readable format, which is usually referred to as RTL code, whereas the verification engineer creates a verification or test plan, which is followed to build tests to check the RTL code behavior with respect to the hardware specification. There are several ways that a design can be tested. The easiest among all is to verify the designs at

the block level, that is, modules designed by one person. It is simple and easy to plan and develop directed tests to detect the bugs in such designs, as the design is within a single block and the designer has complete understanding of the design as only one person has designed it. Next to block level, another aspect to check is at the boundaries between the blocks. Critical problems come up when multiple designers go through the same description and interpret it differently. First design may build transmitter with his view on the specification while the second designer may build a receiver with his own view on the specification, with a slight different view. The verification engineers role is to detect the disputed sites of the logic and may even help reconcile the different interpretations. When the DUT is a complex design then the directed tests will become a difficult job. So it demands for a properly pre-planned verification or test plan for completely testing all the scenarios in the DUT.[5]

3.1.1 Test bench Basic Functionality

The main purpose of a test bench is to ensure the correctness of the given Design Under Test (DUT). This can be accomplished by:

- Generating the stimulus
- Applying the stimulus to the DUT
- Capturing the response
- Checking for the correctness

3.2 Directed Testing and Constrained-Random Testing

3.2.1 Directed Testing

Traditionally, the process of verification a design uses directed testing. With this approach, a verification engineer goes through the given hardware specification and develops a verification or test plan containing a list of tests, with each test focused on a set of features in the design. Under this plan, verification engineer applies a stimuli that can exercise a particular set of features in the DUT. Then he/she will simulate the DUT with the stimuli and reviews the results in log files generated and in waveforms to ensure that the design is doing whatever is expected. If the design works correctly for the given stimuli, the verification engineer checks that particular test off in the test plan and moves on to the next one and applies other stimuli. Figure 3.1 shows how the direct tests cover the features in the DUT with the test plan. Each test in the test plan is targeted for a specific set of features in the design. If the verification engineer has enough time, he/she writes all the tests to get 100 % coverage of the entire test plan.

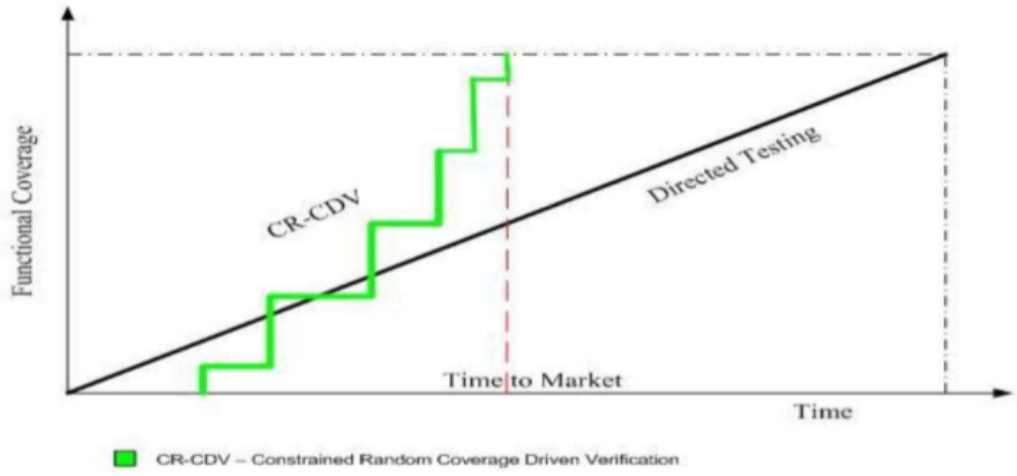

Figure 3.1: Functional Coverage:CR-CDV v/s Directed Testing

If the verification engineer doesnt have enough time nor resources to go with directed testing method or if the design complexity increases, then it takes more time to finish testing or it needs more resources /people to finish the testing. None of the above situations is feasible. So it demands for a methodology, which can detect bugs faster and can reach the 100 percent coverage goal. Here comes the role of the role of constrained random way of testing.

3.2.2 Constrained-Random Testing

Every design may not be able to be tested with complete random values. It may be needed to randomize among a setoff values (for example, address is 16-bits;op-code is MUL,DIV,or SAVE; length<16 bytes), then we need constrained randomization to randomize only in a set of constraints. System Verilog provides the constrained randomization feature. The constrained randomized values are driven to the design and then to a top-level model that can estimate the output. The actual output of the design for the given randomized values is then compared with the estimated output to ensure the proper functionality of the design. Figure shows the paths to reach the maximum coverage. Starting with the simple constrained-random tests. Proceed to run the tests with multiple seeds with different seed values. Find the holes in the coverage from the coverage report. Now make a minimal tests for only the those few features that are so unlikely to be reached through random tests changes in the test code, using new constraints if required, or by error injection, or by introducing delays in the DUT. Spend more time in the outside loop by developing directed test case.[6]

3.3 Verification Metrics: Coverage

Coverage metrics are an indicator of the progress of verification. Coverage indicates how well the design has been verified and what lies unexplored in the code. Code and functional coverage are the two types of coverage metrics. Code coverage shows

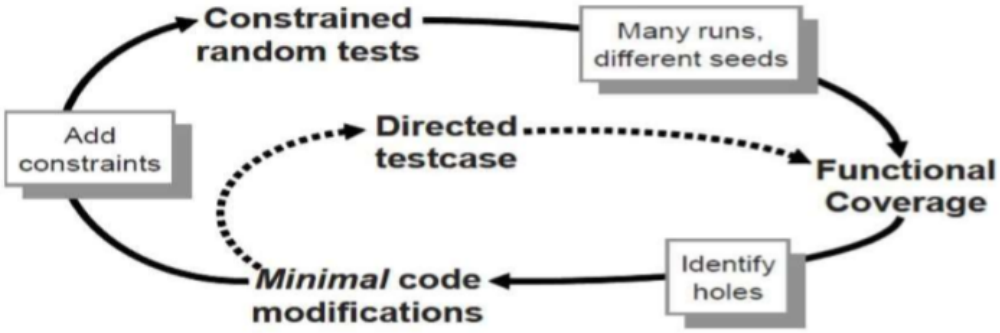

Figure 3.2: Constrained Random Testing vs. Directed Testing

how thoroughly the RTL code has been exercised by the verification environment, using the tests used in regression. This is a basic coverage type which is collected automatically by the tool running the simulations. The various classes of code coverage are;

- Block coverage
- Branch/Decision coverage
- Toggle coverage
- FSM coverage
- Statement coverage
- Conditional/Expression coverage

Statement coverage

Statement coverage indicates how many lines of the RTL code are covered in the simulation. All executable constructs like assignment statements, decision making statements are covered, while excluding non-functional constructs like module, timescale etc. For verification closure this should be 100 %.

Block coverage

A block is a group of statements with a defined scope. Examples of a block include statements within begin -end constructs of if-else, case, wait, while for loop. Block coverage gives the indication whether these blocks gets hit during a simulation run. Though it may look similar to line /statement coverage, Block coverage inspects blocks while line coverage, the statements.

Conditional/Expression Coverage

Conditional coverage is the ratio of number of cases hit to the total number of possible cases upon which a logical expression evaluates to true.

Branch/Decision Coverage

In Branch coverage checks if the constructs like ternary operator (?:) if else etc are evaluated for both true and false cases.

Toggle Coverage

Toggle coverage measures toggle activity on signals and ports during a simulation run. It uncovers unused signals or signals that undergo less or not much activity.

State/FSM Coverage

This checks if all the states of the state machine model led by the given RTL code are reached in the simulation run, and if all possible arcs for state change are traced. This coverage type tracks the behavior of the HDL design and hence is complex.

Functional Coverage

Functional Coverage is the metric of how much design functionality has been exercised/covered by the test bench or verification environment which is explicitly defined by the verification engineer in the form of a functional coverage model.In its simplistic form,it is user defined mapping of each functional feature to be tested to a so called cover point these coverage point also have certain conditions (ranges, defined transitions or cross etc.) to fulfill before it is s announced as 100 % covered during simulation. All these conditions for a cover point are defined in form of bins. During simulation, as and when a certain condition of a cover point hits, those bins (conditions) are getting covered and it gives us the measurement of verification

progress. After executing a number of test cases,a graphical report may be generated to analyze the functional coverage report and plan can be made to cover up the holes. Number of cover points can be captured under one cover group collection of number of cover groups is usually called a functional coverage model.

Both code and functional coverage are equally important in verification process. Even with 100 % code coverage, verification is still incomplete, if functional coverage is not 100 %, alternatively, if functional coverage is 100 %, without 100 % code coverage, it indicates dummy code. Verification completeness can be achieved with 100 % code functional coverage along with targeting proper functional coverage goals.[7]

3.4 Summary

The constant growth in the complexity of the designs demands for automated, systematic, and flexible methods to create the test benches. The cost of a bug fix grows exponentially as project transits from each step to the other step, like hardware specification to RTL development, synthesis, layout, fabrication, and delivery to the customer. Directed test approach can only check for a particular set of features at a given time and cant make complex stimuli that the device could be subjected to in the real world. To produce more robust and efficient designs, it becomes necessary to use the constrained randomized stimuli with the coverage collection to make the broader range of stimuli.

Chapter 4

Scalable Integration Platform

In the previous chapter, Pre-Verification Process, Functionality of Test Bench and Types of Testing were discussed. This chapter explains Scalable Integration Platform,Implementation details and challenges faced during Multi Instance SS Verification Enablement.

4.1 Scalable Integration Flow

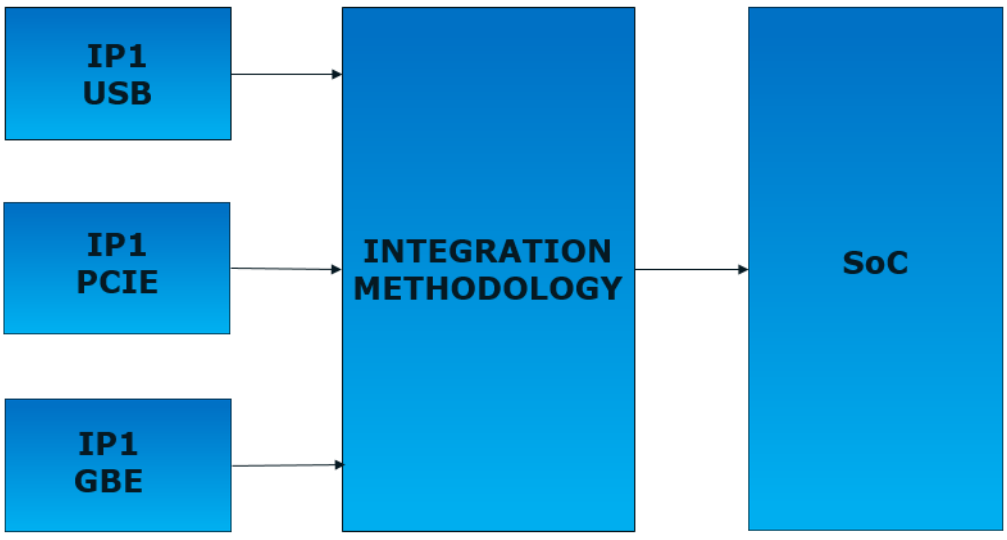

Figure 4.1: SoC Integration Flow

At Intel, most of the IPs that get consumed at SoC are never verified for multi-instance scenarios at the IP level verification environment, even though that is the SoC ask. This causes SoC teams to instantiate the multi-instance or multi-configuration scenarios at SoC directly. Due to this direct stamping of the IPs at SoC, the SoC teams experience various issues which are not encountered and envisioned by the IP teams. This results in multiple unique debug iterations at SoC to address these issues and each iteration consumes a lot of simulation, debug time thereby impacting the overall SoC schedule. This is the motivation to develop scalable integration and validation flow at Subsystem using mini SoC methodology. In todays SoC world multiple instance requirement per IP is a very common scenario and these integration/validation problems are seen whether SoC uses mini SoC approach or not. Focus is on providing a template base solution for integration and verification that leverages mini SoC frame work under the hood. Subsystems are required to verify IP integration and create integration verification collateral that can be reused by SoC.

The intent of a subsystem is to:

- Maximize IP test sequence reuse, improve overall porting/debug time at SOC, reduce validation cycle at SoC.
- Be able to have flexibility to run traffic concurrently and reduce complete reliance on Emulation/FPGA to find basic issues.
- Find and fix verification Environment and collateral issues before SoC.
- Build configurable and scalable Subsystems to easily and quickly deliver Subsystems to derivatives and multiple SoCs.

IP integration platform supports many different solutions to the same Chassis feature in a compatible manner.

A platform that includes-

- Infrastructure capabilities to integrate the Chassis components and build the integration methodology
- IP Integration Platform to integrate and test non-chassis IPs with the Chassis
- Suite of configurable integration tests that would test the chassis and any IPs integrated to the chassis.

4.2 Configurable Multi-Instance Subsystem

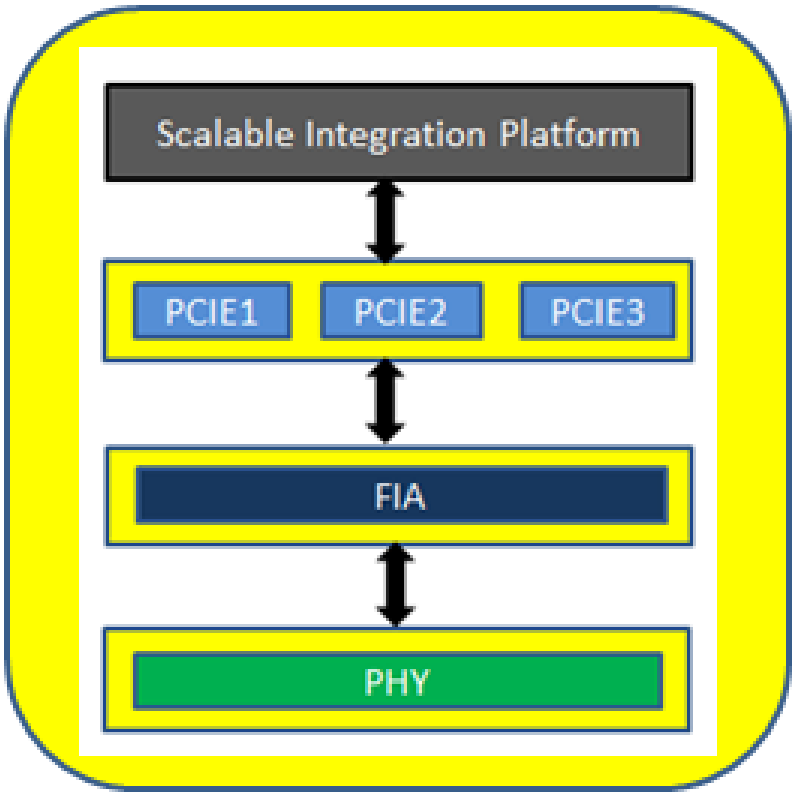

Figure 4.2: Configurable Multi-instance Subsystem

At subsystem (SS) level the IPs to be verified are integrated using mini SoC methodology and an integration wrapper is created across the SS to validate any IP. To build a Multi-instance subsystem a Configuration.tcl template is created which is

used as input to collage flow along with parameterized collage connectivity files. By tweaking the variables/parameters defined in the configuration template multiple instances of an IP RTL and corresponding SoC components can be instantiated in a subsystem. To integrate IP verification collateral (test Island, interfaces, knobs, test sequences) similar technique is adopted thereby making sure that the entire verification environment is configurable and scalable using global parameters, present in a single location. Due to this approach entire design and validation environment collateral can be reused at SoC and being configurable in nature can be easily re-spun and delivered to a derivative SoC as well.

4.3 Configurable Integration Flow for Complex IP

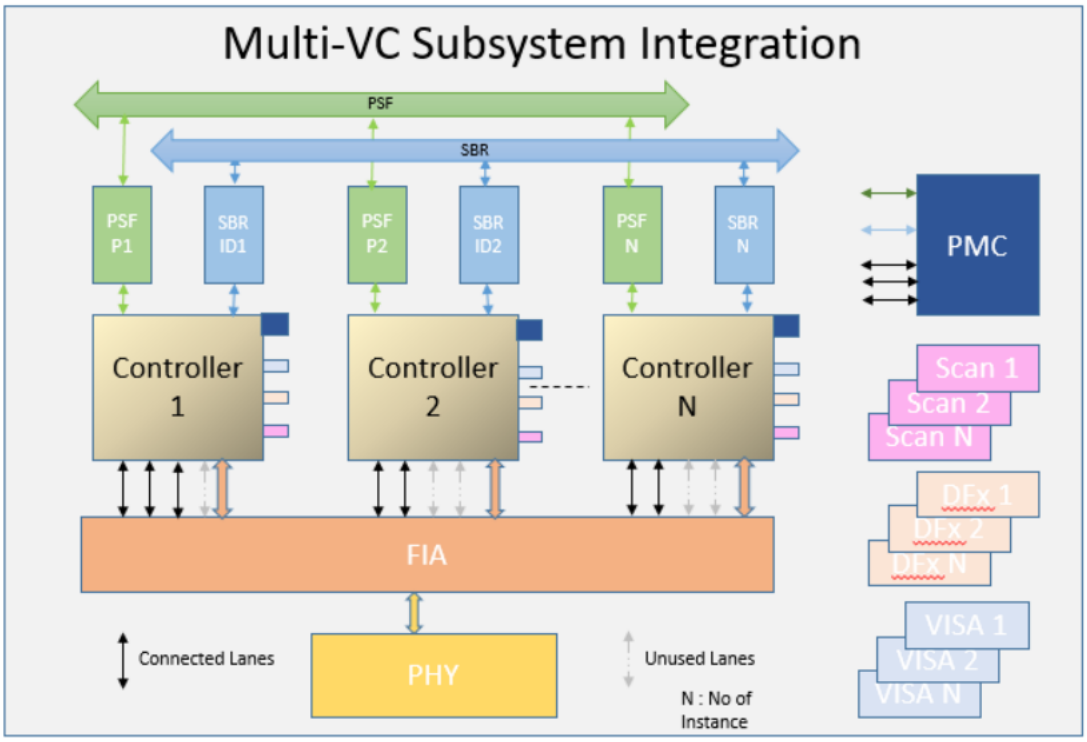

Figure 4.3: Representation of a scalable Multi-Instance Subsystem

Today most SoC create multi-instance integration environment by manually replicating standard interface connections, collage connectivity collateral, IP adhoc connectivity per instance. This results in hand editing multiple collage files, introduction of unintentional manual errors and debugging of integration issues that are not related to actual RTL collateral. To minimize the errors introduced because of manual intervention, the subsystem team decided to create a Configuration.tcl template which will be used as the base for scalability (to avoid manual stamping) and create collage connectivity templates. Figure 4.3 below shows example of multi-instance subsystem with mini SoC components.

4.4 Implementation Details

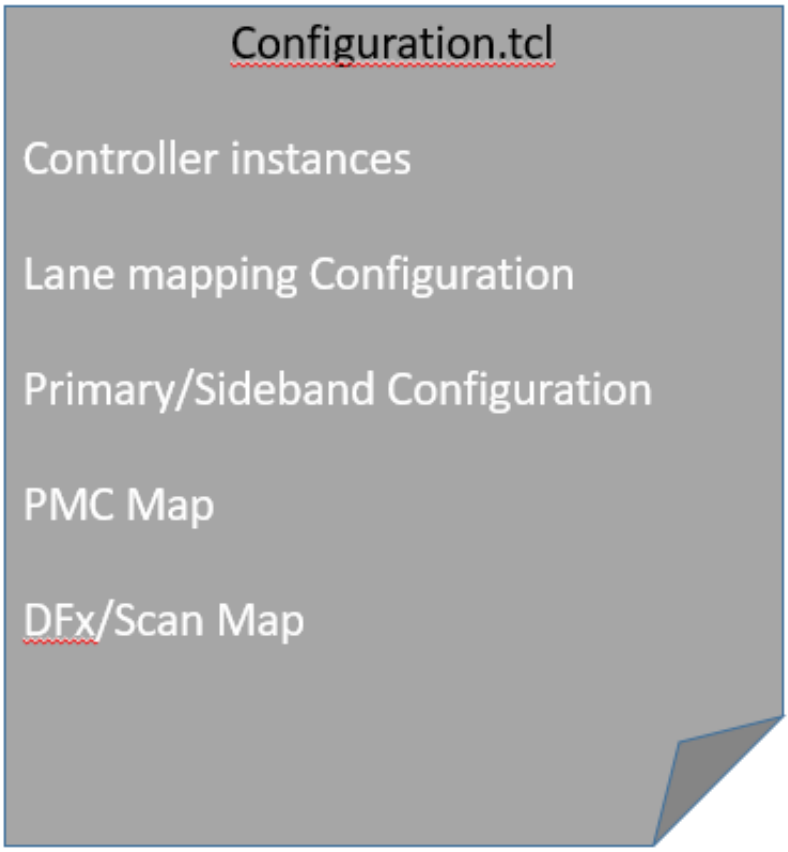

Figure 4.4: Multi-Instance Configuration Script

The Configuration tcl template can be thought as a parameterized input script to collage which consists of all the information that is required for configurability and scalability for a certain subsystem. This information bundle can change depending on the IP, figure 4.4 below shows an example of PCIE configuration script.

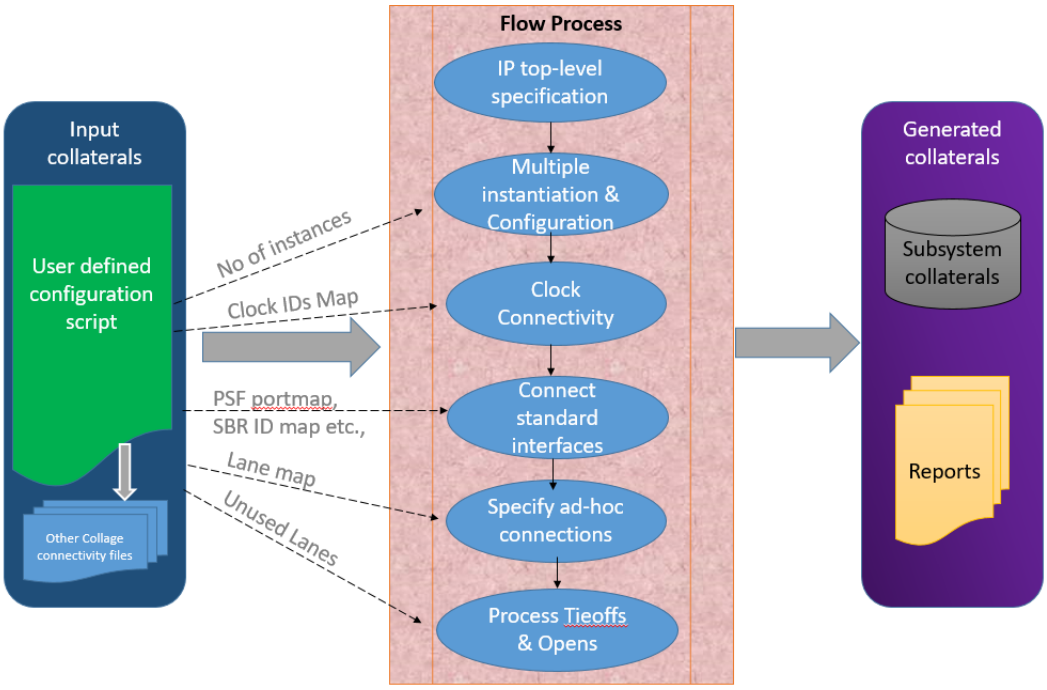

Figure 4.5: Process Flow of the User Defined Configuration Script

The Configuration template contains structure declarations with element value pairs. Elements declaration and the value pairs in the structure is called out once in other collage connectivity files for implementing the scalable and configurable connectivity. The flow using this user defined configuration script is depicted in Figure 4.5.

The code snippet from the template in Figure 4.6 below shows the structure map of controller instances for PCIE. This structure for controllers instance is used in the collage connectivity file as shown in figure 4.7. Just by manipulating the number of elements in this controller structure we can achieve scalability of creating RTL connections between N number of controllers and the PHY.

Another example of configurability is achieving multi-instance PCIE Controller

```
set pciemultivc::parcore_ctrls {
`PCIE_MULTIVC_PARCORE_0_CTRL
`PCIE_MULTIVC_PARCORE_1_CTRL
`PCIE_MULTIVC_PARCORE_2_CTRL
}
set pciemultivc::parlink_ctrls {
`PCIE_MULTIVC_PARLINK_0_CTRL
`PCIE_MULTIVC_PARLINK_1_CTRL
`PCIE_MULTIVC_PARLINK_2_CTRL
}
```

Figure 4.6: Configuration template snippet

and FIA lanes mapping based on the structure defined in the template file as shown in Figure 4.8. Based on SoC requirements the controller to FIA lane mappings can change, number of controllers can change from one project to another but at the subsystem level only the template file needs to be modified with the desired mapping and the entire connectivity can be re-generated without touching the collage connectivity as shown in Figure 4.9.

Similar to the examples shown above PSF, SBR, BDF maps are created in the configuration template to achieve scalability.

4.5 Configurable Verification Flow

To create scalability similar to integration flow, parameterized templates for test-island wrappers, test island interfaces, test bench environment and verification test sequences are created. Modifying these parametrized values in the verification files results in scaling of the IP verification environment from 1-N instances as per the project requirements. Some System Verilog limitations currently prevent from completely parameterizing of the test-island template. Verification test templates were

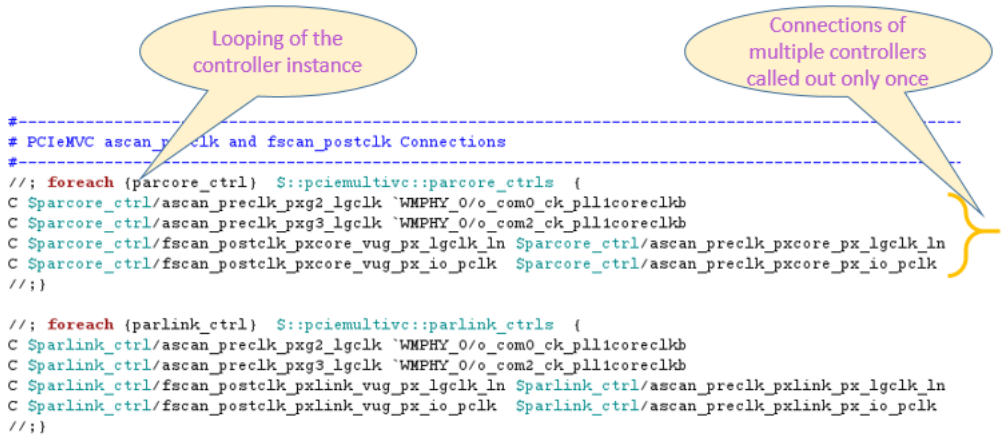

Figure 4.7: Parameterized collage connectivity file

```
set pciemultivc::parcore_ctrl_fia_map {
 `PCIE_MULTIVC_PARCORE_0_CTRL  0 pcie 4
 `PCIE_MULTIVC_PARCORE_0_CTRL  1 pcie 5
 `PCIE_MULTIVC_PARCORE_1_CTRL  0 pxpb 0
 `PCIE_MULTIVC_PARCORE_1_CTRL  1 pxpb 1
 `PCIE_MULTIVC_PARCORE_2_CTRL  0 pxpb 2
 `PCIE_MULTIVC_PARCORE_2_CTRL  1 pxpb 3
 }
```

Figure 4.8: Controller and FIA lane template map

created with hooks and parameters that provide configurability control of the IP test sequences to run for 1 to N IP instances based on the values set at subsystem as well as SoC. In order to achieve high test content reuse from the IP-SS-SoC, Subsystem team worked closely with the IP team to create modular and scalable sequences at IP level itself.

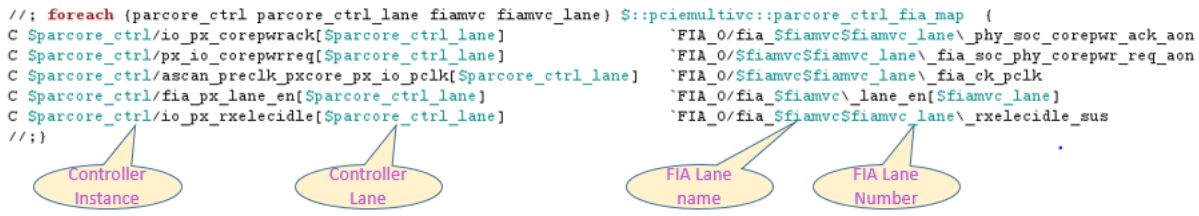

Figure 4.9: Collage connectivity parameterization for Controller and FIA

4.6 Challenges during Multi-instance SS Verification Enablement

- IP Environment Incompatibility

The way IP environment is structured and designed, it does not lend itself to be reused across different IP instances at subsystems or SoC. For multi-instance scenario the testbench and interface files were not scalable to subsystem. This resulted in making tweaks and hacks the IP environment directly while validating the flow at Subsystem and unique debug scenarios that couldnt be replicated at IP level. Some of the issues can be found early-on in the IP environment itself if they support multi-instance or at minimum create test bench collateral with scalability in mind and run some basic sequences at IP level to flush out basic issues such as static variables, hardcoded environment variables/parameters.

- IP non-portable sequence

Due to the nature of IP level validation, the test sequences were not directly reusable from IP environment. While porting the IP sequences to SS, issues related to hardcoded parameters, missing constraint hooks, static environment variable etc. were found, which required creating test sequences at subsystem or tweaking IP level sequences. Because of this issue, some of the earlier SoCs created test sequences directly at the SoC. This resulted in maintaining and debugging a separate set of the test sequences at SoC with no debug leverage from the IP verification environment. To avoid facing these kind of issues in future, IP teams should provide a minimal set

of test sequences to SS/SoC as part of the integration test suite that can be re-used and are more directed to cover main feature validation. SS/SoC teams can build on these sequences if more coverage is required.

- Concurrent Traffic validation

Due to limitation of the IP verification environment that prevents running of test collateral on more than one instance at a time on Subsystem, concurrent traffic validation scenarios couldnt be natively verified in the SS model. This causes high reliance on emulation at SoC model to get coverage. Recommendation was provided to the IP team to provide hooks in the IP tb env as well as flexibility in the IP sequences so that they can be easily scaled for multiple instances.

4.7 PCIE IP

Peripheral Component Interconnect Express (PCIe or PCI-E) is a serial expansion bus standard for connecting a computer to one or more peripheral devices.A computer expansion card standard. A standard type of connection for internal devices in a computer. PCI Express interface allows high bandwidth communication between the device and the motherboard, as well as other hardware.

PCI has some shortcomings. As processors,video cards, sound cards and networks have gotten faster and more powerful, PCI has stayed the same. It has a fixed width of 32 bits and can handle only 5 devices at a time. The newer, 64-bit PCI-X bus provides more bandwidth,PCI Express(PCIe) eliminates a lot of these shortcomings, provides more bandwidth and is compatible with existing operating systems

PCIe provides lower latency and higher data transfer rates than parallel buses. Every device that's connected to a motherboard with a PCIe link has its own dedicated point-to-point connection. This means that devices are not competing for bandwidth because they are not sharing the same bus. Peripheral devices that use

PCIe for data transfer include graphics adapter cards,network interface cards(NICs), storage accelerator devices and other high-performance peripherals.

Data is sent via paired point-to-point serial links, called lanes, allowing data movement in both directions simultaneously and allowing more than one pair of devices to communicate simultaneously. Serial buses transmit data faster than parallel buses due to the latters limitation requiring data to arrive simultaneously at their destination (This has to do with the frequency and wavelength of a single bit). With serial buses there is no requirement for signals to arrive simultaneously.

4.8 Demux wrapper for IP

Automated Perl Script to generate Demux wrapper for IP was developed to reduce manual work.

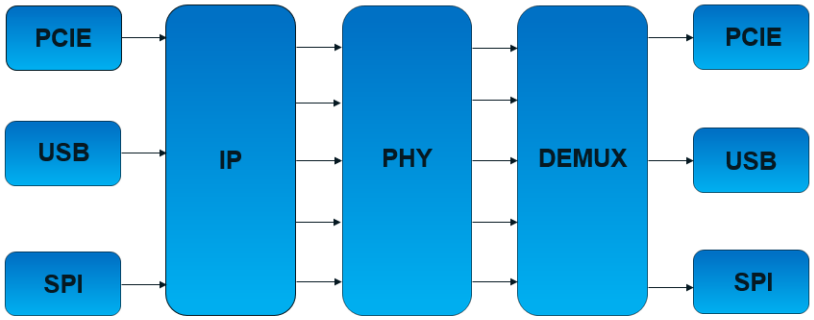

Figure 4.10: Block Diagram for Demux Wrapper

4.9 Summary

This chapter explains in brief about Scalable Integration Platform,Implementation details and challenges faced during Multi Instance SS Verification Enablement due to IP Environment Incompatibility, IP non-portable sequence and Concurrent Traffic validation.

Chapter 5

Results

The main aim of Scalable Integration Platform is to provide the right quality of Validation Collateral to SoC team for integration of subsystems that indirectly reduces TTM for SoCs.

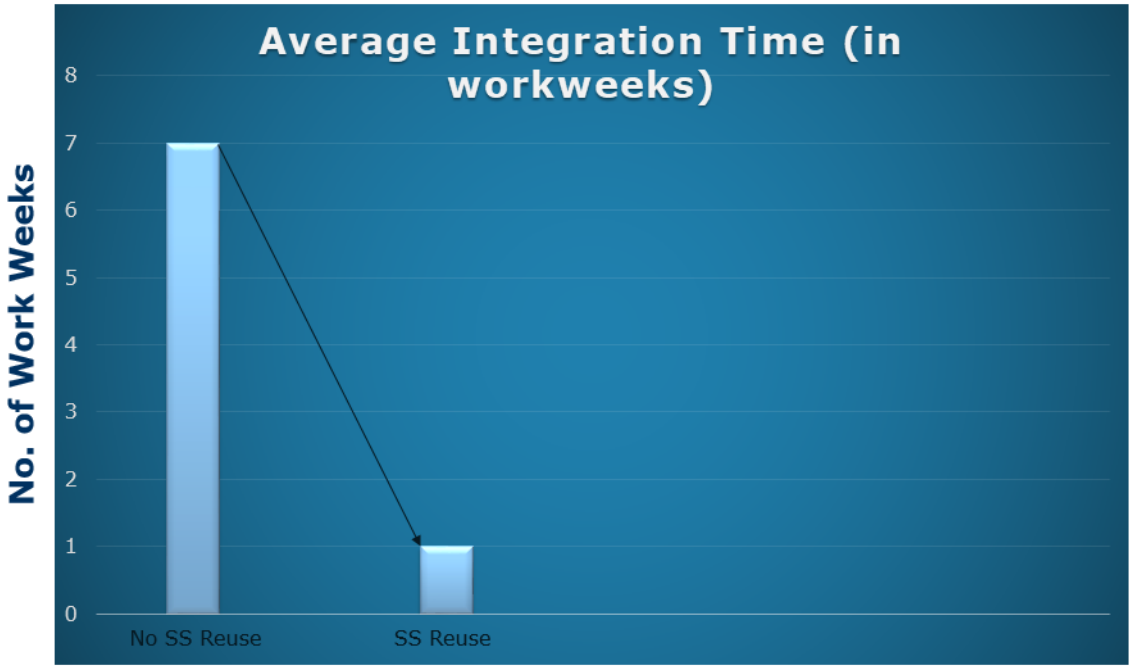

Figure 5.1: Integration timelines at SoC with SS usage

PCIE PXP configurable subsystem was successfully integrated and validated at SoC. The combined improvements from integration time at SoC, verification content

reuse, productivity (SoC debug cycles) and quality of collateral (bugs found) were significant.

Ease of integration at SOC: By reusing SS integration collateral at SoC the overall integration time and effort was significantly reduced per SoC milestone. Figure 5.1 shows with SS usage takes less then two work week to integrate.

5.1 PCIE Multi Virtual Channel

Regression results before and after Verification are shown in Figure 5.2 and Figure 5.3. After Verification 85 test cases were passing for PCIE Multi Virtual Channel Subsystem.

sc_model						
Subsystem	Owner	Planned Count	Run	Pass	Fail	Pass Rate
PCIEMULTIVC-MCC	sagarpu	85	85	63	22	74.12
soc_model						
Subsystem	Owner	Planned Count	Run	Pass	Fail	Pass Rate
PCIEMULTIVC-MCC	sagarpu	1	1	0	1	0
TOTAL FOR ALL MODELS						
Total	N/A	86	86	63	23	73.26
test coverage	coverage 1d	coverage 1w	coverage 4w	coverage 8w		

Figure 5.2: Central Regression Result (Before Verification)

sc_model						
Subsystem	Owner	Planned Count	Run	Pass	Fail	Pass Rate
PCIEMULTIVC-MCC	sagarpu	87	86	85	1	98.84
soc_model						
Subsystem	Owner	Planned Count	Run	Pass	Fail	Pass Rate
PCIEMULTIVC-MCC	sagarpu	1	1	0	1	0
TOTAL FOR ALL MODELS						
Total	N/A	88	87	85	2	97.7
test coverage	coverage 1d	coverage 1w	coverage 4w	coverage 8w		

Figure 5.3: Central Regression Result (After Verification)

Figure 5.4 shows that Toggle coverage before verification was 40.12%. Figure 5.5 shows that Toggle coverage after verification toggle coverage was 95.71% for PCIE Multi Virtual Channel Subsystem.

Module Instance : soc_tb.soc.par_multivcpcie_top

Instance :

SCORE	TOGGLE

Instance's subtree :

SCORE	TOGGLE
40.12	40.12

Parent :

SCORE	TOGGLE	NAME
		soc

Subtrees :

NAME	SCORE	TOGGLE
wrap_sippcie2_parcore_0	32.71	32.71
wrap_sippcie2_parcore_1	33.06	33.06
wrap_sippcie2_parcore_2	32.82	32.82
wrap_sippcie2_parlink_0	78.60	78.60
wrap_sippcie2_parlink_1	79.42	79.42
wrap_sippcie2_parlink_2	78.14	78.14

Figure 5.4: Toggle Coverage Result (Before Verification)

5.2 PCIE Single Virtual Channel

Regression results before and after Verification are shown in Figure 5.6 and Figure 5.7. After Verification 120 test cases were passing for PCIE Single Virtual Channel Subsystem.

Figure 5.8 shows that Toggle coverage before verification was 63.22%. Figure 5.9 shows that Toggle coverage after verification toggle coverage was 99.34% for PCIE Multi Virtual Channel Subsystem.

5.3 Passing and Failing Scenarios for Test Case

Figure 5.14 and Figure 5.15 shows the failing logs and waveforms for L1_LOW Test Case.

After debugging the issue,L1_LOW Test Case was passing.Figure 5.16 and Figure 5.17 shows the passing logs and waveforms for L1_LOW Test Case.

Module Instance : soc_tb.soc.par_multivcpcie_top

Instance :

SCORE	TOGGLE

Instance's subtree :

SCORE	TOGGLE
95.71	95.71

Parent :

SCORE	TOGGLE	NAME
		soc

Subtrees :

NAME	SCORE	TOGGLE
wrap_sippcie2_parcore_0	96.46	96.46
wrap_sippcie2_parcore_1	93.58	93.58
wrap_sippcie2_parcore_2	94.91	94.91
wrap_sippcie2_parlink_0	99.78	99.78
wrap_sippcie2_parlink_1	96.41	96.41
wrap_sippcie2_parlink_2	99.11	99.11

Figure 5.5: Toggle Coverage Result (After Verification)

sc_model						
Subsystem	Owner	Planned Count	Run	Pass	Fail	Pass Rate
PCIE-MCC	bknayaka	100	86	77	9	89.53
soc_model						
Subsystem	Owner	Planned Count	Run	Pass	Fail	Pass Rate
PCIE-MCC	bknayaka	100	4	4	0	100
TOTAL FOR ALL MODELS						
Total	N/A	100	90	81	9	90
test coverage	coverage 1d	coverage 1w	coverage 4w	coverage 8w		

Testlist Used: /nfs/sc/disks/sscoe_ip_model_builds2/pcie_mcc_integ/pcie_mcc_integ-sscoe-a0-18ww37c/subsystems/pcie_0_sbox/verif/regress/pcie0_level2_upf.list
Regression Area: /nfs/sc/disks/sc_ssc_00003/sub_sys_18ww38/PCIE-MCC_18ww38/5.2.1_pcie_multivc

Figure 5.6: Central Regression Result (Before Verification)

sc_model						
Subsystem	Owner	Planned Count	Run	Pass	Fail	Pass Rate
PCIE-MCC	bknayaka	100	120	120	0	100
soc_model						
Subsystem	Owner	Planned Count	Run	Pass	Fail	Pass Rate
PCIE-MCC	bknayaka	100	5	5	0	100
TOTAL FOR ALL MODELS						
Total	N/A	125	125	125	0	100
test coverage	coverage 1d	coverage 1w	coverage 4w	coverage 8w		

Testlist Used: /nfs/sc/disks/sc_ssc_00005/models/pcie_mcc_integ/pcie_mcc_integ-sscoe-a0-19ww13b/subsystems/pcie_0_sbox/verif/regress/pcie0_level2_upf.list
Regression Area: /nfs/sc/disks/sc_ssc_00003/sub_sys_19ww16/PCIE-MCC_19ww16/5.3.2

Figure 5.7: Central Regression Result (After Verification)

Module Instance : soc_tb.soc.par_pcie_0

Instance :

SCORE	TOGGLE
63.22	63.22

Instance's subtree :

SCORE	TOGGLE
54.56	54.56

Parent :

SCORE	TOGGLE	NAME
		soc

Subtrees :

NAME	SCORE	TOGGLE
pcie_0_pcie_top_wrap	51.78	51.78

Figure 5.8: Toggle Coverage Result (Before Verification)

Module Instance : soc_tb.soc.par_pcie_0

Instance :

SCORE	TOGGLE
99.34	99.34

Instance's subtree :

SCORE	TOGGLE
99.09	99.09

Parent :

SCORE	TOGGLE	NAME
		soc

Subtrees :

NAME	SCORE	TOGGLE
pcie_0_pcie_top_wrap	98.99	98.99

Figure 5.9: Toggle Coverage Result (After Verification)

sc_model						
Subsystem	Owner	Planned Count	Run	Pass	Fail	Pass Rate
PCIE-ADPLP	bknayaka	100	366	295	71	80.6
soc_model						
Subsystem	Owner	Planned Count	Run	Pass	Fail	Pass Rate
PCIE-ADPLP	bknayaka	100	23	7	16	30.43
TOTAL FOR ALL MODELS						
Total	N/A	430	389	302	87	77.63
test coverage	coverage 1d	coverage 1w	coverage 4w	coverage 8w		

Testlist Used: /nfs/sc/disks/sscoe_ip_model_builds1/pcie_adplp_integ/pcie_adplp_integ-sscoe-a0-19ww05a/subsystems/pcie_sbox/verif/regress/pcie0_level2_init.list
Regression Area: /nfs/sc/disks/sc_ssc_00003/sub_sys_19ww05/PCIE-ADPLP_19ww05/5.6

Figure 5.10: Central Regression Result (Before Verification)

sc_model						
Subsystem	Owner	Planned Count	Run	Pass	Fail	Pass Rate
PCIE-ADPLP	bknayaka	100	477	477	0	100
soc_model						
Subsystem	Owner	Planned Count	Run	Pass	Fail	Pass Rate
PCIE-ADPLP	bknayaka	100	26	25	1	96.15
TOTAL FOR ALL MODELS						
Total	N/A	503	503	502	1	99.8
test coverage	coverage 1d	coverage 1w	coverage 4w	coverage 8w		

Testlist Used: /nfs/sc/disks/sscoe_ip_model_builds1/pcie_adplp_integ/pcie_adplp_integ-sscoe-a0-19ww14a/subsystems/pcie_sbox/verif/regress/pcie0_level2_upf.list
Regression Area: /nfs/sc/disks/sc_ssc_00003/sub_sys_19ww16/PCIE-ADPLP_19ww16/5.6

Figure 5.11: Central Regression Result (After Verification)

Module Instance : soc_tb.soc.par_pcie_0

Instance :

SCORE	TOGGLE
64.76	64.76

Instance's subtree :

SCORE	TOGGLE
72.03	72.03

Parent :

SCORE	TOGGLE	NAME
		soc

Subtrees :

NAME	SCORE	TOGGLE
pcie_0_pcie_top_wrap	77.72	77.72
pcie_1_pcie_top_wrap	74.19	74.19
pcie_2_pcie_top_wrap	74.30	74.30

Figure 5.12: Toggle Coverage Result (Before Verification)

Module Instance : soc_tb.soc.par_pcie_0

Instance :

SCORE	TOGGLE
97.75	97.75

Instance's subtree :

SCORE	TOGGLE
97.84	97.84

Parent :

SCORE	TOGGLE	NAME
		soc

Subtrees :

NAME	SCORE	TOGGLE
pcie_0_pcie_top_wrap	97.55	97.55
pcie_1_pcie_top_wrap	97.34	97.34
pcie_2_pcie_top_wrap	98.76	98.76

Figure 5.13: Toggle Coverage Result (After Verification)

```
================================================================================
[Test Summary]
  Test ID              : pcie_0_REG_l1low_mseq_test_model_sc+plusarg_pcie_0_fuse_pcie0_fuse_RPCFG=01
  Test Name            : pcie_0_REG_l1low_mseq_test
  Test Type            :
  Status               : FAIL
  Cause                : Logfile Errors
  Logfile Error Count  : 1
  Logfile Warning Count : 10076
  Host Machine         : sccj004314
  CPUTIME              : 83212.90
  Runtime              : 83803
  Sim_Time             : 2500000000
  SimRate              : 0
================================================================================
[Error Summary]
# (All, Sorted by Severity, Other)
  1 - ALL : 1
    OVM_ERROR @ 2500000ns: reporter [TIMOUT] Watchdog timeout of '2500000ns' expired.
================================================================================
```

Figure 5.14: PCIE REG l1low mseq test:Failing Postsim Log

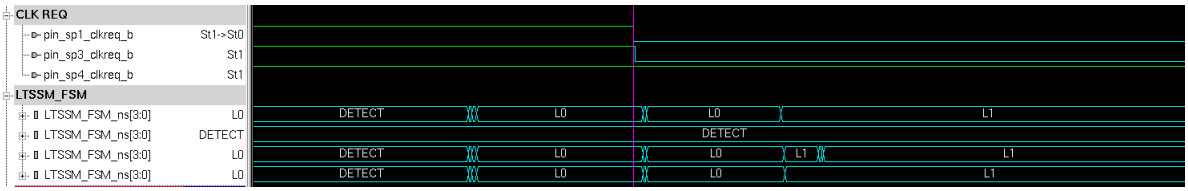

Figure 5.15: PCIE REG l1low mseq test:Failing Wave Results

```
================================================================================
[Test Summary]
  Test ID              : pcie_0_REG_l1low_mseq_test_model_sc+plusarg_pcie_0_fuse_pcie0_fuse_RPCFG=01
  Test Name            : pcie_0_REG_l1low_mseq_test
  Test Type            :
  Status               : PASS
  Logfile Error Count  : 0
  Logfile Warning Count : 10274
  Host Machine         : sccj000913
  CPUTIME              : 33920.50
  Runtime              : 34287
  Sim_Time             : 1215817000
  SimRate              : 0
================================================================================
```

Figure 5.16: PCIE REG l1low mseq test:Passing Postsim Log

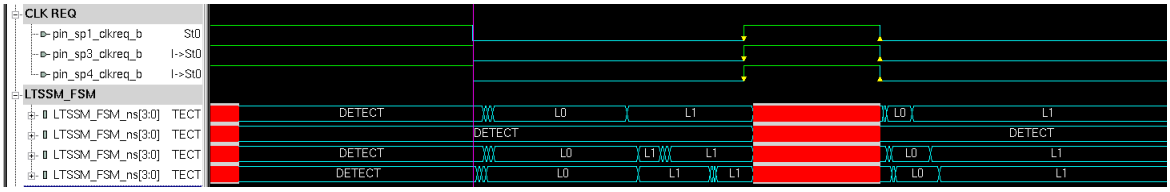

Figure 5.17: PCIE REG l1low mseq test:Passing Wave Results

Chapter 6

Conclusion

This chapter gives a brief about what has been learnt through out the project and results obtained in the previous project.

6.1 Conclusion

Various verification strategies will help the verification engineer to verify the complex design in a short period of time and will reduce the time to market for SoCs. Various verification and debugging approaches will help in reducing the effort put by the verification engineer in solving the debugging issues. Some of the debugging issues mentioned above can be overcome by having the mentioned debugging infrastructure and tools in the Scalable verification environment.

Also, a significant amount of time spent on verification can be saved by having these various metrics with proper architecture and flexible test benches.

With the scalable integration platform,it takes less than 1 week to integrate and run L0 content and over 75 % reuse of validation content. The majority of bugs are solved which helps SoC team and reduces time to market for SoC .It also provides maximum coverage of validation.Parallel and faster runs against subsystems enables quick debug

6.2 Future Scope

The platform can be modified in future for more scalability in order to reduce the number of bugs filed by SoC teams.

Power Management validaion features can be added to the platform which are now not present.

6.3 Summary

This chapter shows that the purpose of undertaking this project is fulfilled. It also gives an idea of how the Scalable integration platform is developed and improved in future.

Bibliography

[1] B. Patel, "Verification approach for asic generic ip functional verification," *Configurations*, vol. 1, no. 4, 2013.

[2] https://intelpedia.intel.com/, "Intel internal document,"

[3] N. Bamford, R. K. Bangalore, E. Chapman, H. Chavez, R. Dasari, Y. Lin, and E. Jimenez, "Challenges in system on chip verification," in *Microprocessor Test and Verification, 2006. MTV'06. Seventh International Workshop on*, pp. 52–60, IEEE, 2006.

[4] "Intel specific ip - high abstraction specification(has) documents,"

[5] C. Spear, "Verification guidelines," pp. 1–24, 2008.

[6] Y.-N. Yun, J.-B. Kim, N.-D. Kim, and B. Min, "Beyond uvm for practical soc verification," in *SoC Design Conference (ISOCC), 2011 International*, pp. 158–162, IEEE, 2011.

[7] C. Spear, *SystemVerilog for verification: a guide to learning the testbench language features*. Springer Science & Business Media, 2008.