EFFIECENT FE METHODOLOGY FOR RTL INTEGRATION IN LARGE CPU DESIGN

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

MASTER OF TECHNOLOGY

IN

ELECTRONICS & COMMUNICATION ENGINEERING

(VLSI Design)

By

PRAJAPATI PINKESHKUMAR HARISH

12MECV22

Department of Electronics & Communication Engineering

Institute of Technology, Nirma University

Ahmedabad

May, 2014

EFFIECENT FE METHODOLOGY FOR RTL INTEGRATION IN LARGE CPU DESIGN

Major Project

Submitted in partial fulfillment of the requirements
for the degree of
M.TECH in VLSI

By
PRAJAPATI PINKESHKUMAR HARISH
12MECV22

Guided By
Internal Guide
Dr. Usha Mehta
External Guide
Mr. R, Satish Kumar
Mr. Sandip Rajput

Department of Electronics Communication Engineering
Institute of Technology, Nirma University
Ahmedabad

May, 2014

Declaration

This is to certify that

i) I, Pinkesh Prajapati a student of Master in Technology in VLSI Design, Nirma University, Ahmedabad hereby declare that the project work EFFICIENT FE METHODOLOGY FOR RTL INTEGRATION IN LARGE CPU DESIGN has been independently carried out by me under the guidance of Mr. R, Satish Kumar and Mr. Sandip Rajput, Intel Technology India Private Limited, Bangalore and Prof. Usha Mehta, Program coordinator, Department of VLSI Design, Nirma University, Ahmedabad. This Project has been submitted in the partial fulfillment of the requirements for the award of degree Master of Technology (M.Tech.) in VLSI Design, Nirma University Ahmedabad during the year 2013 - 2014.

ii) I have not submitted this work in full or part to any other University or Institution for the award of any other degree.

- Prajapati Pinkeshkumar Harish
12MECV22

Certificate

This is to certify that the Major Project entitled Efficient FE methodology for RTL integration in large CPU design submitted by Pinkesh Prajapati H (12MECV22), towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my 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 my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Dr. Usha Mehta
Internal Project Guide,
Institute of Technology,
Nirma University, Ahmedabad

Dr. N.M. Devashyaree ,
Program Coordinator
Institute of Technology,
Nirma University, Ahmedabad

Mr. R, Satish Kumar
Mr. Sandip Rajput
External Project Guide
Intel Technology India Pvt. Ltd.,
Bangalore

Prof. P N Tekwani
Head of EE Dept
Institute of Technology,
Nirma University,Ahmedabad

Dr. K Kotecha
Director, IT-NU
Nirma University, Ahmedabad

Date: Place:Ahmedabad

Abstract

RTL (register transfer level) integration is an important part of a VLSI design flow. It basically define the quality of the full chip system. Also in large CPU design, whole design are partitioned in different parts to reduce the complexity of the design and at the end all individual RTL logic are integrated to get full chip design. Thus for good quality and well featured chip we need to have a good quality RTL integration in our design. This can be achieved by having well defined FE (front end) methodology or flow. This report will give different FE flow example for different design or application and will explain in detail how we can modify the flow for getting better and verified RTL integration at the end. It will also demonstrate some example which will explain how proper FE methodology can reduce the initial bugs in the design. At the end will also explain the efficient, simple and much faster method to testing the system/tool functionality.

Acknowledgements

First and foremost, sincere thanks to Mr. R, Satish Kumar, Manager, Intel Technology India Private Limited, Bangalore for assigning me such project and guide me through. I would like to thank my Mentors, 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 Project Co-ordinator, Professor Usha Mehta and Dr. N.M. Devashrayee, 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.

Prajapati Pinkeshkumar Harish
12mecv22

Abbreviation Notation and Nomenclature

RTL ..... Resister Transfer Level
API ..... Application Programming Interface
FE ..... Front End
TTM ..... Time To Market
QA ..... Quality Analysis
FEV ..... Formal Equivalence verification
HDL ..... Hardware Description Language
MUT ..... Module Under Test

Contents

List of Figures

Chapter 1

Introduction

In order to compete in modern era for getting high performance, low power and less area chip, we need to find more efficient methodology and flow starting from start to end level of designing. Also In order to go parallel to Moores law we need to reduce the size of the chip for same number of the transistor in specific period of time and in process we need to increase the performance, reduce the power and cost. And for achieving this goal one of the important step is to have good quality RTL integration. Here RTL integration means integration of small or low level design to form top module and in order to have good quality chip we need to have this RTL integrate to be of high quality.

The good quality RTL integration can be done by having efficient front end methodology and flows. This front end methodology basically contains the flow development for different design as per their requirement. Also by having good flow at the start will reduced the number of the bugs at later stages. FE methodology is basically the different Front end method/flow/solutions/innovations which can results in better quality and enhance RTL integration.

Basic process that need to be follow while defining proper FE flow are as follow

- Clear Specification
- Dividing design in different process step as per the complexity of the design
- Defininf each steps in detail
- Making wrapper system with all the steps included in it.
- Testing each step functionality by running system on proper test cases.
- Making modification in flow if specification doesn't met.
- Testing whole system in design environment and validating the results it produce.

- Implementing the system in design for customer use.

Level at which all FE methodology can be define, can be anywhere, starting from specification to RTL integration step. The level at which I had normal worked is at RTL stage i.e. the stage after all the designing of different module has been done and there is need to have flows for checking the module quality of the design.

Below are few things which I have worked on developing/testing and deploying to different design in Intel.

- Efficient front end design equivalence flow for different designs
- Automating regression test for different Linting tool version
- Interconnections mismatch detection system
- Test script deployment for system feature verifications
- And some supporting system.

Above all work is basically done for enhancing the quality of the module or system for getting better results. Brief description of all the topics is done in chapter 3.

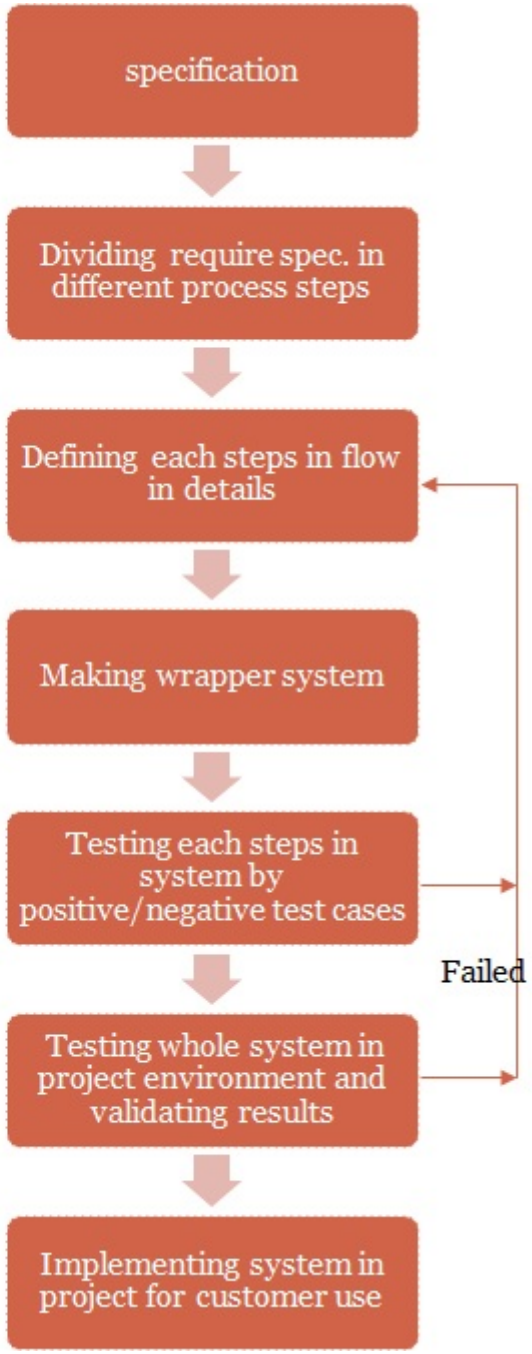

Figure 1.1: Basic Steps to Create Efficient Flow

Chapter 2

Project Details

2.1 Introduction

This chapter will explain briefly the motivation of efficient FE methodology and explains different terms like RTL integrations, different challenges that need to be taken care while integrations and importance of having efficient FE methodology.

2.2 FE Methodology For RTL Integrations

2.2.1 Definitions:-

In simple words it is defined as different methods/flow for good quality RTL integration. This include defining the efficient flow for front end and implementing the flows for same. This flows include the basic step, by which we can make good quality, cost efficient, bug free RTL integration.

The flow step may differs as per the requirement of the design. While making an efficient flow we may need to consider all customer requirement, feature that we required or quality that we may need to maintain throughout our design

Thus, in general to provide stable, good quality, time efficient, bug free design at initial phase and somewhat automating RTL integrations we make some FE methodology which can fulfill our basic requirement.

2.2.2 Challenges

The basic challenges for making and efficient FE methodology of RTL integration are as follow:-

- Design Complexity

- Automating the design flow
- Module quality

Design Complexity: Now a days the design complexity of the whole chip is getting increased, because of many reason like integrating more and more component in single chip, cost of the product, power requirement and less area The system design problem describes the process used to translating the need or requirement for a system into an actual design and this is an NP- hard problem.

Thus it important that we should include some well-defined flows in the design by which we can meet all our requirements and reduce the design complexity. We can even divide the complex area in the flow with some sub flow for further reducing the complexity of the flow and thus defining each of the steps in the flow in more detail.

Automating the design flow: As the time to market of any product in VLSI is very important, if our product in not available in market in specified period of time, then we can lose our pride of our product quiet easily.

Thus its important that our product reached the customer in minimum time. In order to reduce the time consumed while going through all the steps of the flow manually we can automate the whole flow as much as possible and can reduce the flow time. This can be done by using many scripting language. The selection of the scripting language is totally depended on user who is going to automate the system. I had mostly used Perl as a scripting language as it is very strong language if we want to deal with string operation (i.e. extracting the information from result created by different tools). Also by creating whole single system will results in better user usage i.e. other user who dont have the knowledge about all the steps in flow can also run the whole flow with single command. Also Less TTM can also be achieved by inheriting some good features of other design and include same flow with some minuet changes in it.

Module Quality Its an important aspect of any design. If we cannot maintain the quality of the module in our design, whole design is of no use. So in order to maintain the quality of the design we need to monitor the quality of the modules. This can be done by including the some quality check steps in the flow and thus confirming the functionality of each steps in the flow. We can also may need to modify the steps in flow for better quality of the module, we can also may need to include some more steps of quality checking of the module.

Such one step which we had included in the FE RTL integration flow is interconnection mismatch detection system. This basically finds the interconnection mismatch

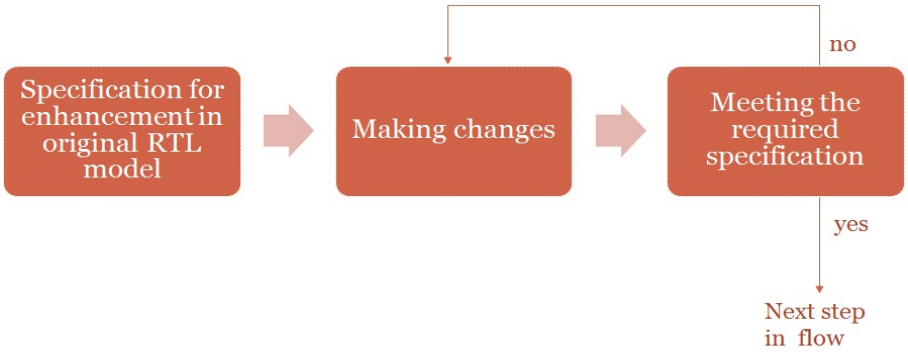

Figure 2.1: Original Design Flow which Required Some Module Quality Checking

between the golden and modified RTL module. By finding such interconnection mismatch we can identify which interconnection are changed or are getting effected by modifying the original RTL module.

Interconnection mismatch detection system:- The concept of the applying different FE methodology can be understand by taking understanding problem statement of interconnection mismatch detection system. This is basically enhancing the feature of existing flow. Suppose you had made some RTL logic and suppose if there are some modification or enhancement are required in the design and because of this you make some changes in it. While requirement of the design is that there should not be any mismatch in interconnecting signals between original RTL design and new RTL design.

This requirement can be achieve by including one intermediate stage in between. This stage will contain the system which can compare the interconnection mismatch between the two modules and can identified the list of the mismatches. By doing so, we can analyze the list of mismatch interconnection between original and new RTL modules. Thus we can increase the quality of the new RTL by removing the bugs at the initial stage only.

Figure2.1 show the original flow, and Figure2.2 show the modified flow with interconnection mismatch detection system step placed in flow. As shown the figure interconnection mismatch system will find the list of interconnection mismatch between original and modified design.

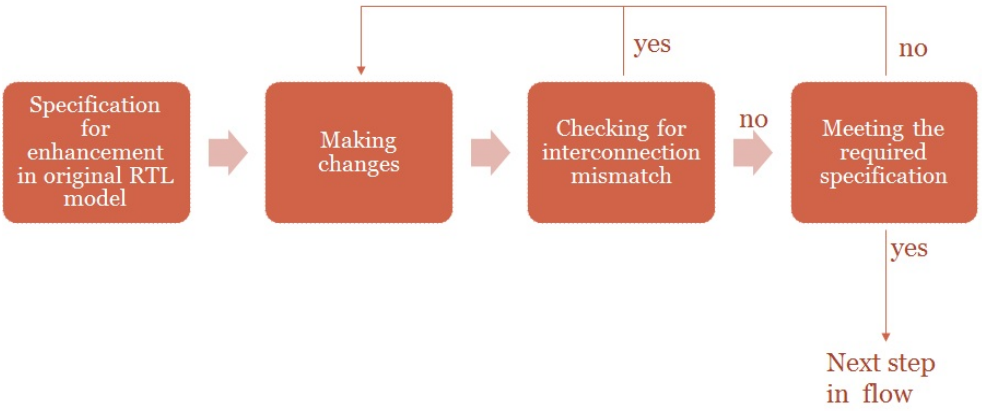

Figure 2.2: Flow With Interconnection Mismatch Detection System

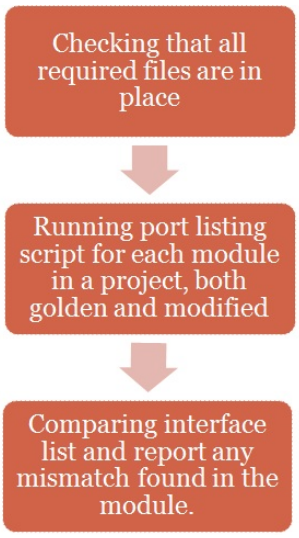

Figure 2.3: Sub Steps of Interconnection Mismatch Detection System

This interconnection mismatch system can be included in any design flow and can be used in any design flow, where it is necessary to find the changes in interface made by the modified RTL module. By including this kind of system in between can increase the module quality and thus can reduce the number of bugs at initial phase of design.

This concept can be shown by block diagram as follow:

This interconnection mismatch detection system can be further be divided in three basic step as follows:-

i) Checking that all required files are in place.

ii) Running port listing script for each module in a design both golden/modified.

iii) Comparing interconnection list and report any mismatch found in the module.

While running interconnection mismatch system, if we get the interconnection mismatch list between some golden and modified module we have to report that to design team and should have some reasoning for having such mismatch. If this mismatch are valid and approved by the design teams, then that interconnection we can ignore and design can processed to next step of the flow. Thus by this mechanism we can at least reduce the bugs related to interconnection mismatch in any design and thus increase the design quality.

Thus in general, we can enhance the quality of the module by putting some intermediate quality checks steps which verify the result produce by preview flow step and thus this will reduce the number of the bugs at initial stage of the design. Also automating the flow steps will results in very less flow time.

Chapter 3

Enhancement Made In Different FE Flow For Increasing Module Quality

3.1 Introduction

This chapter specify the summary all the enhancement done in different FE flow for increasing the module quality of model and will explain in detail each of the enhancement with appropriate example.

The list of the things that we had developed or in progress or had worked on are as follows with some brief point that I had done on it

- Efficient FE design equivalence flow for different design:-
    1. Wrapper systems
    2. Deploying the systems
    3. Testing the systems for both negative/positive conditions.
    4. Implementing whole system in design.
- Automating regression test for different Linting tool version:-
    1. Collecting the steps which are necessary.
    2. Making the wrapper systems.
    3. Running the wrapper systems in different design environment.
    4. Testing and analysing the results.
    5. Enhancing the system as per user requirement.
- Improving waiver mechanism in Linting tool

1. Analyzing the requirement and creating required specification out of it.
2. Dividing the complex flow in to different stages for making it simple
3. Implementing each stages
4. Testing in whole RTL model

- Interconection mismatch detection system:-
  1. Wrapper system, specific to particular design
  2. Testing the system
  3. Implementing system in design.
- Test script development for system feature verifications
- Debugging and enhancement (side work)
- Others tools flow ramping up.

3.2 Efficient FE Design Equivalence Flow For Different Designs

In VLSI design flow, its very important that whenever some modification/enhancement are made in RTL, we need to verify the change/enhancement in RTL first. Thus formal equivalence checking of the modified module with golden module is very important to verify or prove the correctness of designs. This formal equivalence verification can be done by comparing the mathematical module of both golden and modified module. There are many other way by which we can to FEV between two modules.

There are many vendors providing the Formal equivalence verification (FEV) tools for different stage in design. We can use any of it as per our requirement and cost of product. Now in using FEV in any of the stage required some basic steps to follow for quality checking of the module. This steps includes taking proper golden reference module, proper environment setup for FEV tool, proper modified module design files, creating proper configuration files for different feature checking of the module while running FEV and more steps like this.

Thus, in order to have quality FEV checking with minimum bugs at the initial stage we need to create efficient flow of it, with all the required steps. In between this steps we can include verification steps which can verify the results produce by the preview step in flow. By combining this all point we had made an efficient FEV flow. We had made this full flow fully automated.

The basic steps included in this flow are shown in Figure3.1.

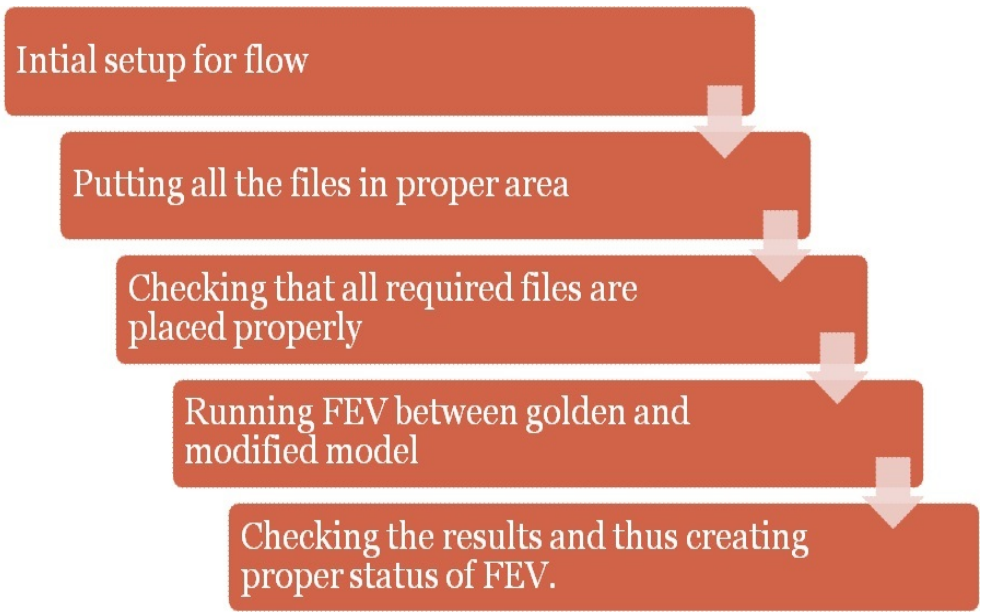

Figure 3.1: Step of Efficient FE FEV Flow

Details of the each steps:- Initial setup for flow: - This step contain all the environment setup of the tools or system to work properly.

Putting all the files in proper areas: - for running FEV tool we required modules files, one golden one and other is modified one and this list of files which are required in the FEV should be placed in proper directory format. This format may differ as design get changed.

Next step to check that whatever the files are being using are placed in proper format as per requirement

Running FEV tool: - After getting all the required files we can run our FEV tool in it, which will does the formal equivalence verification between two specified modules.

Checking results: - after running FEV tool we need to analyze report generated by it and by doing so we can report the type of result produce by FEV tool.

At later stage, in this system/flow there was a need to create some notification features in the case of the any failures. This type of notification where required, as this equivalence flow was used in some top flow were it was very difficult to find any

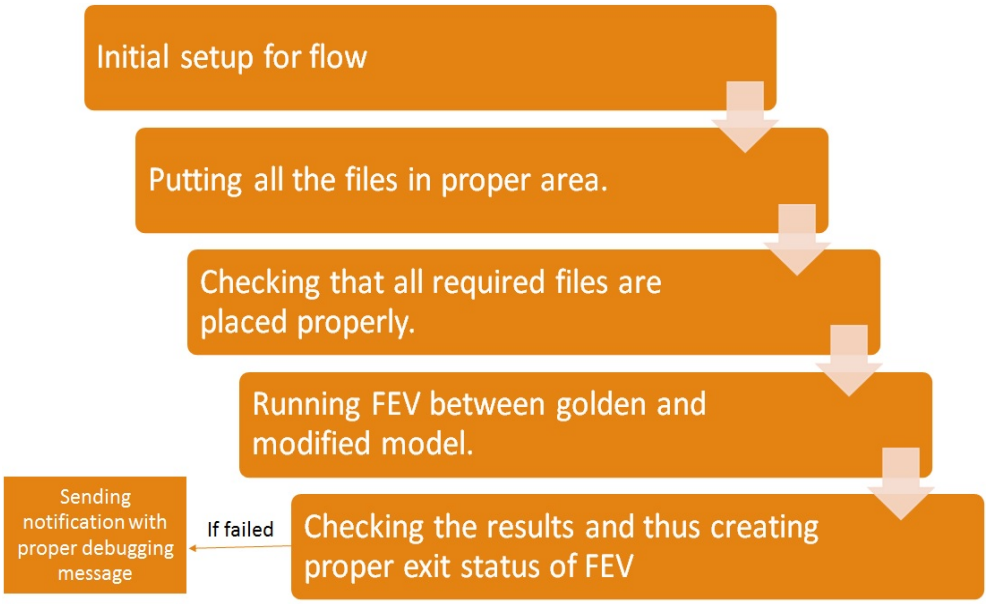

Figure 3.2: Inclusion of Email Notification

failure cause of full run. Thus, if by some means we can include some notification for failure cases with some debugging message then this will help designer to notify and debug the issue in right direction. Thus if equivalence flow can include some notification method for same then it can help user/developer to find out the root cause of the failure.

Notification mechanism with debugging message:- In equivalence flow there are many stage where chances of RTL module getting failed will be present. Some of the failure causes which can occur are as follow.

- System somehow is not able to setup initial setting.
- System is not able to copy the files due to some permission problem
- If some of the required files like some golden RTL module files are not present.
- If system is not able to run FEV tool (may be due to some license error).

There are many chances of getting above mentioned errors while running flow and because of this error, flow will fail and will exit with some exit status. In our notification mechanism we will list down all such kind of the error/failure and will send email to user/developer notifying them that their run had failed with some

errors. Corresponding notification message that can be send for above four failure types are as follow

- Error: - system is not able to set environment variables.
- Error: - Not able to copy file ¡file_name¿ : permission denied
- Error: - Cannot find required file ¡file_name¿
- Error: - Cannot run FEV tool, Please check the proper license file

Above notification mails to user/developer will allow them to debug the issue in the case of any failures. Thus this kind of notification mechanism can reduce the user time while debugging the cause of failures.

Now in order to include this type of feature, we had used some mailing Perl API for sending proper formatted email notification. Small example showing the way we can enable mailing feature is shown in Appendix A. it uses library MIME::LITE for mailing. Please follow the comments in code for understanding mail notification.

3.3 Automating Regression Test For Different Linting Tool Vesion

In general Lint tools that flag suspicious and non- portable usage of language construct in any programming language. It points out the code where it likely to be bugs. In chip design world Lint tools (some time referred as Design Rule Checker) check the cleanness and portability of the HDLs code for various EDA tools. Usually compiler does not show the errors and warnings which detected by lint tools.

Linting tool offers you a simple but fast method to improve the quality of the HDL code. It is intended to find all kinds of language constructs that are formally correct but probably not intended. Examples are signals that are defined but never used; signals that are on the sensitivity list of a process but not used inside the process; etc. You can run the Lint tool on the whole design or on packages, entities or modules separately.

Linting tool library will have all the list of such rule and also one may include its own rule i.e. user defined rules in it. This rule can be classified by different severity types. Some of the example of it are as follow:

- Error
- Warnings
- Fatal

The type of severity that the rule will have will all depend on how critically the rule is defined in a design.

Some of the attribute which define the linting rules are as follow

- Syntax and semantic rules that needs to be followed.
- Non-Synthesizable logic in the design.
- Implicit latches formation.
- Violation in some downstream tools.

Syntax and semantic rules check: For having proper working design logics. It is necessary that design logics are free from syntax and semantic error. For this we can include different kind of syntax and semantic rules in linting tools which can reduce it.

Non-Synthesizable logic in the design: HDL language can be used for making both simulation code and synthesizable code. Thus it is necessary that while making synthesizable code, we dont include any non-synthesizable thing in it, because it will not be able to create logical hardware from it. This can be achieved by including non-synthesizable codes type on linting rule library.

mplicit latches: Sometime while writing HDL code, we might forget to include all if else or case conditions, which can be unintentional in some design cases. Synthesizing tool will generate the latches for all the remaining conditions which are not mention and this can results into undesirable latches in the design. Such kind of latch generation points in the design can be highlighted by linting tool by including the rules related to such conditions.

Violation in some downstream tools: Some linting rules will also depend on the downstream tools which will be run after the completion of synthesizing step. There can be some cases in which violation on such tools can occur due to the bug in design logic. Such kind of the bug can be pointed out by the linting tool by including rules related to such conditions.

In order to improve the quality of the design style and reduce the number of the bugs in the design code, we goes on including the different types of rules in the linting tool system. By adding new rules in the linting tool library and thus enhancing the capability of linting tool to catch the violation in design, can results into new linting tool version.

Now changing the tools version in any design is not that easy. We must test the new version tool in many design and then after validating we can implement it in real design environment. Thus for good quality checking we need some well-defined flow for testing the migration of the linting tools version.

This flow should include running different linting tool version on all test cases of design. Some basic step of such flows are shown in Figure3.3.

Detail of each step:- *Taking proper input commands:-* In this step user need to specify the linting tool version by which the all test case need to be ran on all the given modules.
Initial Setup:- Placing all the required design files in their proper structure and creating proper system environment.

Checking for required files:- Confirming that all the required module file are in place and this can be used by regression tools ahead.

Running all the test cases on each module for given linting tool version:- This test cases are run by a regression tool which will create violation report of each module. The violation report basically is a report which will contain all the rules which are not follow by the module design.

Merging all the violation reports:- In this step all the individual violation reports are merged and formed a single merge report which will contain all the violations of each module.

Comparing the violation reports:- The merged violation report is compared with golden violation report. This golden violation report is the report which is generated by running the older linting tool versions on all the modules.
Extracting the information form violation reports:- In this step many information like same violation and different violation list are made and according to this it will produce the summary of the result.

While changing the linting tool version if we find any difference in the original and new violation report. Then we need to analyze this result and validate the results produce by it or debug the false violation differences.

3.4 Improving Linting Tool Waiver Mechanism:-

Waiver Mechanism:- Linting tool is defined by set of rules that need to be followed by the designer for efficient RTL integration. This set of rules can also be created by

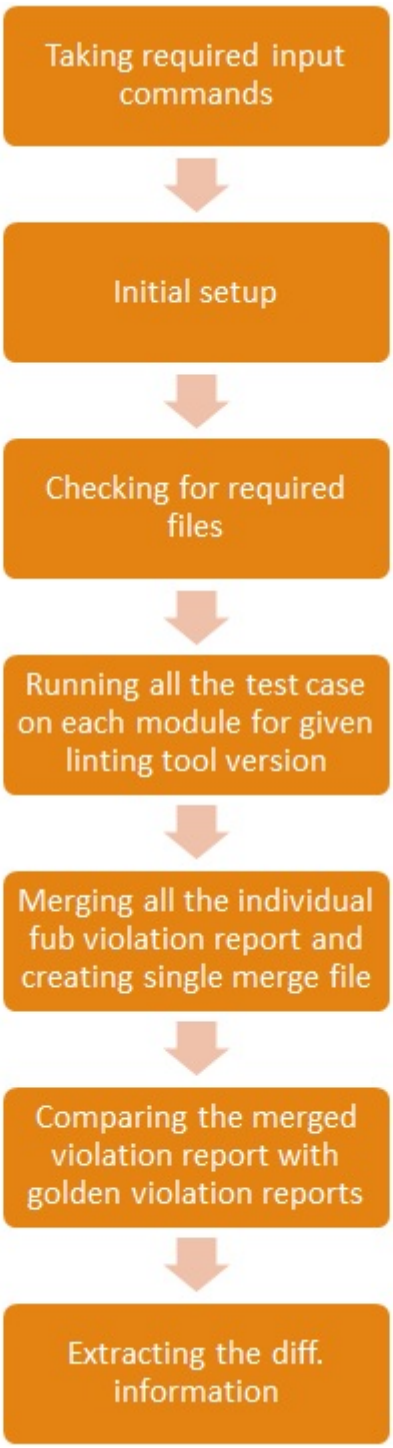

Figure 3.3: Automatic Regression Test Flow

any designer as per his requirement. Linting tool when ran on whole RTL model will generate list of violation that designer had violated in his RTL code. This violation can be validated by the designer and can be waived as per requirement (i.e. remove it temporarily or permanently). The mechanism used to waive this violation are called waiver mechanism.

For understanding this waiver mechanism one need to have some understanding of violations definition (i.e. how violation are defined). Lets take simple example of one violation/rule defined for some Verilog file module.v. it can be defined in many ways one of it are as follow

For Module.v file coded as Line 1: .
Line 2:
.
Line 4: module <> ();
.
Line 12: int [7:0] system_bus;
..

Line 100: Endmodule

Below is the one method by which violation/rule can be defined
Violation_id = 001
Verilog_file = .*/module.v
Line_number = 12
Reason = “invalid ’system_bus’ port type “
Types = “Error”
Waiver = “ “

Thus by providing different field (like mention above) one can define his own violation, which he/she wants to include while running Linting tool. Now in order to waive this violation one create waiver as follow.

Waiver_id = <any id which define this waiver>
Violation_id = 001
Messages = “Invalid ’system_bus’ port type”
Line_number = 12
Type = “Error”

By comparing different field of this waiver with the corresponding field of violation list, we can waive that particular violation (temporary or permanently). If the waiver

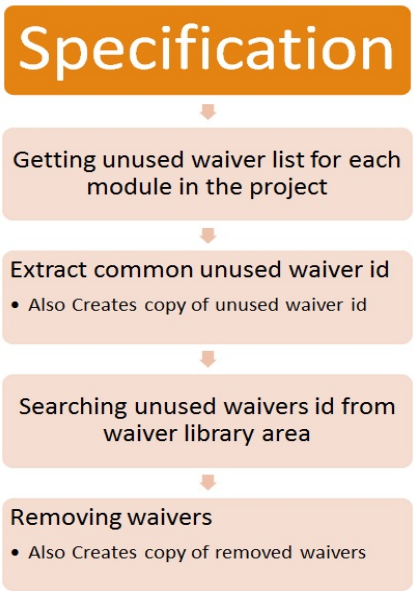

Figure 3.4: Flow For Removing Unused Waiver List

is successfully applied to violation, then "waiver" field in that particular violation will have the value equal to applied waiver_id.

Actual violation list and waiver list can be very huge and complex. Also different operation and system involved in this can be very complex. In any of the project we ran whole linting tool flow and generate the violation report which contain whole list of violation that were found in design. Also if all this violations were waived then all the violation in violation report will have "waiver" field with value equal to applied waiverpar_id. All applied waiver list will be kept at some directory in a project (this can also be called as waiver library).

Waiver list which are created may depend on the designer who wants to waive some violation in it design. Also there is good possibility that two designer may define similar type of waiver and there is also a good possibility of having extra waiver list which are never being used in the project.

While running Linting tool, this unused waiver list may create some unnecessary load in a machine. Thus its better to remove this unused waiver from waiver library. For that we had created a system which can find the unused waiver from waiver library and can remove them. Basically the whole system was divided into three main operation.

Stage 1: This will generate unused and used id list for each individual modules in project
Stage 2: After creating unused/used id list for each module in whole project, we combine them and create common unused/used id list and store them in some file for analysis purpose.
Stage 3: After getting common unused id list we search each id from all waivers file and delete them from waiver library. All deleted waiver are stored in some file for analysis purpose.
System was design in such a way that we can ran system in following modes

- Run only Stage1 and Stage2
- Run only Stage3
- Run whole flow

Figure 8 shows the optimized flow for removing unused waiver list. Each block in the flow was implemented using Perl language and after implementation of each block, had combined all the stages to form whole single system.

3.5 Interconnection Mismatch Detection Flow

This flow included comparing the interface between two modules and listing the interface mismatch between them.
This flow was explained in chapter 2 while explaining the different FE methodology challenges. Brief step that I had done are as follow:

- Making wrapper system (initial dummy system).
- Testing wrapper system on some test cases
- Implementing the system on specific design.

3.6 Debugging And Enhancement

1. Made an enhancement in the existing supporting systems for migrating it to new control systems. Control system: - Its a control system used to control data of the project. It includes transferring, updating or creating repository of project. It helps in handling the project data and its changes/enhancement more efficiently. There are different script or system used in the front end flow which are designed/coded according to control system that it uses. Thus while

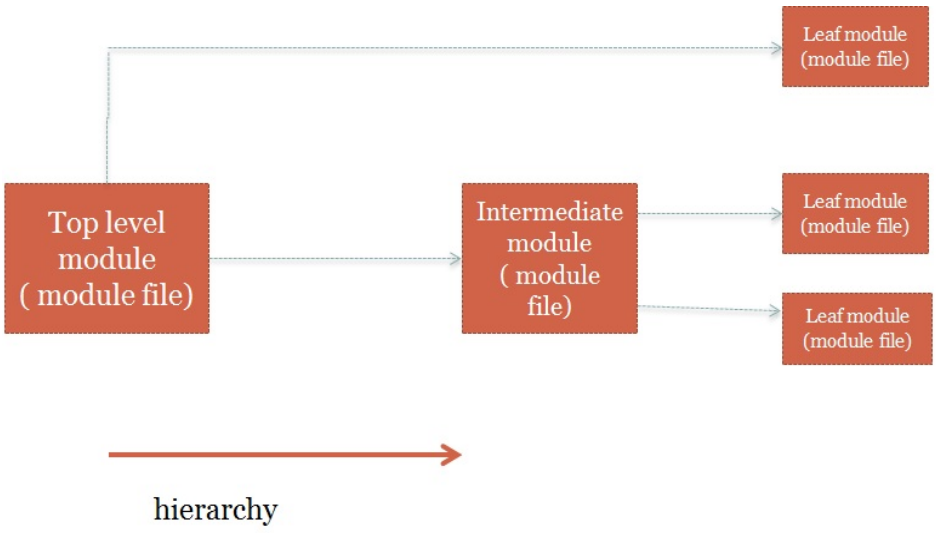

Figure 3.5: Hierarchy Interface Tools Example

shifting to other control system we need to make corresponding changes in this system for making them supportive to new control system

There was similar type of migration of control system in different projects in Intel for increasing data handling techniques. Thus it was necessary to make corresponding changes in different system/script (which uses old control system features) for making them compatible to this new control system.

Mostly the changes that need to be done were as follow.

- Changing all old control system command with corresponding/equivalent command in new control system (this required knowledge of both control system).
- Making directory structure of all data as per new control system requirement.
- Processing the data, while putting into consideration the directory structure of new control structure.

2. Debugged the supporting script used in Linting tool waiver system. We had discussed about waiver mechanism in section 3.4. There was one bug in system which was used to update waivers list. It was originally made for overcoming the waiving issue in waiver mechanism, during the linting tool version migration. During migration there was some list of new/old violation having only

message difference. Thus, if waiver was applied in old violation, then ideally same waiver should also waive the corresponding violation in new violation report. But this was not the case, actually because of message differences, waiver list was not able to waive the corresponding violation in new violation report. This was debugged by providing proper message field to waiver list for applying them in violation list found in newer violation reports.

3. Created documentation of the all the system that I was involved in making and had uploaded in Intel documentation site. Using this documentation any user can use this system more efficiently and can do some first level of debugging in the case of any failures or issues.

3.7 Other Tool Flows Ramping Up

To provide stable, good quality, time efficient and automated hierarchy management in RTL. Take cares connectivity of the different modules. Consider simple example for understanding how this tool works. As shown in the figure 6. Input file to the module will be the module and hierarchy file of the top and intermediate modules and module file of leaf modules

Tools will first takes the module and hierarchy file of top module and will in depth of hierarchy up to leaf module. This is shown in the figure 7. While going to the leaf module it will go on making the connection between the blocks in the same hierarchy. At final it will generate single module file which will contain all the information about the module starting from top module to leaf module.

List of the thing which I had done are as follows

- Made a system which uses the file generated by tool and find the interconnection mismatches detection system.
- Had run the connectivity tools on different design units and analyzed the results
- Made a script for automatic generation of the collateral for tool new version test case.

In future deployment of new version of hierarchy interface tool with enhanced quality is required for different designs.

Chapter 4

Testing Tool Features

4.1 Introduction

This chapter focus on method of testing feature of different Tools and motivation dehind such methodology. It also specify the different advantage and disadvantage of this methodology with appropriate examples.

Each tools are implemented through some languages like Python, Perl, C++ or mixture of any languages. Now a day the complexity of the tools are increasing significantly, as complexity of the data handling, different operation that to be done and inclusion of intelligence in tools are increasing. As design complexities increases or design type changes, for getting efficient model quality, we need to make corresponding or equivalent changes in tool also. Thus while making any changes it is very important that we test all the old/new features of the tools. Also, Its very important that while including any feature in tool, old features of tool should not be effected.

Normally, testing any type of feature in tools is very time consuming process and will also required some manual efforts. Thus, in order to reduce time spend in testing, we need to have some mechanism, which can test features of tool and which is somewhat automated. One such mechanism can be made using Perl Test API (if system/tool is written in Perl language). We had designed similar testing system for testing Front End setup flow system, called Automated Functional Regression System. It was designed using different feature of Perl test API.
Front End setup flow system: - It is a part of FE flow which produce an efficient simulated model. It will setup the flows basic requirement like stages that need to be ran, dependences of all stages/tools, setting up proper tool version and so on. It is written in Perl languages and thus we can make use of Perl Test API for creating test mechanism of it. This will enable us to test the basic feature of the flow and will reduce the time of testing & effort required in testing while doing enhancement in existing flow.

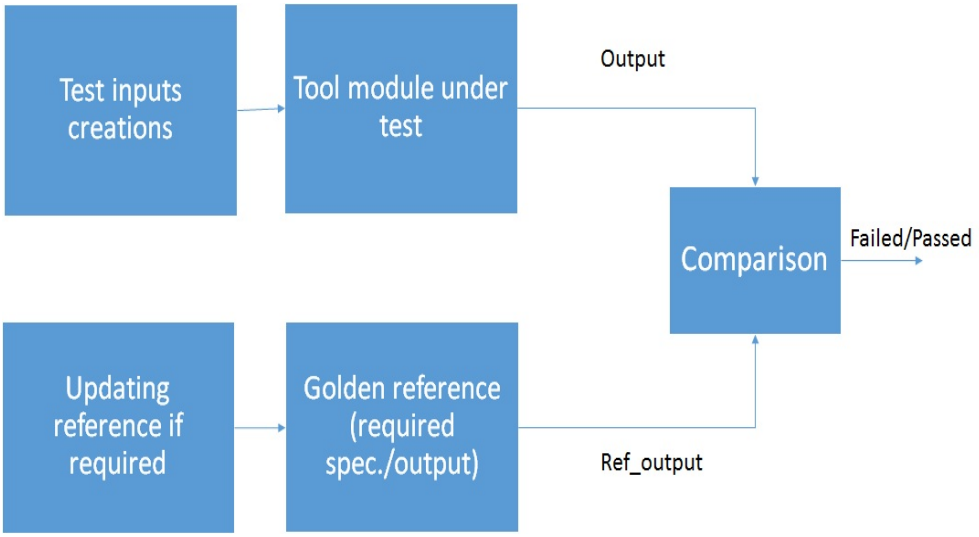

Figure 4.1: Simplified Test Environment

4.2 Automated Funtional Regression System

Motivation:- In order to generate proper simulated model in complex design we need well defined FE flow steps, which can work accurately. This FE flow steps get on changing as per requirement and thus to maintain the better quality of flow, there should be some system for testing basic feature of flows itself.

Example for understanding the Perl test flow.

- *Test inputs creations:* - It will contain mechanism (automated or hard code) for generating test inputs, which need to be applied to module under test for testing its features. This can either be coded in automated form or can be hard coded as per options that are available. If possible, automated test creations is more preferred.

- *Module under test:* - It include the module code whose features need to be tested.

- *reference:* - It will contain the list of the golden output that we expect to get, after applying corresponding input to module under test.

- *reference:* - This block are required only in the case when expected output which are there in golden reference are getting changed after doing some enhancement

in module under test.

- *Comparison:* - It will compare the output of the module under test and Golden reference output for specific input types. This block can be designed by using Perl test APIs like Test::More, Test::Exception or Test::Trap (so on as per requirement).

After doing different study in Perl test, main features that we found are

Features:-

1. Can be used to test any Perl module.
2. Have an access to global variable defined in any of the Perl script used.
3. Can generate code coverage report for finding the code covered by Test script.
4. Can give input to any of the method define in Perl module and can check the return output of it.
5. Can use different kind Perl test API for testing purpose.
6. Can run any specific methods/function/subroutines in the class and check its output.
7. Can create any number of test case for given Perl module.

Using above basic feature of Perl test we had made an automated functional Regression System for testing basic features of frontend setup flow (mainly main machine of it). Appendix B contain some list of Perl test API which are important, its preferred to look at that first.

In order to explain in detail I had made very small and simple example. Please refer the Appendix A for getting detail of each codes.
List of files which are required: -

1. MUT.pm (module under test)
2. Dummy.pl
3. Test.t
4. TestCreation.pm

Brief Descriptions of each file:-

MUT (module under test):- Its a Perl module which is under test, basic feature that it contains are as follow.

- Checking Input Hash syntax and semantics.
- Checking Input Hash keys and fields.

Dummy.pl:- It is a supporting script which is used to provide Flow specification to MUT. (Using this only MUT check the syntax and semantics of it.)

Test.t:- Its a test script for testing MUT using different Perl test API features.

TestCreation.pm:- It will create input test vector. It will generate the input test vector as per the specification defined in Dummy.pl. Thus its in automated form.

General Step to run Perl test:-

1. Source <setting of variables required by MUT for running it smoothly> (if required then).
2. Perl/prove/cover should be pointing to latest version (preferred is more than 5.11.1).
    - Alias perl <path to latest version>
    - Alias prove <path to latest version>
    - Alias cover <path to latest version>
3. Prove t/ :- this will run all the *.t file in the directory t/

Equivalent step that need to follow in for example mentioned in Appendix A are..

1. $ Cd <working directory>
2. Copy MUT.pm and Dummy.pl file at working directory area.
3. $ mkdir t/
4. Copy Test.t and TestCreation.pm file in t/ directory.
5. $ alias prove <path to latest version> (5.14.1 is preferred)
6. $ prove v t/

Note: - line starting with $ indicates that its shell command that need to be printed.

Tips/tricks:-

- You can generate coverage report of the all test case by including include Devel::cover in test script.

    1. In this case previous prove statement will also generate coverage report with it.
    2. Use cover command for converting the cover reports in html format.
    3. In order to open coverage report type
        - Firefox ./cover_db/coverage.html

- Use v switch with prove for detailed of each test while running. (i.e. prove v t/).

Limitation: -

i) If the complexity of the Perl module is large (complexity here is defined as depth of modules called by it i.e. its hierarchy depth.), then its become difficult to prepare test mechanism in it.

ii) If some global variable is defined in the Perl module, then after the completion of one run, I should reset the value of the variable before applying second input to module. This is because of the fact that global variable value in second run will remain as it is.

iii) For creating the Perl test system, one should either be having all list of inputs & corresponding output or should have good understanding of the MUT for making proper inputs which can test Perl module more efficiently.

Chapter 5

Conclusion

- For efficient RTL integration we need to have proper front end flow or methodology with all well-defined steps in it.
- We can enhance the quality of the module by putting some intermediate quality check steps which verify the result produce by preview flow step and thus this will reduce the number of the bugs at initial stage of the design. Also automating the flow steps will results in very less flow time.
- Also we can reuse the some well-define flow steps in different design environment with some modification in it.
- We can also test our system flow by using much faster, simple and efficient method. i.e. By using Perl test module APIs.

Appendices

.1 Appendix A

.1.1 Automatic Functional Regression Test Flow(example for explanation of Perl test system)

A.1 MUT.pl(module under test)

```
#!/usr/pinkesh/bin/perl -w

package MUT;

use strict;
use warnings;
use Data::Dumper;

require (Dummy.pl);
#   1.  Should check whether this was "KEY1" field was same as define in global
# 2. Should check whether ref. of all the keys are are proper types.. as in defi
# Input 1;
# $flow_name = "FE_flow1";
# $spec = {
# "Stage" => [
# stage1, stage2 ],
# "subflow" => [],
# "pre_stage" => {},
# },
# Input2;
# $flow_name = "FE_flow2"
# $spec = {
# "pre_stage" => {},
# "Stage" =>  [],
# "subflow" => [],
# "post_stage" => sub{"DUMMY"}
# }
#



##############################################3
# syntax checking.. ..
##############################################

sub syntac_check
```

```
{
my $flow_name = shift;
my $spec    = shift;
my %global_hash = %Spec_hash;
print Dumper(%global_hash);
checking_data();
# checking the type of the value that will exists in input hash
foreach my $key (keys %{$spec})
{
my $type = ref($spec{$key});
unless(exists $global_hash{$type})
{
Print_Error("Invalid type of the spec for $key");
}
}
}

##########################################
# Sub will check for some invalid data
##########################################

sub checking_data
{
unless(exists $global_hash->{$flow_name})
{
Print_Error("Please enter valid flow_name");
}
foreach my $key (keys %{$spec})
{
unless(exists $global_hash->{$flow_name}->{$key})
{
Print_Error("spec '$key' doesn't exists in the global spec variable");
}
}
}
sub Print_Error
{
my $msg = shift;
print("ERROR : $msg \n");
exit 1;
}

1;
```

A.2 Dummp.pl file:- used to get the some flow specification (This is very simple example for explanation only, actual flow specification were very complex)

```
#!/usr/intel/bin/perl -w

use strict;
use warnings;

our %Spec_hash;  // Note variable spec_hash is defined as global variable.

############################
# We are making simple Two flow spec, defining the type of the value they can ha
# Note:- this is an very simple example for explaining the use of perl test
############################################################

%Spec_hash = {
"FE_flow1"
=>{ "Stage" =>  { 'ARRAY' => "it should contain all type of stages names",
  'Help'  => "Stage name"},
"Subflow" => { 'ARRAY' => "it will contain all type of stage value"},
"pre_stage" => { 'Hash'  => "list the pre_stage and its corresponding value"}
},
"FE_flow2" => { "Stage" =>  { 'ARRAY' => "it should contain all type of stages n
  'Help'  => "Stage name"},
"Subflow" => { 'ARRAY' => "it will contain all type of stage value"},
"pre_stage" => { 'HASH'  => "list the pre_stage and its corresponding value"},
"post_stage" => { 'CODE' => "give code/subrouting which need to execute" },
},

};
```

A.3 TestCreation.pl

```
#!/usr/pinkesh/bin/perl5.14.1 -w

package TestCreation;

use strict;
```

```
use warnings;
require(Dummy.pl)

sub extracting
{
my $flow_name =  shift;
my $Input_test;
my $Flow_spec = \%Dummy::Spec_hash;  # check it..
($Input_test->{Positive},$Input_test->{Negative}) = create_input();
return($Input_test);
}

sub create_input
{
my $positive;
my $negative;
foreach my $key (keys %{$Flow_spec->{$flow_name}})
{
my @types;
foreach my $inner_key (keys %{$Flow_spec->{$flow_name}->{$key}})
{
push @types, $inner_key;
}
if(grep{/ARRAY/} @types)
{
$positive->{$key} = [];
$negative->{hash}->{$key} = {};
$negative->{scalar}->{$key} = "";
}
elsif(grep{/SCALAR/} @types)
{
$positive->{$key} = "";
$negative->{array}->{$key} = [];
$nagative->{hash}->{$key} = {};
}
elsif(grep{/HASH/} @types)
{
$positive->{$key} = {};
$negative->{array}->{$key} = [];
$negative->{scalar}->{$key} = "";
}

}
```

```
return($positive, $negative);
}

1;
```

A.4 Test.t:

```
#!/usr/pinkesh/bin/perl -w

use strict;
use warnings;

use FindBin qw($RealBin);
push @INC, $RealBin;
use lib "$RealBin";

use Test::More "no_plan";
use Test::Trap;

use_ok('MUT');   # This will check whether MUT module is able to load or not if
can_ok('MUT',qw(syntex_check)); # This will check whether MUT module has any "sy

##############################################################################
# Call the test_creation.pm  and gathering all types of inputs.. pos/neg.. :-)
##############################################################################
use TestCreation;
my $flow_name = "FE_flow1"
my $Test_Input = &TestCreation::extracting($flow_name);
my $positive = $Test_Input->{Positive};

##############################################################################
# running the MUT for positive type of inputs. i.e it is expected to pass in thi
##############################################################################

foreach my $key (keys %{$positive})
{
    $testname = "Testing for $key positive type of inputs";
    trap{&MUT::syntex_check($flow_name,$positive->{$key})};
    $debug = "MUT was expected to pass for this condition, but it is exiting wit
    pos_status($trap,$testname,$debug);
}
##############################################################################
# running the MUT for negative type of inputs. i.e. it is expected to fail in th
```

```
##############################################################################
foreach my $key (keys %{$Test_Input->{Negative}})
{
my $negative = $Test_Input->{Negative}->{$key};
foreach my $inner_key (keys %{$negative})
{
$testname = "Testing for $key negative type of inputs";
trap{&MUT::syntex_check($flow_name,$negative->{$inner_key})};
$debug = "MUT was expected to fail for this condition, but it is passing negativ
neg_status($trap,$testname,$debug);
}
}
##############################################################################
# for positive type of the inputs it will check proper exit status of MUT and in
# whether test had passed or not
##############################################################################
sub pos_status
{
    my $trap = shift;
    my $testname = shift;
    my $debug = shift;

    if(!(exists $trap->{exit}) && !(exists $trap->{die}) && (exists $trap->{retu
    {
        pass("$testname");
    }
    elsif(((exists $trap->{exit}) && !(exists $trap->{die}) && !(exists $trap->{
    {
        pass("$testname");
    }
    else
    {

        fail("$test_name");
        diag("$debugmsg");
    }

}
##############################################################################
# for negative type of input it will check for proper exit status of MUT and ind
#  whether test has pass/failed with some debugging msg if required
##############################################################################
sub neg_status
```

```
{
    my $trap = shift;
    my $testname = shift;
    my $debug = shift;

    if((exists $trap->{die}) && !(exists $trap->{return}) && !(exists $trap->{ex
    {
        pass("$testname");
    }
    elsif ((!(exists $trap->{die}) && !(exists $trap->{return}) && (exists $trap
    {
        pass("$testname");
    }
    else
    {
        fail("$test_name");
        diag("$debug_msg");
}
}
```

A.5 Notification Method Example

```
#!/usr/pinkesh/bin/perl5.14.1 -wait

use strict;
use warnings;

use MIME::Lite;

#######################################.....
#.....
#... Equvilance flow code and librarys
#...
# In the case of any failure we call the suboutine sending and exit the flow
# For example:- sending(<failure_message>,<email_address_of user>,<email_address
#############################################################

sub sending
{
    my $failure_message = shift;
    my $to = shift;
    my $cc = shift;
    my $project_type;
```

```
    my $cmd = "equivalence_flow.pl -run_full_flow ";

    my $email;
    my $failure_cause;
    my $date = `date`; chomp $date;

#########################################################################
# Main Body part of the email. Its in HTML format. This will enable proper forma
formatting of messages
#########################################################################
$email = <<MSG;
<body style="font-family: Arial; font-size: 11pt;">
<p><font style="font-size: 12pt;"></font><br/>
$date</p>
<p>Project:- $project_type<br/>
Command:- $cmd</p>
<p>Your run has equivalence_flow Failures.\n</p>
<p>Failed due to: $failure_messages   </p>
<p>Please verify the logs <br/>
Log File:- <b>$opts->{'logfile'} </b></p>  # this will be the path the logfile o
<p> For more information on equivalence_flow Please type the command:- script/eq
<p>Note:- In order to debug please read failure message clearfull and in the cas

MSG
# creating object of MIME::LITE and setting html file types
# setting To, Cc, Subject
    my $message = MIME::Lite->new(
        To      =>  $to,
        Cc      =>  $cc,
        Subject =>  "Notification for equivalence_flow failure",
        Type    =>  'text/html',
        Data    =>  $email,
    );
# call the send() method and thus will send the email according to required data
# which are set.
    $message->send();
    &$pr("Sending the Notification mail");
}
```

.2 Appendix B

.2.1 List of the usefull Perl APIs

use Getopts::Long :- It is used to processes switches given with Perl file. It parses the command line from @ARGV, recognizing and removing specified options and their possible values.
use Data::Dumper :- stringified Perl data structures, suitable for both printing and eval, the complex data structures.
use XML::Simple :- used for reading/writing XML files. Using different function in it we can read/write XML file quick easily.
use Cwd :- used to get pathname of current working directory
use lib :- it is typically used to add extra directories to perls search path so that later use or require statements will find modules which are not located on perls default search path.
And many more as per requirement try to search in `www.cpan.org`

.2.2 Useful Perl Test API

- Test::More:- it include basic Perl test function like ok(), use_ok(), can_ok() etc.
- Test::Trap: - Used typically for traping the exit, return, stdout, stderr, die status.
- Test::Exception: - this module provides a few convenience methods for testing exception based code.
- Test::Group: - use for grouping the list of tests
- And many more as per requirement

.2.3 Useful Perl Test Function Used in Project

- ok()
- use_ok()
- can_ok()
- trap
- pass()
- fail()
- is()

- SKIP block
- TODO block

.2.4 Some Trick/Tips

1. Try to use hash more in the case if you want to access the some particular field type many time from huge file. This will reduce the run time of your system.
2. Try to make your code some generic and independent (i.e all type of required operation should be handed by single command line only i.e. command line of system itself).

References

[1] *Intelpedia*;

[2] `www.wikipedia.org`

[3] *Training Material and Foils on Different FE flow*, Intel doc.

[4] `http://perldoc.perl.org/` :- for understanding Perl, Perl OOP Concepts

[5] `www.cpan.org` :- for getting useful Perl module API and understanding Perl Test

[6] `http://stackoverflow.com/` - FAQ in Perl

[7] *Larry Wall, Tom Christiansen & Jon Orwant, "Programming Perl".*