Advance Aautomated Mechanism For Design, Compilation And Validation Flows Of Complex SoC Design

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master Of Technology

in

Electronics & Communication Engineering

(VLSI Design)

By
Ankur Sharma
(13MECV12)

Electronics and Communication Engineering Branch
Electrical Engineering Department
Institute Of Technology
Nirma University
Ahmedabad-382481
May 2015

Advance Aautomated Mechanism For Design, Compilation And Validation Flows Of Complex SoC Design

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master Of Technology

in

Electronics & Communication Engineering

(VLSI Design)

By

Ankur Sharma
(13MECV12)

Under the guidance of

Mr. Venkatesh K Elayavalli	**Prof. Vaishali Dhare**
Mr. Sandip Rajput	**Assistant Professor (EC)**
Intel India Technology Pvt. Ltd.	Institute of Technology,
Bangalore.	Nirma University, Ahmedabad.

Electronics and Communication Engineering Branch
Electrical Engineering Department
Institute Of Technology
Nirma University
Ahmedabad-382481
May 2015

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in VLSI Design at Nirma University and has not been submitted elsewhere for a degree.

b. Due acknowledgment has been made in the text to all other material used.

- Ankur Sharma
13MECV12

Certificate

This is to certify that the Project entitled **'Advance automated mechanism for design, compilation and validation flows of complex SoC design'** submitted by **Ankur Sharma (13MECV12)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design at Nirma University, Ahmedabad is the record of work carried out by him 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.

Prof. Vaishali Dhare
Internal Guide

Dr. N. M. Devashrayee
PG Coordinator (VLSI Design)

Dr. P.N.Tekwani
Head of EE Dept.

Dr. Ketan Kotecha
Director, IT-NU

Date:

Place: Ahmedabad

Certificate

This is to certify that the Project entitled **'Advance automated mechanism for design, compilation and validation flows of complex SoC design'** submitted by **Ankur Sharma (13MECV12)**, towards the submission of the Project (Phase-II) for requirements for the degree of Master of Technology in VLSI Design, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.

Date: Place: Bangalore

Mr. Venkatesh K Elayavalli
Engineering Td Manager
Intel Technologies India Pvt. Ltd.
Bangalore

Abstract

Time-to-market is the key factor in continuously evolving very large scale integration (VLSI) market and it is forcing the participants to improve methodologies and tool flows to gain a foothold in this highly competitive and fragmented business. The designing must be completed in a shorter period of time so that verification and fabrication can be completed without missing technology window. The design steps and software tools required on the front end part for designing modern systems-on-chip are no longer adequate for the complex systems of today. VLSI design process which involves writing RTL, compiling and verifying it are time consuming, verification itself consumes around 70% of the design time. Regardless of using a lot of CAD tools for the VLSI design, the process looks very complex and requires a lot of efforts for managing and verifying the design.

To handle this complexity different tools and flows are been developed to reduce the time required for design and verification, where the inputs required for the tool are generated by the flow. Verification gets easily done if the flow is tool friendly and generates all the required files by tool in proper format.

This report details about advance automated mechanism used for frontend design and verification through which time required and the complexity of frontend process can be reduced. Advance mechanism is a converged front-end flow that is flexible and easy to use. It supports all the modern design and verification methodology like formal verification, functional verification, design linting, static timing analysis, low power checks, emulation etc; it is generic and encompasses different VLSI flows. A higher degree of design confidence and reduction in the risk of re-spin or repeated efforts is the main aim of the advance system. It also provides a common environment to many projects while providing project-specific customization.

This report also details about modern and efficient methodology of formal property verification for design verification. It is observed that design data and verification and debugging time reduces significantly by using formal methodology for verification.

Acknowledgements

First and foremost, sincere thanks to Mr. Venkatesh K Elayavalli, Manager, Intel Technology India Private Limited, Bangalore for assigning me such project and guide me through. I would like to thank my Mentor, Mr. Sandip Rajput, Intel Technology India Private Limited, Bangalore for valuable guidance. Through-out the training, he had given me much valuable advice on project work which I am very lucky to benefit from. I would also like to thank my teammates, form Intel India Technology for their valuable time in ramping me up on some basic flow of different projects.

I would also thank to my thesis supervisor Mrs. Vaishali Dhare, Assistant Professor (EC) and Dr. N.M. Devashrayee, PG Coordinator VLSI Design, Institute of Technology, Nirma University, Ahmedabad for giving valuable support for project work and also teaching me some very intersecting subject in post-graduate programs. I also owe my colleagues in the Intel, special thanks for helping me on this path and for making project at Intel more enjoyable and more memorable.

- Ankur Sharma
13MECV12

Abbreviations

RTL	Register Transfer Logic
HDL	Hardware Discription Language
EDA	Electronic Design Automation
CAD	Computer Aided Design
A2M	Advance Automated Mechanism
FE	Frontend
TCL	Tool Command Language
PERL	Practical Extraction and Report Language

Contents

List of Figures

Chapter 1

Introduction

A modern VLSI chip is a remarkably complex beast: billions of transistors, millions of logic gates deployed for computation and control, big blocks of memory, embedded blocks of pre-designed functions designed by third parties (called Design property or Design blocks). How do people manage to design these complicated chips? Answer: a sequence of computer aided design (CAD) tools takes an abstract description of the chip, and refines it step-wise to a final design.[2]

1.1 Motivation

Extent of project covers working on various front end VLSI flows and providing a common flow solutions that reduce the design time, time to market and handle the design complexity. This project gives a good idea of front end flow methodology and designing techniques. It involves development and enhancement of advance features required in the front end part of complex SoC designing that helps in facilitating the process.

Project require working with teams involved in major projects around the world. Implementation and debugging of flows is also a part of this project.

1.2 Problem Statement

In order to tackle rising time-to-market pressures and handle the increasing design complexities, we need to find more efficient methodologies and flows starting from beginning to end level of designing. Objective of this project is to provide design related solution to various front end flows and providing interactive features that helps designer to perform the front end activity with no need to be aware of the different tool flows.

This also requires integration of new property checking flow with existing flow. Understanding new tools by implementing the methodology on design is also carried out as a task.

1.3 Thesis Organization

The thesis is organized in 5 chapters, the details of each chapter is as follows

Chapter 1: This chapter gives information about VLSI flow and the front end flow used in SoC design process.

Chapter 2: Advance Automated Mechanism, It describes the Advance automated mechanism flow and give details of property verification technique.

Chapter 3: Enhancement and development in system, this describes the new property verification flow integration and the new feature added to the systems along with their need and how they result in design time saving.

Chapter 4: Implementation of property checking and results, this describes the property checking flow performed on simple design analyzing the results

Chapter 5: Conclusion

1.4 VLSI Design flow

The design process at various levels is usually evolutionary in nature. The VLSI IC circuits design flow is shown in the figure below. The flow involve a lot of complex task, various tools at various levels of design are used comprehensively.

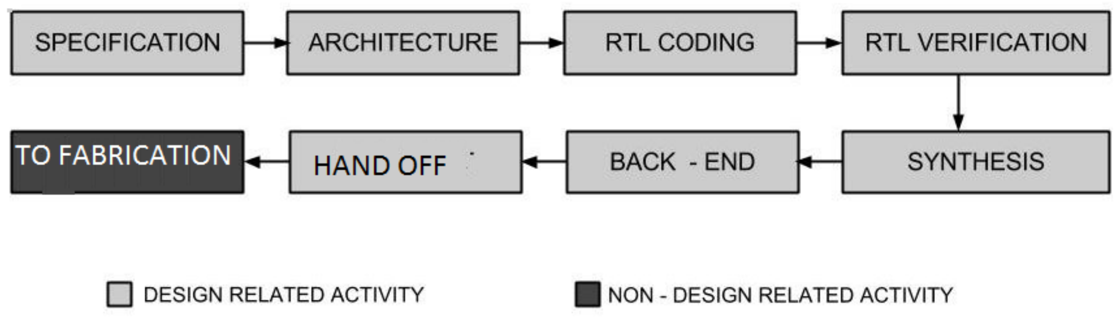


Figure 1.1: VLSI Flow

1) Specification : Lot of activity from gathering market requirement to deciding the technical aspect is done first. This is the crucial step as it will affect the future of the product. Here, vendors may want to get feedback from potential customers on what they are looking for. Once this is done final specification sheet with all possible technical details is made and handed over to the next team.

2) Architecture : This is where the main work starts. With the help of the specification sheet the target ICs architecture is decided and a layout for same is created by design engineers using EDA tools. In the next step this architecture is implemented and tested with the help of programming language and tools.

3) RTL Coding : RTL is an acronym for register transfer level. This implies that the VHDL or verilog code written based on the architecture describes how data is transformed as it is passed from register to register.

4) RTL Verification : Register Transfer Level (RTL) simulation and verification is one of the important step. This ensures that the design is logically correct and without major timing errors. It is advantageous to perform this step, especially in the early stages of the design. Simulation tools are used to perform RTL verification. A test bench file may be used here for verification.

5) Synthesis: This is where the design now start to get physical. Logic synthesis is a process by which the desired circuit behavior i.e. Register Transistor Level is turned into a design in terms of logic gates which drives the circuit or architecture. This is done with the help of FPGA/CPLD/ASIC hardware tools. These target boards may be accessed using the IDEs provided by specific vendor.

6) Back end: Here the final tested design after synthesis is given to the IC manufacturer.

7) Hand Off : Hand off is the process under back end only where the final result of frontend (first 5 steps) is provided to the manufacturer in form of photo mask. Then the manufacturer performs wafer processing, packaging, testing, delivery of samples to test the physical IC.

8) To Fabrication: Once the sample are tested and the entire requirement are furnished the design is sent for mass production.

1.5 Front End VLSI Flow:

The frontend flow is responsible to determine a solution for a given problem or opportunity and transform it into a RTL circuit description. The flow starts from specification, it involves most of the verification work that is required to achieve good initial design confidence. Later on the process in handled over to back end flow, where a number of process required for physical implementation of design are carried out.[1]

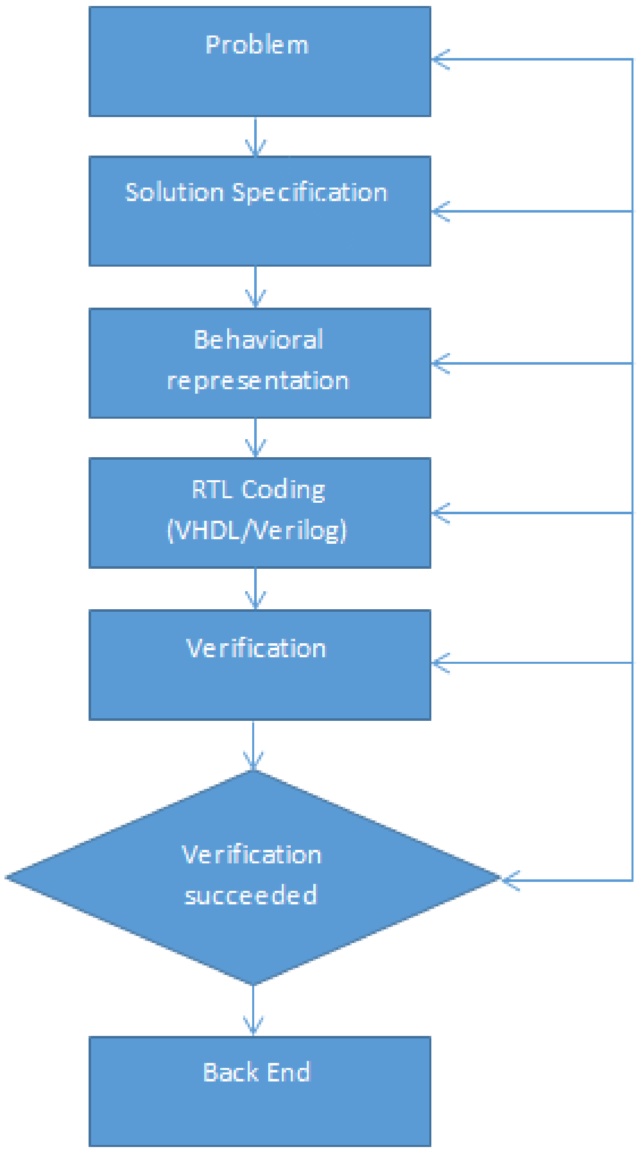


Figure 1.2: Front End Flow

The design complexities are raised enough so that a single tool is unable to provide good remedy for all the design, compilation and verification related issues. Increasing System-on-Chip (SoC) complexity and software content combined with rising time-to-market pressures are driving the need for an advance automated mechanism for design, compilation and verification solution that encompasses different VLSI flows and help early device bring-up, in addition it also provide solution to some other activity which saves time as well as efforts, during the designing period. The term different VLSI flows means the different design, compilation verification flows of tools by different CAD tool provider companies.

Lots of CAD tools are used during the front end cycle for different works like static checks, linting checks, Static timing analysis (STA), Formal verification, property checking, equivalence checking, and functional verification. All these checks give early design confidence and decrease the risk of rework.

The Advance automated mechanism supports for different design methodology; it is generic and encompasses different VLSI flows. A higher degree of design confidence and bringing down the risk of re-spin or repeated efforts is the main aim of the advance system, which ultimately contributes for less time to market. The mechanism is central, intelligent, advance, automated and the most importantly intellectualcite in1

Central: Used by almost all front end flows

Intelligent: It makes design compilation and validation of SoC easier and faster.

Advance: Automated and intellectual. Add-on to the older process which were slow

My work goes around the different periphery of the advance mechanism, developing and enhancing advance features and integrating a new flow for assertion based property checking are the major areas of my work.

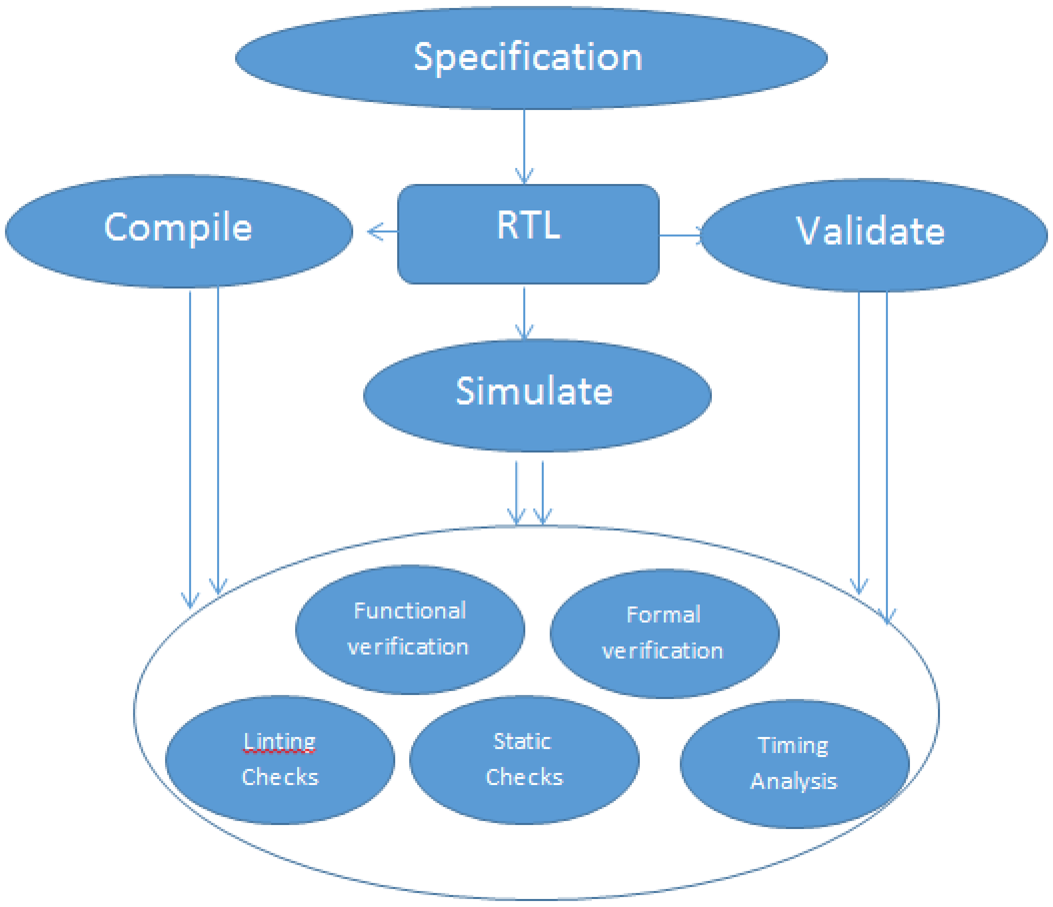


Figure 1.3: Front End Activity

1.6 Summary

This Chapter provides the detail of VLSI design flow and frontend flow process. An introduction to advance automated mechanism and its usefulness is also included in this chapter.

Chapter 2

Advance Automated Mechanism

After user specified the design requirement, the work of gathering the specification out of those requirements starts. If a design is simple a single user can manage the design process by himself. But in case of complex SoC design, a single user cannot perform all the design work instead a lot user work simultaneously on different modules and perform all the process of checking, validation and verification over the design and in the end all the parts are integrated.

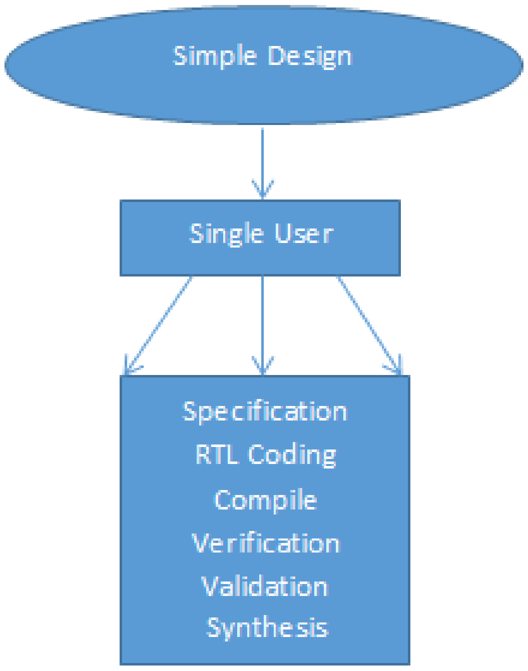


Figure 2.1: Simple Design

In order to manage complex designs there is an Advance automated mechanism which has all the different VLSI flow embedded. This mechanism provides a common platform for all the frontend activity and also provide end to end support to the users, with no need for the users to be aware of different tools methodology and flows. [3]

The A2M consolidates different front end flows in it. The methodology of this mechanism is very clean and all the stages of are determined. Each stage has some specific task linked to it. The working flow for the mechanism is shown below

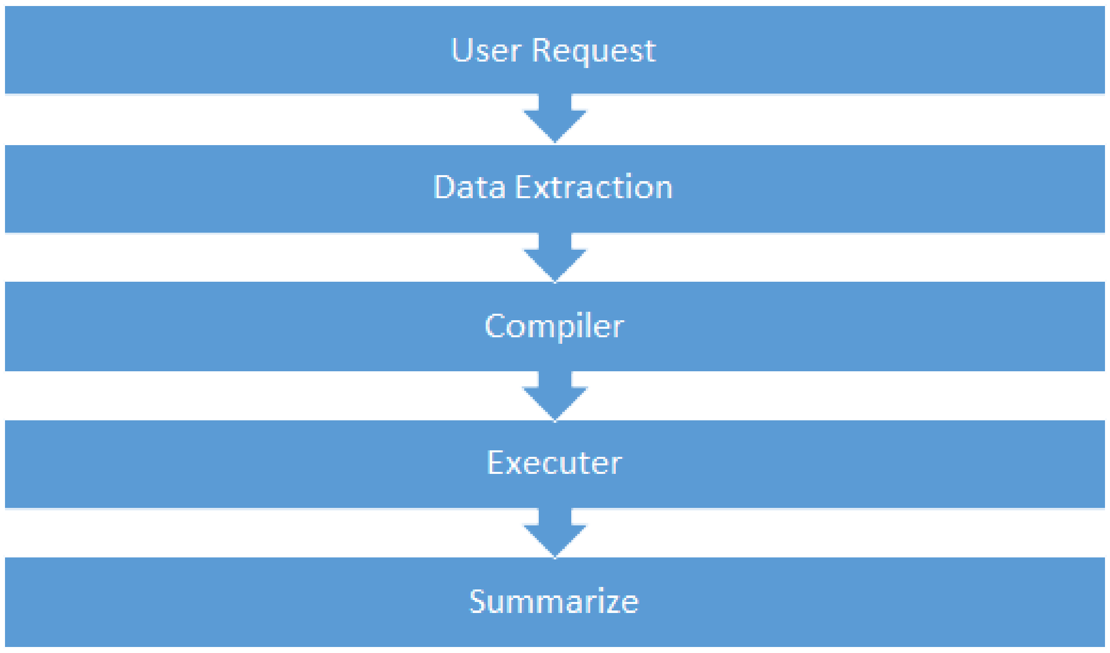


Figure 2.2: Flow of Mechanism

2.1 Different Stages of the Advance Flow

2.1.1 User request:

In this stage user specifies the type of action he/she want to perform over the design, like compilation, test execution, gathering results and simulation using a set of commands. Commands are grouped together to produce a meaning and corresponding actions are performed after decoding the command. Validation of user's commands is also done in this step..

2.1.2 Data extraction:

It is one of the most important steps of the A2M. In this step in accordance with input configuration (set of rules) files, the design data that is the combination of RTL source code, corresponding standard libraries and other test related files are being extracted and stored for processing. Configurations along with the user action dictate the task to be performed by advance flow.Design reuse in one of the feature of the advance system which is done in this step. A large numbers of algorithms are used in this step, which makes the data extraction easy and fast. [5]

2.1.3 Compilation:

In this step the advance flow prepares the set of actual commands that are executed in order to perform the actual task. Various log files and tool flow specific files are generated during this step.

2.1.4 Execution:

Executor executes the different commands specific to the flow in an orderly manner and collects the output in log files. After the flow execution it also reports errors if any.

2.1.5 Summarize:

Finally the advance flow does the work of summarizing the results for user. The summary contains the errors, warnings, pass/fail status of the test.

2.2 Property Checking:

Method to prove properties for the correctness of design or show root cause of an error by rigorous mathematical procedures is property checking. It is used to show that a RTL model satisfies some properties, such as implementing a microarchitecture specification or maintaining an invariant.

In recent years property checking has become more accessible to non-specialists, through the Assertion Based Verification methodologies. These enable RTL authors to add assertions to their code, and use powerful formal methods to prove them.[8] What are Properties:

A property defines behaviour of the design. A property can be used for verification as an assumption, a checker, or a coverage specification. It can be declared in a module, interface, program, clocking block, package for example

```
property rule6_with_type(bit x, bit y);
##1 x |-> ##[2:10] y;
//antecedent |-> consequent
endproperty
```

2.2.1 Verification Statements:

A property on its own is never evaluated for checking an expression. It must be used within a verification statement for this to occur. A verification statement tells what verification function to be performed on the property. [9] The statement can be one of the following:

a. Assert

b. Assume

c. Cover

Assertions:

Assertions are checkers on the RTL that prove certain properties about the design module. It can pass, fail or be unproven within a specific bound and are used to specify requirements from design module and insure design correctness. A passing assertion assures that the property will always hold on the other hand a failing assertion produces a counterexample that can be used to debug and root cause the failure. It also improves observability and shorten time to develop.In addition, assertions can be used to provide functional coverage and generate input stimulus for validation.

Assertions are similar for simulation, which is event-based, and formal verification, which is cycle-based.[11]

a. Specify requirements from DUT, behaviour expected from the design.

b. Insure design correctness

c. assert property (p);

Advantages of assertion

a. Assertion Improves observability

b. Reduces the debug time.

c. Bugs can be found earlier and are more isolated.

d. Controllable severity level.

e. Describe the Documentation and Specification of the design.

f. They are also used to gather information for various level of properties like conditional, signal, sequence.

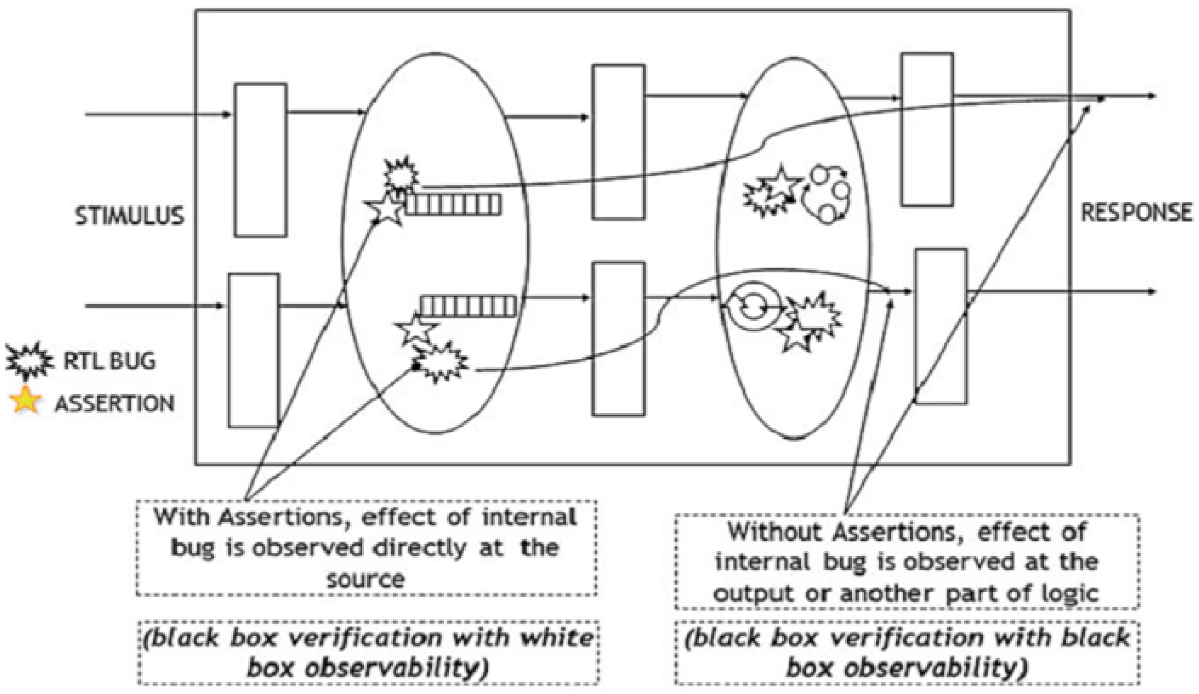

STIMULUS
RESPONSE
RTL BUG
ASSERTION
With Assertions, effect of internal bug is observed directly at the source
(black box verification with white box observability)
Without Assertions, effect of internal bug is observed at the output or another part of logic
(black box verification with black box observability)

Figure 2.3: Assertion improve observability
[7]

2.2.2 Types of assertions

There are basically two types of assertion

a. Immediate assertions: They are based on simulation events and used only with dynamic simulation tool.

b. Concurrent assertions: They are based on clock cycles and can be used with formal or dynamic simulation tool.

2.2.3 Assertion System Task

Assertion severity task

Because the assertion is a statement that says some behaviour of the design must be true, the failure of assertion also has some severity associated with it. The severity can be specified by by using a system task in the fail statement. $fatal, $error, $warning and $info are the system task used to specify the severity of an assertion failure.

$fatal is used for a run time fetal, $error is used for a run time error, $warning is used for a run time warning that can be suppressed and $info is used to produce informational statements and signifies that assertion failure carries no specific severity. Default severity level of an assertion failure is an error.

Assertion control system tasks

Assertion control system system task are used to enable or disable the assertion base on some condition. There are basically three assertion control system task.

a. $assertoff to stop checking of all assertion until a subsequent $asserton is encountered. Already executing assertions are not affected by this system task.

b. $assertionkill system task abort all the assertion including currently executing assertion until a subsequent $asserton is encountered.

c. $asserton system task re enables the execution of all specified assertions

2.2.4 Boolean System Function

Boolean system functions gives the facility to check the behaviour of signal. they are used to check the values and type of transition. some common boolean system functions are

a. $countones: This function return number of 1's in a bit vector.

b. $past: This function return the past value of the signal.

c. $stable: It return a true if a signal is stable.

d. $isunknown It returns true if unknown values are present in the signal.

e. $rose: It returns true if the LSB of the expression goes to logic 1 else it returns false.

f. $fell: It returns true if LSB of the expression goes to logic 0 else it returns false.

g. $onehot: It returns true if only one bit of expression is high.

h. $onehot0: It returns true if at most 1 bit of the expression can be high.

Example:

Assumptions:

A statement that the design is constrained by the given property and a directive to verification tools to consider only paths on which the given property holds.Usually the assumptions are properties written over inputs to restrict the state space, or properties whose proof would depend on logic external to the model. assumptions are used in the same way as assertions.

They are used for the following

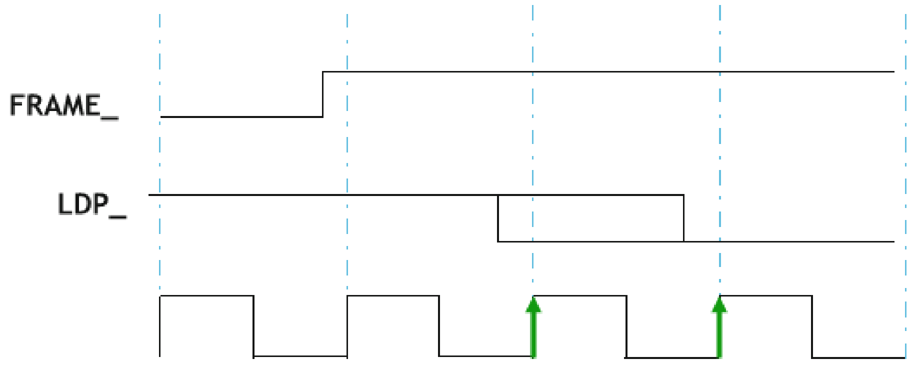
a. Model design environment

b. Specify requirements from environment

c. Restricts the set of feasible traces in the model

Cover:

Cover statements are used to specify scenario we wish to observe or to monitor their coverage evaluation. They mathematically prove that the property holds on some feasible trace. A cover passes as soon as a valid trace that satisfies the condition is found and fails if given the RTL and constraints, the condition can never be matched. Covers that are unproven within the specified bounds should be examined in more detail. They may be soft failures, where new constraints have suppressed valid coverage cases that occur within the property checking bounds. They may reveal property checking modeling inadequacies, where due to max bound or complexity considerations, interesting cases are not covered.



When FRAME_ is de-asserted, LDP_ (last data phase) must be asserted within the next 2 clocks

```
property ldpcheck;
  @(posedge clk) $rose (FRAME_) |-> ##[1:2] $fell (LDP_);
endproperty

aP: assert property (ldpcheck) else $display("ldpcheck FAIL");
cP: cover property (ldpcheck) $display("ldpcheck PASS");
```

Figure 2.4: Property Example
[7]

2.2.5 Property checking advantages:

The advantages of using property checking are as follows

a. It does not require test benches or stimuli. No concept of simulation-style tests

b. Turnaround time is very less.

c. Specify the destination, rather than journey, just specify what we want to see or prove. Instant test bench

d. Engines drive all possible combinations of inputs which are equivalent of every possible test and an impossible tests too plan to spend time ruling these out

e. In simulation, poking an internal RTL signal only affects that signal and any downstream logic. In FV, constraining a signal affects the upstream logic as well. White-box constraining possible in FV. Can constrain output of decoder to force only valid encoded inputs to arrive. Think of constraint on possible universe of simulations, not of constraining specific stimuli.

2.3 Summary

This chapter explains the The advance automated mechanism, its need and the flow adapted. Explanation of each flow step is also given in this chapter.

The later part of this chapter explains the formal verification technique of property checking. Verification statements are also covered in this chapter.

Chapter 3

Enhancement And Development In System

The High level block diagram of the advance automated mechanism is shown below.

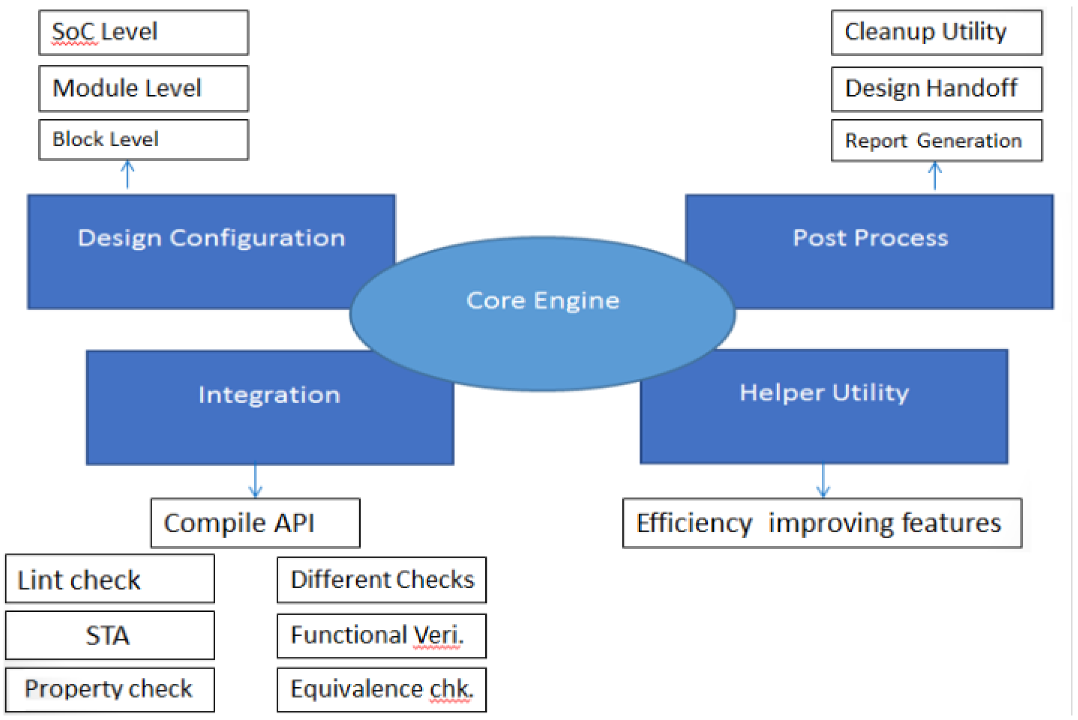


Figure 3.1: High Level Block Diagram
[5]

The design configuration contains all the user specified information like file paths for standard cells library, RTL file information and other project related customized information. The post process part does the cleanup, design records handover and post simulation work. The integration part contain the information of different flows present in A2M. All the simulation, compilation, design checks, validation, formal checks flows are there. New flows can be added through the integration part of A2M.

The helper utility is like a bridge between the design specification and core engine, it contains many small features to display design configuration corresponding to current part of design. Features of helper utility are widely used during compilation and execution time.

The core engine is the heart of the A2M; it initiates different flows present, process the design in a proper order and also maintains the information for the processes. It interacts and controls all the parts of advance flow hence called the engine.

In the system I worked on enhancement, development debugging of post process, helper and integration part that require lots of testing related work too.

3.1 Integrating a new flow

Problem Statement

Property verification is the flow which is not been automated and integrated in any of the Intel's flow and many of the SoC teams were requesting to automate and integrate the this flow in the advance flows present. This task involves understanding the property verification methodology, fine understanding all the steps involved in the flow, understanding the the various options present in the tool, designing a prototype for testing. Above all the thing important is to integrate in an existing flow without out inter-fearing with the functionality of the other flows present, this requires good understanding of the advance flows used in Intel as well.

3.1.1 Property Checking

Property checking is a method to prove the correctness of design or show root cause of an error by rigorous mathematical procedures. In property checking properties are evaluated over the design using verification statements like Assert, Assume and Cover.

Benefits of property checking

a. High design confidence

b. It does not require test benches or stimuli. Instant testbench

c. Turnaround time is very less.

d. It lowers cost by reducing the debug time

3.1.2 A2M steps for any flow execution

There are basically 3 steps in A2M

1. setup the environment for all the flow present and gather the design data.
2. run the flow and generate log files for all the individual design file and step.
3. analyze the log and generate the pass fail status and point reason for failure.

In the mechanism there is a way to specify the design data in a file, and that files serves as input to the advance mechanism. This file basically contains the design definition that tells what are all the library included in the design along with the mapping information for physical library with a virtual name. Similarly there is a way to provide library definition for each library in a file.

Advance mechanism gathers all the design related information from these files and prepares a data base for each design. The data base so prepared contains all the information in a very arranged and hierarchical format.

After gathering all the information a corresponding tool flow called by analyzing the the command line specified to the advance mechanism. A command mapping API present in A2M maps the command with the corresponding tool flow. After this action is performed the whole design data is passed to the tool flow, that generates all the actual command line required by the tool and a shell script that actually runs the flow over the design.

3.1.3 Property verification flow

Following steps are involved in property verification flow

The first step is to compile the design using the corresponding compile command for all the design library and properties.

Second step is to connect all the design modules together using the design top, this is basically the elaboration phase where the flow makes the design pyramid by connecting all the design data.

Third step is to define the environment for property verification, in this step all the clocks and reset are defined for the design.

Fourth step is to constrain the design environment using assume and stopat condition.

Fifth step is to prove the property, property can be proved over the design in one shot or on the basis of bind instance. After this the flow also provide the advance debug facility where user can get the counterexample for failing case through which user can find out bug in the design and fix the cause of bug. Covers for the passing property are also generated that shows whether the condition specified by verification statement is executed or not.

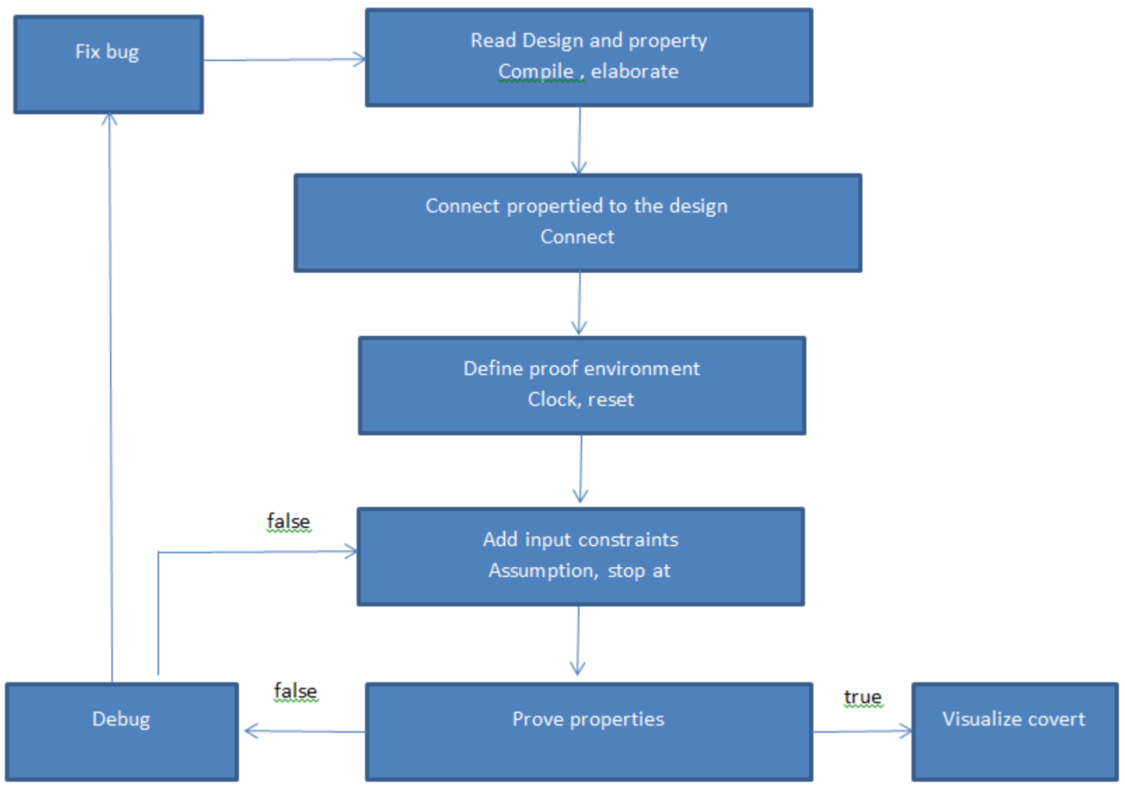


Figure 3.2: Flow steps

3.1.4 Sneak path analysis flow in property verification

property checking involves having a model check which can mathematically analyze the design model against assertion and find every possible condition that violates the assertion without using test bench, if does not find any violation it says property is proved.

In case it finds a violation tool will give a stimulus example that serves as a counter example used for debugging. In this case first we will review the stimulus that tool shows in counterexample and check whether the waveform is correct or not. If wave form is incorrect then we put that condition to an assumption and declare it as a invalid stimulus but if the wave form is correct we have to check the whether the path is valid or not if it is is not valid then it is added to omit condition. If the path is a valid path then its a design bug and need to be fixed.

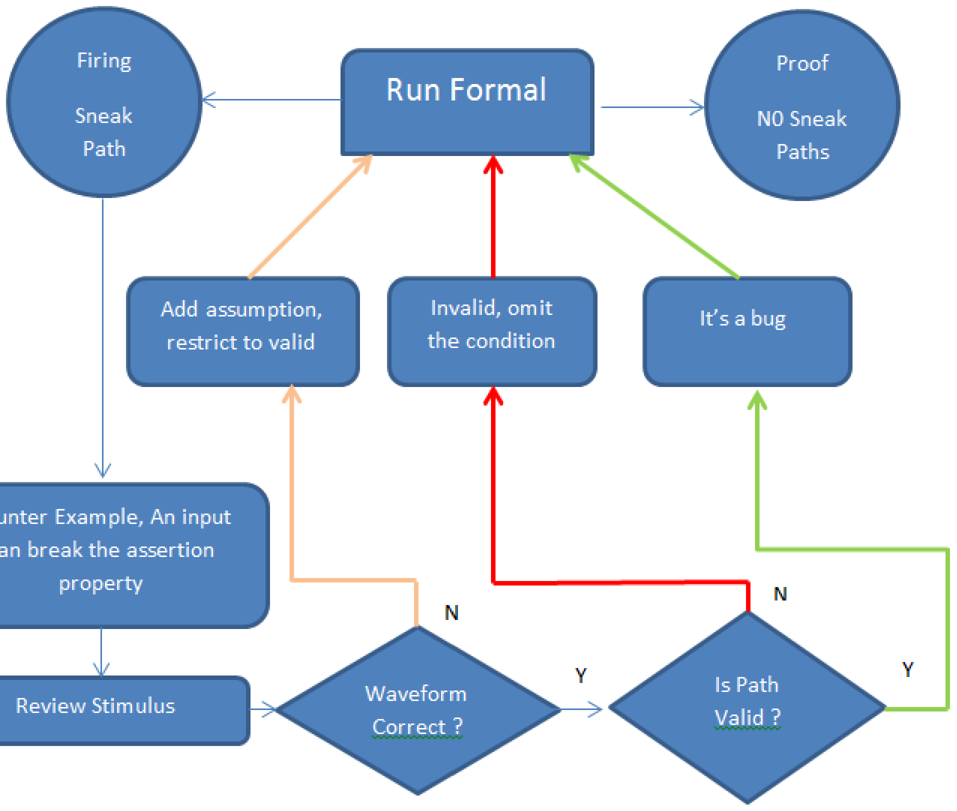

Firing
Sneak
Path
Run Formal
Proof
NO Sneak
Paths
Add assumption, restrict to valid
Invalid, omit the condition
It's a bug
Counter Example, An input can break the assertion property
Review Stimulus
N
Waveform Correct ?
Y
N
Is Path Valid ?
Y

Figure 3.3: Flow steps

3.1.5 Integration steps

The flow is integrated and automated in two steps

a. Compile Process Automation

b. Execution process automation

Compile Process Automation

To automate the compile process first step is to define all the design library and RTL files under a common design top then collect all the design data (library and design files) present in it. Design data of each library include all the design files in the library along with their compile options, compiler directives (+define+, +incdir), path to the imported package and search paths for ‘include statement.

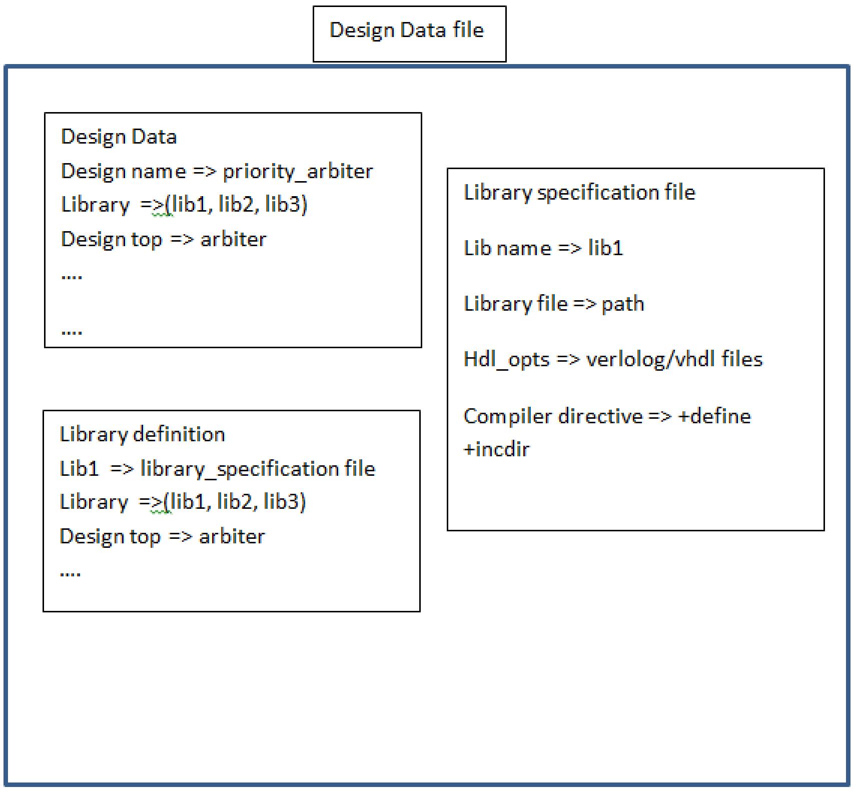

Design Data file
Design Data
Design name => priority_arbiter
Library =>(lib1, lib2, lib3)
Design top => arbiter
....
....
Library specification file
Lib name => lib1
Library file => path
Hdl_opts => verlolog/vhdl files
Compiler directive => +define
+incdir
Library definition
Lib1 => library_specification file
Library =>(lib1, lib2, lib3)
Design top => arbiter
....

Figure 3.4: Design Data file

All this information is gathered for all the libraries in the design and a .f file is generated that is given as a input for compilation of corresponding library. similarly a .list file is also generate that contains the complete path of the design files in a library. So if the design has n library the n number of .f files and n number of .list files are generated.

Example of .f file

// LIBRARY: LIBRARY NAME

// DATA FILE: PATH TO DATA FILE

// DATA FILE: PATH TO DATA FILE

// -verilog opts: -sverilog -sv +define+Macro

// -verilog opts: +libext+.v +define+PROPERTY_VERIFICATION

```
+incdir+ PATH FOR INCLUDE DIR
+incdir+PATH FOR INCLUDE DIR
PATH FOR ‘include statements in design
MORE compile Options
```

Example of .list file

```
// LIBRARY: LIBRARY NAME
// DATA FILE: PATH TO DATA FILE
// DATA FILE: PATH TO DATA FILE
// DESIGN FILE PATH
PATH TO DESIGN FILES
```

These two files and the common compile options provided in the command line are given to the compile design command for the flow.

After this we generate a TCL script for the tool compilation, that compiles all the library and elaborate the design using the design top information given by the user. compilation log for each library is also generated and saved at a pre defined place and a elaboration log is also generated. The compilation and elaboration commonly generates a binary result file, that can be utilized for in property proving stage.

Example of TCL file for compilation

```
auto-generated Property verification TCL file
For: PROPERTY VERIFICATION
Only Synthesizable verilog
```

```
// Common Variables
START TIME [date]
DOT F DIR PATH TO .f file directory
project ARBITER
// Clear all previous analysis
clear
// Library
```

redirect -file PATH FOR GENERATING LOG FOR THIS LIBRARY -force compile -lib lib1 -f PATH TO .f file -f PATH TO

.LIST file

// Library

redirect -file PATH FOR GENERATING LOG FOR THIS LIBRARY -force compile -lib lib2 -f PATH TO .f file -f PATH TO

.LIST file

// Library

redirect -file PATH FOR GENERATING LOG FOR THIS LIBRARY -force compile -lib lib3 -f PATH TO .f file -f PATH TO

.LIST file

// Library

redirect -file PATH FOR GENERATING LOG FOR THIS LIBRARY -force compile -lib lib4 -f PATH TO .f file -f PATH TO

.LIST file

// Start capturing compilation and elaboration results

set capture design on

// Start elaboration using the top

redirect -file PATH FOR GENERATING LOG FOR THIS LIBRARY -force elaborate command -top DESIGN TOP

// Save the results binary file

save -force -elaborated_design PATH FOR GENERATING BINARY FILE

A shell script is prepared after generating these files, the function of shell script is to set the environment variable and invoke the property checking flow using the TCL script that contains the commands to be executed over the property checking flow and collect the corresponding results.

Execution Process Automation

The compilation process generates a binary file that contains all the compile and elaboration result. For proving or executing property, tests are executed over the design. For execution also there is one TCL script generated that reload the compile results, set the proof environment and start property verification over the design and produce the results.

Example of Tcl file for execution

```
// Common Variables
START TIME [date]
DOT F DIR PATH TO .f file directory
project ARBITER
// Restore elab results
restore -elaborated_design PATH TO COMPILE RESULT FILE
// Clock
clock clk
// Reset expression
reset !reset_n
// Max tarce length
maximum trace length 100
// Check that clocks, resets, etc. are reasonable
sanity heck command
// Prove property
prove Instance_name -time_limit 24h
// Prove property
prove Instance_name -time_limit 24h
// Save Last Used TCL Commands
save -force -script PATH TO SAVE RESULTS
// Report File
report -file PATH TO REPORT FILE -force -detailed
```

The TCL script so generated is used to invoke the tool flow, and prove the property specified by the user. Results of property checking are also stored in a log file, and analysis of those log by post process gives the pass fail status of the test.

3.2 Post Process

Problem Statements

a. Cleanup utility advancement for ability to look deeper into the run area.

b. Force post process advancement for rerun with new configuration.

c. Removing and reporting configuration conflicts from post process system.

A2M provides common environment for the front end process among different projects, customization of some process is also possible which results in requirement oriented outcomes only.Before we start any step in VLSI design there is a need that our design environment be clean, not even a single unwanted file present in the run space, if so it can cause misinterpretation and results in fetal error and time wastage. So here comes the existence of cleanup process, which refreshes the full design environment.

All the cleanup process were automated, here we advanced automated the cleanup process for design time saving in the A2M. Cases exists where it happens that some design files need not to be removed from the run space and its a time waste in regenerating those files every time at the same place where we do the design compilation or simulation.

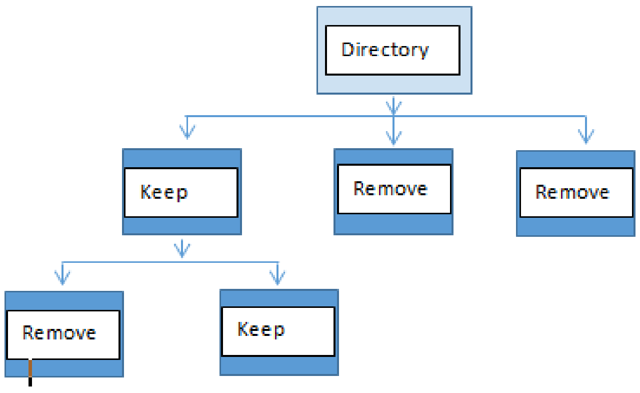


Figure 3.5: Cleanup Utility

For this enhanced the A2M, the enhancement also allows the designer to look inside the run space and delete only the unwanted files; just need to specify the useful file or the other way round. The advance system will automatically handle the specified condition and prepares the design environment accordingly that saves time.

After every compilation simulation or test run, log files are generated which are utilized to see the pass, fail and other important specification of compiled design. Looking into the log for all such detail is a time consuming things, A2M facilitates us this by post simulation process. This feature of the mechanism look into the log for all the required detail and user specified option, and finally produce a report that contain all details like pass fail status, error, type of error, simulation time, test details and other useful information.

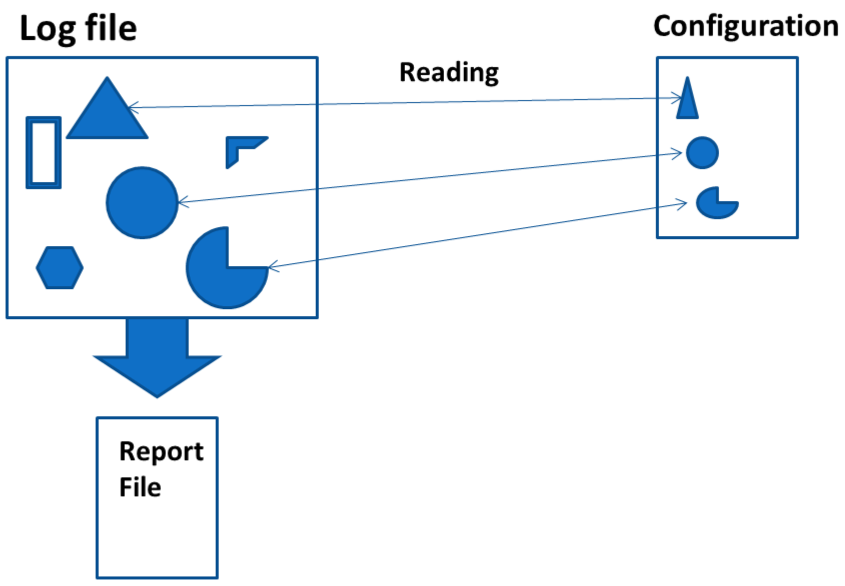


Figure 3.6: Report Generation

The simulation post process system also gives the analysis options to user. The user can run this process on log again and again with new specification, configurations and assumptions. Which actually decrease the need for the designer to rerun the full test again and analyze the log files.

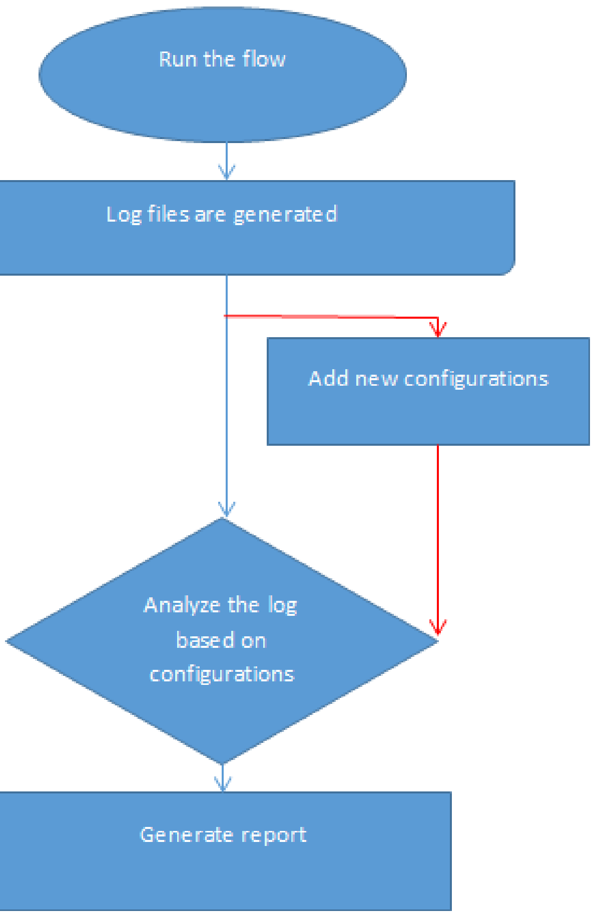


Figure 3.7: Force Post Process

The problem exist when there are some conflicts between two configurations, one configuration overrides the effect of other and produce misleading information. The A2M also handles this problem, It reports the designer with a warning message that a conflict exist in the post process system.

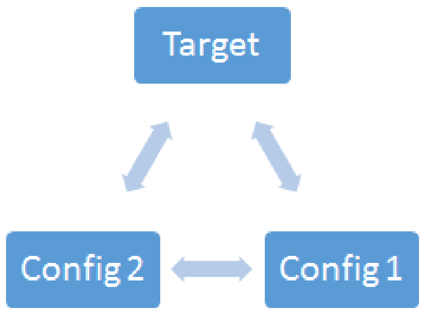


Figure 3.8: Configuration Conflict

3.3 Helper Utility

This utility was already there in the system. In this area of advance system I worked on developing few new features which are widely used and saves design time.

Problem statements for this utility are

a. Locate files on project configuration paths.

b. Exclude unwanted compile option for various flows.

c. Top scope information for data handover feature.

d. Tests to verify invariability of the flow.

3.3.1 Enhanced and developed features

Feature that locate files on project configuration paths

Every project has some configuration that contains the information of required design files, directory and tests. While working on any flow it is needed to look for these again and again in the project development area. As the design is complex and so the number of such files is also huge. The newly developed feature can locate design files and directories in the project development area and provide the information weather the file is present or not in the current area.

Feature which removes unwanted compile option:

In the SoC design, there is always a case of design reuse, where the designer uses some of the modules or property of standard designs as it is. There exists no need to compile or simulate such inherited object. It is always suggested to mark such object and not to do any compilation or simulation for them. Treat them as black box, with no need of elaboration.[5]

Every time we do RTL compilation for any flow, we need to specify various Verilog options which are used for elaboration of design. Sometimes it happens that all the default Verilog options along with designer specific Verilog options for a flow are not required for compilation of design. It is useless to spend the precious time on compilation of such options and there comes the need of the exclude compile Verilog option, which can remove such unwanted options and saves the compilation time

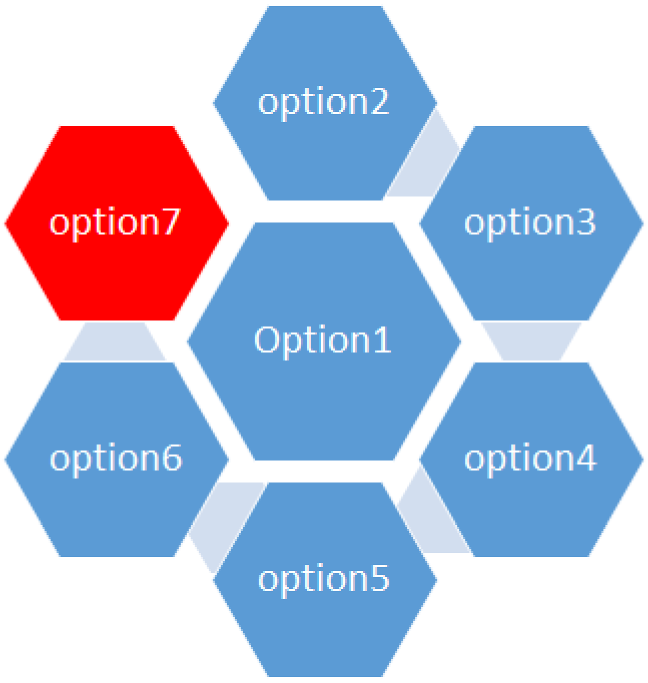


Figure 3.9: Exclude Compile Feature

Enhanced Data handover feature:

In a SoC design process a huge number of modules, models and libraries are utilized, all of them are integrated in a hierarchical manner, relationship is maintained. Most of the time it is a needed to look on all the design objects utilized in a hierarchical manner, and transfer the corresponding information to some other module or flow. The data handover feature is used for this purpose. It gives us the option to look the relationship in both ways, top to bottom or bottom to top. A lot of design and library related information is also handled by this feature of A2M. According to the requirement I did an enhancement in the handover part to produce the library information along with its top module details and report the hierarchy in better way.

Test case development for Enhanced features:

Every enhancement or development done in the system must not affect the old functionality of the flow, this thing must be taken care of. For this writing test case for a scenario that validates and assures the proper functioning is required. I had written test cases for the enhancements done for end to end checking.

3.4 Summary

This Chapter provides the details of the work done in the current system. The work include development and enhancement of new features in the post process and helper part of the A2M. Details of the features, their need and the their efficiency is also described.

Chapter 4

Implementation Of Property Checking And Results

The major input to property verification flow is a design description with a set of properties basically verilog RTL models with embedded system verilog properties. property checking flow compiles and elaborates the design hierarchy, synthesizes the net list, extracts the embedded properties, and lists them in the property pane section according to design hierarchy. After elaboration verification environment setting is required then only user can verify property over the design. 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.

In order to observe and analyze property checking flow integrated in advance mechanism the following tasks are done

a. Written RTL design

b. Added property and verification statements.

c. Run property checking flow for property checking

d. Observed the response

4.1 RTL for design and property adding

The RTL for a simple priority based arbiter that shares the bus among different requester based on current state and request along with the verification statement is shown below. The verification statements used are cover, assert and assume. All the statement can also be written in a separate file and can be embedded into the RTL while elaborating the design. In this design property are included in the RTL itself.

```
module arb (clk,reset_n, req0,req1,req2,req3,
gnt0,gnt1,gnt2,gnt3 );
input clk,reset_n,req0,req1,req2,req3;
output gnt0,gnt1,gnt2,gnt3;
reg gnt0,gnt1,gnt2,gnt3;
reg [2:0] arbiter_state; // Declaration of states
always @ (posedge clk or negedge reset_n)
begin
if (reset_n == 0)
begin // Turning off all grant signal
gnt0 ¡= 1'b0;
gnt1 ¡= 1'b0;
gnt2 ¡= 1'b0;
gnt3 ¡= 1'b0;
arbiter_state ¡= 3'b0;
end
else
case(arbiter_state)
0: begin // Waiting state state 0
gnt0 ¡= 1'b0; // Turning off all grant signal
gnt1 ¡= 1'b0;
gnt2 ¡= 1'b0;
gnt3 ¡= 1'b0;
if (req0==1) // requester 1 raising request
arbiter_state ¡=1;
else if (req1==1)
arbiter_state ¡=2; // requester 2 raising request
else if (req2==1)
arbiter_state ¡=3; // requester 3 raising request
else if (req3==1)
```

```
arbiter_state ¡=4; // requester 4 raising request
else
else
arbiter_state ¡=0; // Waiting state
end
1: begin // Switch off all grant signal except request 1
gnt0 ¡= 1'b1;
gnt1 ¡= 1'b0;
gnt2 ¡= 1'b0;
gnt3 ¡= 1'b0;
if (req0==1)
arbiter_state ¡=1;
else if (req1==1)
arbiter_state ¡=2;
else if (req2==1)
arbiter_state ¡=3;
else if (req3==1)
arbiter_state ¡=4;
else
arbiter_state ¡=1;
end
2:begin
gnt0 ¡= 1'b0;
gnt1 ¡= 1'b1;
gnt2 ¡= 1'b0;
gnt3 ¡= 1'b0;
if (req1 ==1)
arbiter_state ¡=2;
else if (req2 ==1)
arbiter_state ¡=3;
```

```
else if (req3==1)
arbiter_state ¡=4;
else
arbiter_state ¡=1;
end
3:begin
gnt0 ¡= 1'b0;
gnt1 ¡= 1'b0;
gnt2 ¡= 1'b1;
gnt3 ¡= 1'b0;
if (req2==1)
arbiter_state ¡=3;
else if (req0==1)
arbiter_state ¡=1;
else if (req1==1)
arbiter_state ¡=2;
else if (req3==1)
arbiter_state ¡=4;
else
arbiter_state ¡=1;
end
4:begin
gnt0 ¡= 1'b0;
gnt1 ¡= 1'b0;
gnt2 ¡= 1'b0;
gnt3 ¡= 1'b1;
if (req3==1)
arbiter_state ¡=4;
else
arbiter_state ¡=1;
```

```
end
default : arbiter_state ¡= 0;
endcase
end
endmodule
property module
module prop1 (
clk,
reset_n,
arbiter_state,
req0,
req1,
req2,
req3,
gnt3,
gnt2,
gnt1,
gnt0
);
input clk;
input reset_n;
input [2:0] arbiter_state;
input req0;
input req1;
input req2;
input req3;
output gnt3;
output gnt2;
output gnt1;
output gnt0;
```

```
// Functional coverage points
//
// Request
c_req0: cover property (@(posedge clk) (req0));
c_req1: cover property (@(posedge clk) (req1));
c_req2: cover property (@(posedge clk) (req2));
c_req3: cover property (@(posedge clk) (req3));
//
// Grant
c_gnt0: cover property (@(posedge clk) (gnt0));
c_gnt1: cover property (@(posedge clk) (gnt1));
c_gnt2: cover property (@(posedge clk) (gnt2));
c_gnt3: cover property (@(posedge clk) (gnt3));
// State
c_state0: cover property (@(posedge clk) (arbiter_state ==0));
c_state1: cover property (@(posedge clk) (arbiter_state ==1));
c_state2: cover property (@(posedge clk) (arbiter_state ==2));
c_state3: cover property (@(posedge clk) (arbiter_state ==3));
c_state4: cover property (@(posedge clk) (arbiter_state ==4));
property reqs ;
@(posedge clk) disable iff (!reset_n) (req0 ==3);
endproperty
asp : assume property (reqs);
property reqs3 ;
@(posedge clk) disable iff (!reset_n) (($rose(req0) && $rose(req1) && $rose(req2)
&& $rose(req3)) | => $rose(gnt3));
endproperty
ap : assert property (reqs3);
endmodule
property module
module prop (
clk,
reset_n,
arbiter_state,
req0,
req1,
req2,
req3,
gnt3,
gnt2,
gnt1,
gnt0
);
input clk;
input reset_n;
input [2:0] arbiter_state;
input req0;
input req1;
input req2;
input req3;
output gnt3;
output gnt2;
output gnt1;
output gnt0;

logic [3:0]gnt_group = {gnt3,gnt2,gnt1,gnt0};

property p_reset;
@(posedge clk) disable iff(reset_n) (!reset_n) |- > (##[0:1]arbiter_state == 0) ;
endproperty
ap_reset: assert property (p_reset);
cp_reset: cover property (p_reset);
```

```
property arb_state_1_request0;
@(posedge clk)disable iff (!reset_n) ((arbiter_state ==1) && req0 && $rose(req1)
&& $rose(req2) && $rose(req3) ) |− > ##[0:1] (gnt0 );
endproperty
ap_arb_state_1_request0 : assert property (arb_state_1_request0);
cp_arb_state_1_request0 : cover property (arb_state_1_request0);

property arb_state_2_request1;
@(posedge clk)disable iff (!reset_n) ((arbiter_state ==2) && req1 && $rose(req0)
&& $rose(req2) && $rose(req3) ) |− > ##[0:1] (gnt1 );
endproperty
ap_arb_state_2_request1 : assert property (arb_state_2_request1);
cp_arb_state_2_request1 : cover property (arb_state_2_request1);

property arb_state_3_request2;
@(posedge clk)disable iff (!reset_n) ((arbiter_state ==3) && req2 && $rose(req0)
&& $rose(req1) && $rose(req3) ) |− > ##[0:1] (gnt2 );
endproperty
ap_arb_state_3_request2 : assert property (arb_state_3_request2);
cp_arb_state_3_request2 : cover property (arb_state_3_request2);

property arb_state_4_request3;
@(posedge clk)disable iff (!reset_n) ((arbiter_state ==4) && req3 && $rose(req0)
&& $rose(req1) && $rose(req2) ) |− > ##[0:1] (gnt3 );
endproperty
ap_arb_state_4_request3 : assert property (arb_state_4_request3);
cp_arb_state_4_request3 : cover property (arb_state_4_request3);

property reset_state;
@(posedge clk) disable iff (!reset_n) (arbiter_state ==1) —=¿ (##[0:1] gnt0);
endproperty
```

ap : assert property (reset$_state$);

logic [3:0]gnt_group = gnt3,gnt2,gnt1,gnt0;

// one hot and reset check

property p_reset1;

@(posedge clk) disable iff(!reset_n) (!reset_n) |− > (##[0:1]arbiter_state == 0) ;

endproperty

ap_reset1: assert property (p_reset1);

cp_reset1: cover property (p_reset1);

property p_onehot;

@(posedge clk) disable iff (!reset_n)($onehot0(gnt_group)$||(\$onehot(gnt_group)))$;

endproperty

ap_onehot: assert property (p_onehot);

cp_onehot: cover property (p_onehot);

property arb_stt4;

@(posedge clk) disable iff (!reset_n) (arbiter_state==4) |− > (($past(arbiter_state)==4) $||(\$past(arbiter\_state) == 3)||(\$past(arbiter\_state) == 2)$

$||(\$past(arbiter\_state) == 1)||(\$past(arbiter\_state) == 0))$;

endproperty

ap_arb_stt4: assert property (arb_stt4);

cp_arb_stt4: cover property (arb_stt4);

property past_arb_stt4;

@(posedge clk) disable iff (!reset_n) ($past(arbiter_state)==4) |− >

((arbiter_state==4) $||(arbiter\_state == 1)||(arbiter\_state == 0))$;

endproperty

ap_past_arb_stt4: assert property (past_arb_stt4);

cp_past_arb_stt4: cover property (past_arb_stt4);

```
property past_arb_stt3;
@(posedge clk) disable iff (!reset_n) ($past(arbiter_state)==3) |− >
((arbiter_state==3) ‖(arbiter_state == 4)‖(arbiter_state == 1));
endproperty
ap_past_arb_stt3: assert property (past_arb_stt3);
cp_past_arb_stt3: cover property (past_arb_stt3);

property past_arb_stt2;
@(posedge clk) disable iff (!reset_n) ($past(arbiter_state)==2)|− >((arbiter_state==2)
‖(arbiter_state == 3)‖(arbiter_state == 4)‖(arbiter_state == 1));
endproperty
ap_past_arb_stt2: assert property (past_arb_stt2);
cp_past_arb_stt2: cover property (past_arb_stt2);

property arc1;
@(posedge clk) disable iff (!reset_n) ((arbiter_state ==1) && $rose(req1) &&
$fell(req0) ) —=¿(##[0:1]gnt1);
endproperty
ap_arc1: assert property (arc1);
cp_arc1: cover property (arc1);

endmodule
binding property with design
module bind_assertion();
bind arb prop1 p2(
.clk(clk),
.reset_n(reset_n),
.arbiter_state(arbiter_state),
.req0(req0),
.req1(req1),
.req2(req2),
.req3(req3),
.gnt0(gnt0),
.gnt1(gnt1),
.gnt2(gnt2),
.gnt3(gnt3)
);
bind arb prop p1(
.clk(clk),
.reset_n(reset_n),
.arbiter_state(arbiter_state),
.req0(req0),
.req1(req1),
.req2(req2),
.req3(req3),
.gnt0(gnt0),
.gnt1(gnt1),
.gnt2(gnt2),
.gnt3(gnt3)
);
endmodule
```

4.2 Results

Results of property checking is shown below

Properties Considered : 49

assertions : 13

- proven : 13 (100%)

- cex : 0 (0%)

- error : 0 (0

covers : 12

- unreachable : 3 (8.33333%)

- covered : 33 (91.6667%)

- error : 0 (0%)

4.3 Summary

This chapter shows the property verification work, that involve writing RTL, adding the verification statements and verifying the design using property checking flow integrated in the advance mechanism. Behaviour of the design is analysed and coverage of different scenarios is also observed.

Chapter 5

Conclusion

Time-to-market is one of the key factors for any company to release its product and get hold on customers. The designing part should be completed in a short span of time so that verification and fabrication can be completed without missing technology window. Verification of the design takes about 70% of the total turnaround time.

The main aim of this project is to provide efficient methodology for front end process through which designers and verification engineers can complete the front end tasks using different verification flow over their design efficiently with some time saving. Advance automated mechanism provide efficient flow methodology from start to end level of front end design and verification process. A2M provides a common flow for all the front end activity and using a common flow leads to huge time saving, it has all the tool flow steps automated in it and also contains all the modern tool flow that help in cutting down the verification time. It is capable of handling repetitive and time consuming task for complex designs and managing all the design data .

The advance features present and developed in the system are actually the time saving utility that helps to cut down the time required for front end process and provide a super fast debugging help. Setting the design environment for different verification methodology was also an issue for all the projects A2M also solves this issue by providing a common platform. Property checking flow integration in advance flow also helps to cut down the over all verification time. It is a proven fact that using formal for verification reduces design data and verification time.

Bibliography

[1] A. Panda, "Front-End Design Flows for Systems on Chip: An Embedded Tutorial," in IEEE, Bangalore, 2010.

[2] R. A. Rutenbar, "VLSI CAD: Logic to Layout," coursera, February 2015. [Online]. Available: https://www.coursera.org/course/vlsicad.

[3] "Intelpedia," Intel, April 2015. [Online]. Available: https://intelpedia.intel.com.

[4] "Lint (software)," Wikipedia, 3 February 2015,. [Online]. Available: http://en.wikipedia.org/wiki/Lint_(software).

[5] Training material and foils on different FE flows

[6] Larry Wall, Tom Christiansen Jon Orwant, Programming Perl, 3rd Edition, O'Reilly Media, July 2000.

[7] A. B. Mehta, SystemVerilog Assertions and Functional Coverage: Guide to Language, Methodology and Applications, Springer, August 19, 2013.

[8] Erik Seligman, "Bringing Formal Property Verification Methodology to," in DV-Con, 2012

[9] P. Dasgupta, A Roadmap for Formal Property Verification, 2006: Springer Netherlands.

[10] Rajesh Gupta, et al., Panel: Formal Verification: Prove It or Pitch It, Design Automation Conference, June 2003.

[11] Frank Dresig, et al., Assertions Enter the Verification Arena, Chip Design Magazine, December/January 2004.

[12] Property Verification tool user guide.