Enable high quality and extended debug mechanism in next generation design validation framework

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

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

By

Kruti Vadhavaniya
(15MECE28)

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

May 2017

Enable high quality and extended debug mechanism in next generation design validation framework

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

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

By

Kruti Vadhavaniya
(15MECE28)

Under the guidance of

External Project Guide:

Mr. Sandip Rajput
Intel Technology Pvt. Ltd.
Software component engineer
Banglore.

Internal Project Guide:

Prof. Akash Mecwan
Assistant Professor
EC Department, Institute of Technology,
Nirma University, Ahmedabad.

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

May 2017

Declaration

This is to certify that

a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems 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.

- Kruti Vadhavaniya

15MECE28

Disclaimer

"The content of this thesis does not represent the technology, opinions, beliefs, or positions of Intel Technology Private Limited, its employees, vendors, customers, or associates."

Certificate

This is to certify that the Major Project entitled **"Enable high quality and extended debug mechanism in next generation design validation framework"** submitted by **Kruti Vadhavaniya (15MECE28)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmadabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge, haven't been submitted to any other university or institution for the award of any degree or diploma.

Date:18th May 2017 Place: Ahmedabad

Prof.Akash Mecwan
Internal Guide

Program Coordinator

Dr. D.K.Kothari
Head of EC Dept.

Dr. Alka Mahajan
Director, IT

Certificate

This is to certify that the Major Project entitled **"Enable high quality and extended debug mechanism in next generation design validation framework"** submitted by **Kruti Vadhavaniya (15MECE28)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.

Mr. Sandip Rajput
Software Component Engineer
Intel Technology Private Limited
Bangalore

Acknowledgements

I would like to express my gratitude and sincere thanks to **Dr.D. K. Kothari**, Head of Electronics and Communication Engineering Department, and **Dr.N.P.Gajjar**, PG Coordinator of M.Tech Embedded Systems program for allowing me to undertake this thesis work and for his guidelines during the review process.

I take this opportunity to express my profound gratitude and deep regards to **Prof.Aakash Mecwan**, guide of my major project for his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I would take this opportunity to express a deep sense of gratitude to my Project Manager **Mr.Sushant Madan** (Intel) for his cordial support and for providing valuable information regarding the project and guidance, which helped me in completing this task through various stages.I would also thank **Mr. Sandip Rajput** who is the mentor of my project, **Miss Poornima Khullar**, **Ms. Ruttika Jaju** for always giving good suggestions and solving my doubts to complete my project in a better way.

Lastly, I thank almighty, my parents and friends for their constant encouragement without which this assignment would not be possible.

- Kruti Vadhavaniya

15MECE28

Abstract

In this era, designs become more complex which can not be handled by simple design tool. Main concern is design time to market that can be minimized by advanced automated tools must be needed. Therefore the use of design validation framework provides a higher degree of design confidence and reduces efforts in a re-spin process that result in less time to market. But there are some flaws like not proper qualification steps, complex debugging, high maintenance cost and compatibility with technology. All problems can be solved by next generation design validation framework. Compilation time is reduced, so overall design time to market. The purpose of this project is to identify quality gaps and overcome it with new features and improvising existing features.

Thesis work is divided into two parts, one part explains the improvement in debugging approach while another part has detailed run time analysis to improve the overall performance of the flow. For debugging, availability of information is required. Dependency analyzer tool helps to verify functionality of the flow and debug, generated data. To provide information of flow directly to user several help commands are introduced. Enabling standard debug support make flow efficient and more useful. Performance improvement depends on many factors where run time analysis is one of the critical factors for performance. Run time analysis is done on flow and several methodologies are discussed to reduce run time. Results achieved as 56% of improvement. Testing is one the important factor of framework. Different types of tests are there to check correctness and working of the flow. The framework should efficient, for that run time analysis is done for the test framework. Many unnecessary and inefficient test cases are removed with the suggestion of several methods of improvement. Hence, total design time to market can be reduced.

Contents

List of Tables

List of Figures

Abbreviation Notation and Nomenclature

DUT ..... Design Under Test
EDA ..... Electronic Design Automation
PERL ..... Practical Extraction and Report Language
HDL ..... Hardware Description Language
SoC ..... System on Chip
RTL ..... Resister Transfer Logic

Chapter 1

Introduction

1.1 Motivation

VLSI circuits are dominant in today's world. Everywhere there is a technology with tiny chips in all things. How much accurate work is done on that tiny chips? Design flow shows all steps which are necessary to build these tiny chips.There are the steps from design specification to fabrication. Many important design and verification is including in this. First for any chip specification is required and after that design according to specification is done. After complete design layout is planned and then placement of components are done. Between all these processes testing is continuous process. At each step testing is done by different methods so that rather than big fault at end small errors can be rectified at early stages.Importance of this flow is high as each and every chip is designed by same flow with different methods. Here so many methods are used within flow. Every step need very specific data to process further like design need modules and component type to design and placement needs all different block's information to place beside each other. These all data are not easily gathered specially when design is varying. Some environment is needed for that. Also this flow can not done manually. Automation is the key element for the ease of design and also for time perspective.

Lots of automation tools are there for different stage who automate designing and testing. At every stage different tools are needed. Tools also require some sort of data to process. Raw data can be taken by tool in the first step and can be given further. There is always some wrapper required for these tools which can provide organized data to the tool. This can be seen as framework for the tool. There is a design validation framework which do automation for the tools. Some of the lagging in this framework is removed in next generation design validation framework but efficiency and performance is not up to mark. Flow of this next generation validation framework is very much important to learn and understand. Performance is needed to measure for finding loop holes so that framework can be made more powerful for all the designs and all the tools.Areas which give massive improvement for delta change should be explored.

1.2 Problem Statement

In last few decades electronic industries are growing more and more. Reason behind this growth is due to the quick advancement in technologies of integration, large scale designs which we can include in VLSI designs. Pick up any sector and you will find application of circuits like high performance computing, communication, processors, electronics application etc. where need is rising very fast. For mentioned applications and in all other applications with high performance high computational power is required. With requirement more and more complex functions all these functions should be integrated in small system. In all these designs key factors are power , space, reliability and cost effectiveness.Every design have different requirement so flow for designing them is also different. Design flow includes all steps from specification to fabrication. Lots of data and computations required during this. With small design it is very simple but for bigger design number of components are increasing so for that manual designing is not possible. Automation is the key factor for this

design flow. There are different needs for every design and methods are also varying with designs. Flow of automation is done by automation tool. Tools are required for the designing of tiny chips automatically. There are lots of tools available for different stage of the flow some are for designing while some are for testing.

For any of the design to process through automation tool there are certain steps to follow. Automation tools require specific data format like for specification it is in proper format and if something to design then all supported files and libraries need to be loaded before processing. This is different for different design and varying according to complexity. So these much amount of data must be handled somehow and processed accordingly. For any of the tools there is a wrapper required for the process. All automation tools of different category required the same.To provide data in proper format to all these tools are very important.

There is a framework needed which works as wrapper for these tools. Design validation framework is one of the framework which provides organized data to all the tools. Different tools need different type of data. This framework have very well defined flow. In this flow input data or request is come by command line then this is processed by flow manager which is the heart of the flow. Configuration engine pre-processes data and also generate flow hierarchy. Resource allocator allocate all required resource and provide to operation engine where all operations are done like code-generation, compilation, static check etc. This framework is extended by caching interface mechanism where previous successful run is saved so that saving of time can be done in case of very little or no improvement.

Design validation framework needs some improvement specially for parallel processing which is improved in next generation design validation framework. Still there are several factors where this framework is lagging. Major concern is quality of framework and debug required. Testes designed for different scripts for different

step are also not enough in quality where several factors are needed to add for improvement of features. Performance gaps should be minimized. There is one flow for the design which needs complex data and several amount of time but this is not feasible where only initial staged data is required by the user so user friendly commands and data need to be added in the framework. Overall problem with the framework is in performance which includes high quality and debug mechanism.

1.3 Overview

Integrated circuits are every where in the world in all the things we see. These circuits are designed by different electronic components like transistors, capacitors, resistors, switches and all interconnection of these components.This is designed on small piece of silicon which is called chip. Designing is very important as many things should be taken care of like isolation, connectivity, power consumption etc. There is specific flow for designing integrated circuits. First basic step in the flow is specification of the design.For any design there is market survey done for the technology used. This one is the most important as from specification rest of the flow is designed and circuits are made. For this step some times feedback is also taken for better understanding of specification. After all these information final technical data of specification is prepared.

Second step is to prepare architecture from the specification. Here main work of the flow is started. Architecture is decided by specifications and technology. When architecture is prepared from the specification then it will be implement and tested in the next phase. Prepared architecture now needs to be coded and for that next step is RTL coding. This deign is done at gate level as from acronym register transfer level where coding is done in hardware description language. Code represents deign described by architecture by different components where different blocks are coded as per architecture. Prepared design should be verified correctly. In this step design

is verified logically with timing errors. Different methods and tools are used for this purpose.

Synthesis is one of the key step in this flow. There are two ways one is logic synthesis and second is physical synthesis. In the logical synthesis behavior of the circuit is tested. For that gate level testing is done that behavior of the circuit is same after design which can be done by the hardware tools like FPGA or CPLD boards. In physical synthesis design is tested at physical level. After synthesis final design which is tested is given to manufacturer. Manufacture then perform wafer processing , chip packaging, testing and sample design delivery. Once this sample design is passed in all factors then given it to mass production. For the design of integrated circuit according to flow we need lots of automation tool for different processing Digital design flow regardless of technology is a fully automated process.

Mainly tools can be categorize as design capture tools, simulation and verification tools, layout tools and synthesis and optimization tools. Design entry tools are used for design description. Here, design is captured first and then prepare it for simulation. Type of design tool extracted from the design requirement. In the verification tool it confirms that functionality of the model should be same as design definition by any of the method for formal verification. There is also one more tool which is known as timing simulation tools. Functional simulation tool is used to verify the logical or functional behavior of the design and timing simulation tool is used to verify timing for multiple stages. Circuit delays can be measured actually by this type of verification for the design so it shows actual length of design and sometimes known as back annotation simulation also. Layout tools are generally used by ASIC designers who design them to convert from logical to physical design. Floor planning is working in conjunction with physical design where several levels of cell are design and implemented. Last category of tools include synthesis tools where operations like abstraction of functionality like HDL into physical realization, routing, net list

generation etc. From that designers design gate level design considering parameters like speed, area or power.

EDA tools are used for automation of designing integrated circuits but flexibility is sometimes not up to the mark and also need is changing for every tool. There is a framework needed for those tools to process further.Every tool require different set of input like design tool need specification data while physical design tools needed process data in form of blocks to place and route them. For these different tools design validation framework is needed which organize data in proper format.Library dependencies are very important to maintain for the flow. This framework have several blocks for processing like configuration block, flow manager , operation engine etc. From the command line user can invoke any EDA tools so after that command framework starts processing. After command given from command line specific tool should be invoked but for that first flow manager of the framework is activated.

Flow manager gives data to configuration engine where pre-processing of the data is done from different category. Flow handler is used to handle the flow of process.Operation engine is the onw where all operations like code generation, compilation or static check is done.All data are scheduled by scheduler where data flow is generated. But for the complex design data processing is also increasing. For that parallel processing of the data can help to speed up the processing. Tree structure is developed to process data parallel. Still there are some loop holes present in the framework. Performance is the major factor for any of the framework which includes quality of the processing. Moreover flow of the framework is as long as the complexity of the design so every time for the small result one should wait till end. This can be improved by adding features for user interface where they can direct get required information without delay.So all performance gaps are required to fill. High quality can be bring by continuous adding features and improve existing one.

1.4 Project Flow

The time-line of project work from the start of the project is shown as below.

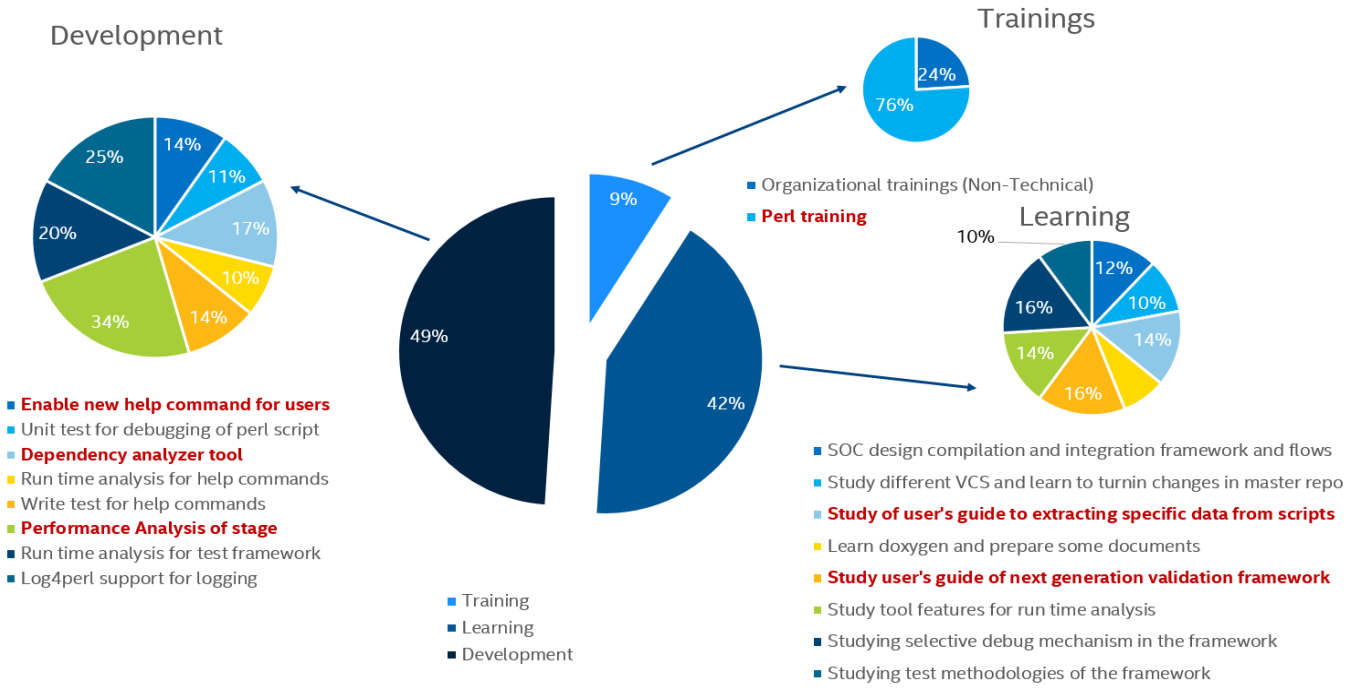


Figure 1.1: Work time-line

1.5 Thesis Organization

The rest of the thesis organized as follows.

Chapter 2 describes literature survey done for the project. It includes VLSI design flow, their models, flow for design as front end and back end. Legacy flow are explained for IP validation.

Chapter 3 deals with the brief introduction of next generation validation framework. This describes complete functionality and flow of operation for design data.

Chapter 4 deals with explanation of extended debug mechanism of the framework. Here need of debugging and importance of information extraction for user is

explained. By extracting data from the data base at early stage time to response is reduced for the user and fault can be detected at early stage.

Chapter 5 deals with the dependency analyzer tool for the framework. This is the tool developed for enhancing debug mechanism and make system more user friendly. Overall performance of system is improved by this tool.

Chapter 6 gives performance analysis. Importance of performance and introduction of perl profiling tool is given here. Study is done on one of the feature of framework and captured improved results in profiling results.Run time analysis is done at stage level and methods for improvement are suggested with results.

Chapter 7 explains testing mechanism in perl. Several modules are described with their functionality which can be added into scripts of framework to improve the performance.

Chapter 8 concludes this report regarding high quality and extended debug mechanism in next generation validation framework.

Chapter 2

Literature Survey

In this chapter brief introduction of the VLSI deign flow is given. More focus is on front end design flow. VLSI design flow is the core need for designing any of the integrated circuit. There are two parts of the flow one is front end and other is back end. Here we focus on front end VLSI design flow. Further different between structural and behavioral modeling is defined. Then two step compilation is explained Legacy and integration flow is also very important. Advanced streamlined system is the solution for both the flow which takes the advantage of both the flow.

2.1 VLSI design flow

Flow for designing VLSI integrated circuits start with formal specification which is then continued by series of steps to produce chip. Following all different steps are elaborated. [3]

- **System Specification**
  - System specification is the core part of any designing. This is the high level representation of the system. Parameters which focused during this phase are performance, quality, dimensions, efficiency, functionality etc. Design technologies and fabrication methods are also considered for this.

- Not only technical but market requirement and economic viability is also very important for design. At the end of design it meet some of the standards of size, power, efficiency, speed,functionality etc. for the system.

- **Architectural Design**

  - Raw architecture at basic level is designed at this step. For designing architecture first decision is based on RISC or CISC for instructions whether it's reduced or complex instruction set, number of ALUs and floating point units or size anf pipelines for the structure.
  - Micro-Architectural Specification is the result of architectural design. But this is only textual description from which designers can predict size, performance and power factors based on this description.

- **Functional or behavioral Design**

  - As described from the name here main functionality of the system is extracted. Estimation of power, area and other parameters are done with interconnect requirements.
  - Without implementation of actual specification behavioral aspects are considered. Like if system need some addition function that only addition function is mentioned but by which method is not bothered. Number of methods and blocks can be used for addition function. We can consider it like a black box where only functionality mentioned in terms of input, output and timing for different unit without internal structure.
  - Ultimately output of this stage is diagram which shows timing and relationship between different units. Benefit from this process is to improve overall design and complexity reduction of different phases of the design which allow better debugging of full system. Mostly this is done manually.

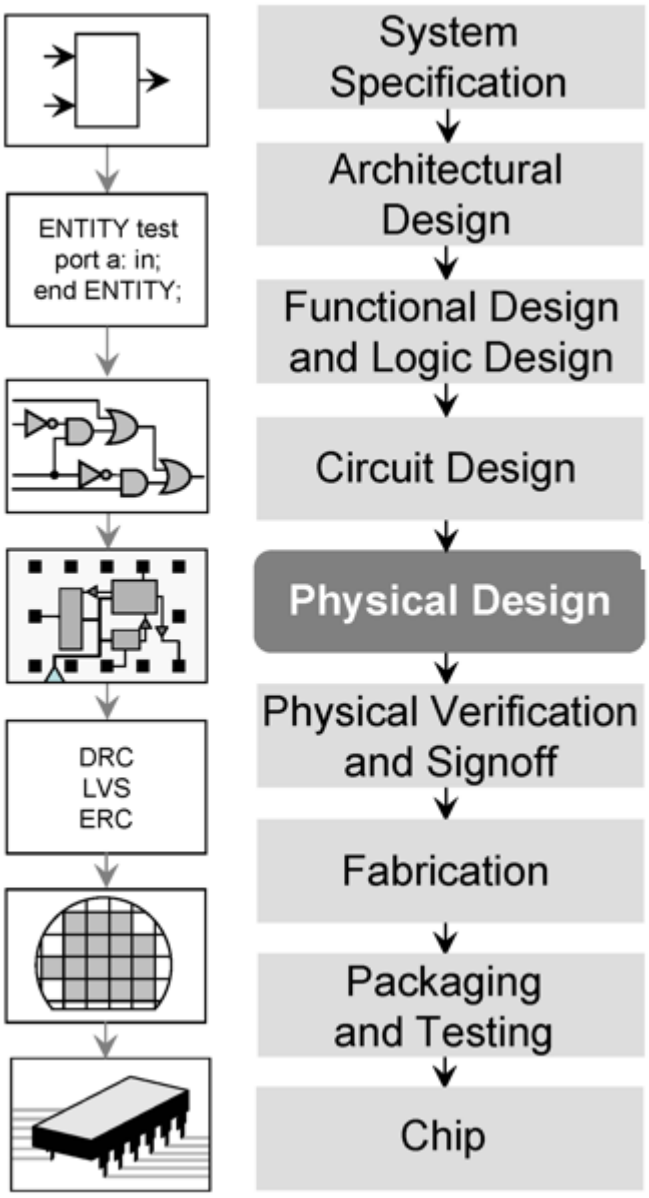


Figure 2.1: VLSI Design Flow [3]

- **Logic Design**
  - Here the logical operations of the design is done like flow control, word width, allocation of different components etc. with testing.
  - As per the operation it is known as Register Transfer Level (RTL) description. For this description some language is required like VHDL or Verilog which are hardware description languages. Simulation and verification is done here with the goal of minimizing Boolean expressions to achieve smallest logic design. Correctness is measured by testing and simulations. Automation can be done at this stage by various tools.
- **Circuit Design**
  - Based on the logic design circuit design is processed. In previous stage we have Boolean expressions, so here they should be converted into a circuit representation by considering speed, power and area requirement. Simulation is done to verify correctness of the circuit.
  - For designing circuit description of the circuit is needed where details of needed elements are expressed like resistors,transistors,gates,cells nad interconnection between them. Representation is known as netlist which can be prepared either manually by schematic capture tools or automatically by using logic synthesis tools.
- **Physical Design**
  - After designing behavior and logical way of circuit now it's time for geometric representation. This type of representation is called Layout. Each component of the design is represented here by different layers which are suppose of doing logical function of the design. Not only place but interconnection is also important and taken care here.

- There are some limitations of fabrication method and material which are given in guideline that must be considered while collecting exact details of the layout. This process is very complex so needed to divide into several steps. many number of verification and validation tests should be performed.
- For the physical design it may be partially or fully automated. Netlist is used to generate layout from layout Synthesis tools. But these tools have performance gap so can not be used for all designs. Manual layout can be accurate more than automated but for complex designs this is not suited way.

- **Fabrication**
  - After all the steps mentioned above design is ready for fabrication. Release of the data is also known as tape out process. Fabrication is done by layer to layer where masks are applied. This mask is important as it measure space required, exact position of different units or removal of extra content. Material used for this process is silicon where perfection is needed for each and every position of unit. For complete fabrication process number of masks are used one over other and layered structure is being ready.
  - Mostly size of larger wafer is of 20 cm in diameter which can be used for fabrication of hundreds of chips depends upon size. This size and capacity is improved more and more day by day.

- **Packaging, debugging and testing**
  - Fabricated wafer is converted to individual chip. After that packaging is done for every chip. Testing is one of the most important factor to be concerned before delivery. Dual line package, pin grid Array , Ball Grid Array are different package type according to usage.

2.1.1 FE(Front End) VLSI design flow

In design flow main there are two parts one is front end design flow and another is back end design flow. The front end design flow focuses on solution of user problem or specifications should be converted in RTL circuit design. It starts with specification and that verification at each step of the flow. All the processes done from front end side then passed to the back end flow, there mainly implementation steps are done for the design. [3]

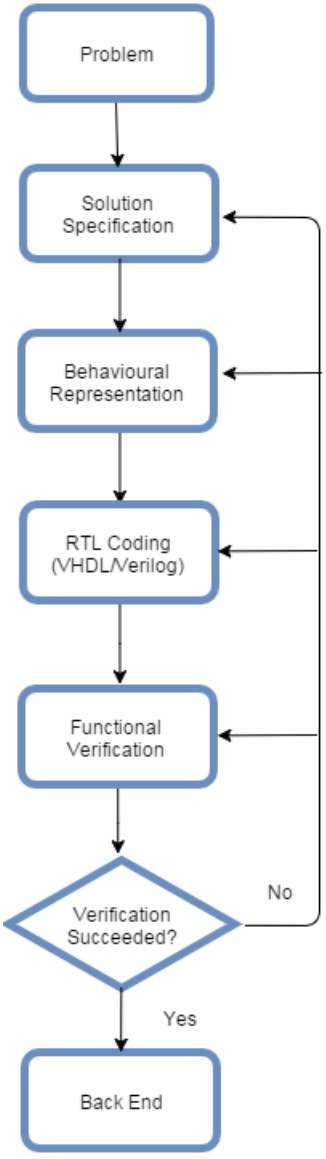


Figure 2.2: Front end VLSI Design Flow [3]

These days designs become more and more complex which can not be handle by any one of the tool because of unable processing for design, compilation and verification issues. For final system on chip to market least design time should be planned and to reach that critical time line advanced automated tools must be needed. Therefore use of AS-2 provides higher degree of design confidence and cutting down efforts in re-spin process that results in less time to market. The system which used by all front end flows known as converged. That should amalgam means validation and compilation of circuit should easier and faster. While streamlined system is compatible with today's circuit designs and validation of different blocks. [3]

2.2 IP Reuse

To increase the productivity one of the possible method is design re use. That means re-exploration of the design modules which are already existing into different context. Leads this concept to IP reuse where new modules are generated using existing IP modules which explains IP is nothing but module with reuse capabilities. Reuse capabilities then lead to concept of 2-step compilation. [3]

2.2.1 Structural v/s Behavioral Modeling

At the time of modeling the hardware code can be written in many different ways irrespective of HDL model you are using like either in VHDL or verilog. In behavioral modeling method to write code is by considering functionality of the design more like problem solving algorithm. But only the major concern with behavioral modeling is lack of support for re-usability. Processing like loading, simulation and elaboration done at same time which results into more market time for any of the design. Not only time but quality and optimization is also not up to the good level because of non-modular nature.

In structural model coding is more based on partitions as final code is ready by collecting small small coded parts. For bottom-up approach first to create behavioral

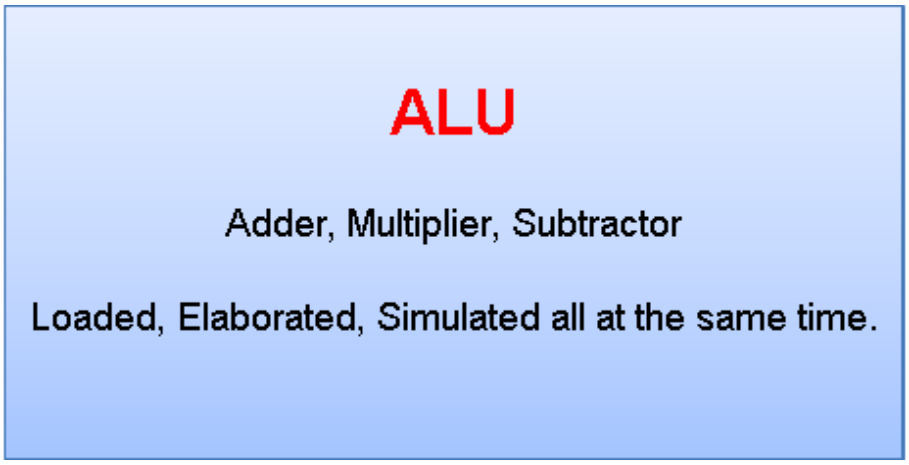


Figure 2.3: Behavioral model [3]

code for least size of module and after that collection of designs you can come up at the abstraction part of the design which is basically structural code of the design. Modular design can be given by structural part of the code which ultimately supports re-usability of the design. If design module is small then it is easy to optimize them day by day so for final design also small small modular designs can help to optimize overall design.

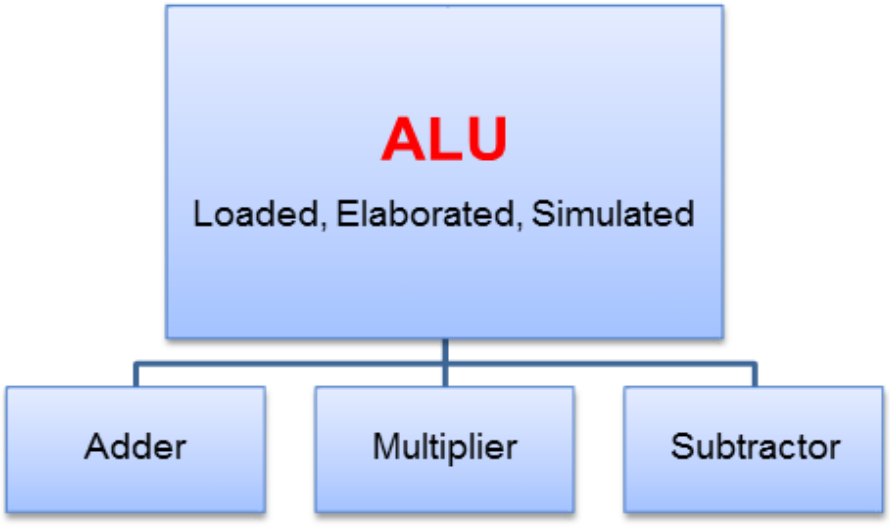


Figure 2.4: Structural model [3]

2.3 Two step compilation

As we seen in the previous section IP reuse is based on the modularity where once any IP module is compiled it will be linked to entire soc module. This way compilation is done twice one for individual stage and other is at time of integration for entire module. So it is known as two step compilation. Parallel processing is very easy with this way moreover linking with SOC is also simplified. Concept of IP reuse can only used for the RTL collateral and not for the Validation collateral. There are several categories which must be satisfied by module for re-usability. Categories are mentioned below: [3]

- Configuration- To solve user problem it must be constructed.
- Portability - Portable anywhere irrespective of tools and technology.
- Debugging- Should be verifiable and validated to make bug free.
- Readability- It can be easily documented either acceptable, applied,restricted ot defined interfaces.

Any of the module can be classified by this module on base of re-usability.

- Functionality - Least requirement to be satisfied for module to work logically.
- Maintainability - it means being functionally correct, a maintainable module which is well documented, with clean and commented coding.
- Re-usability- Many improvements can be done on re-usable model.

"In 1997 one industrial corporation called VISA - virtual socket interfaces alliance was established for the purpose of module re-usability. Purpose of this corporation is to create set of standards for re-usable IP. All standards created are very well defined, explained and accepted in semiconductor world".[2].

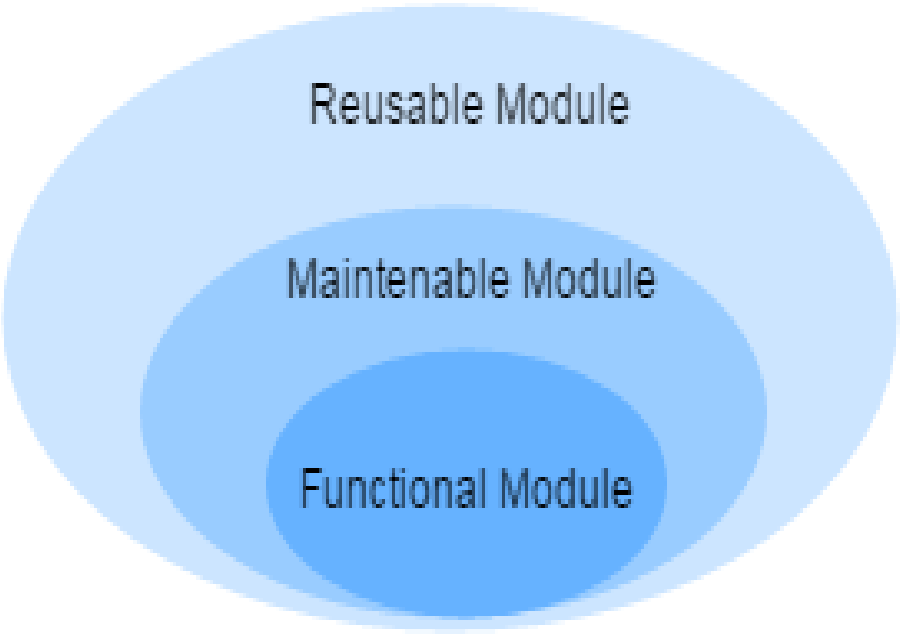


Figure 2.5: Classification of IP [3]

2.4 Legacy and SOC Integration Flow

Today's generation soc's and ip's are different in design so flows used for validation are not compatible with them. Basically two flows are there which used for this purpose one is Legacy flow and second is soc integration flow or advanced flow. Not both the flows are perfect one common disadvantage in these flows is time taken to compile soc design which is large for time to market. [3]

2.4.1 Legacy Flow

Legacy flow is basically worked on the concept of behavioral modeling design description 2.1 This is basically non-modular design model description and not support IP reuse concept results into no two step compilation. When design can not be optimized quality improvement is limited and compilation time is higher with lower performance.

2.4.2 Advanced flow or SOC integration flow

Advanced flow supports modular design model as well as IP reuse which results into savage of design time and time to market. But only the lack in this flow is it has not proper qualification step. So maintenance cost would be high and also difficulty in deployment is increasing. Debugging is also more complex when flow failed at some point. Managing design complexity is hard as well as compatibility with growing technology is difficult. [3]

Problem with this flow can be solved by system like AS-2 which takes all the benefits of the above flows and help in reducing compilation time drastically which results in the reduction for time to market.

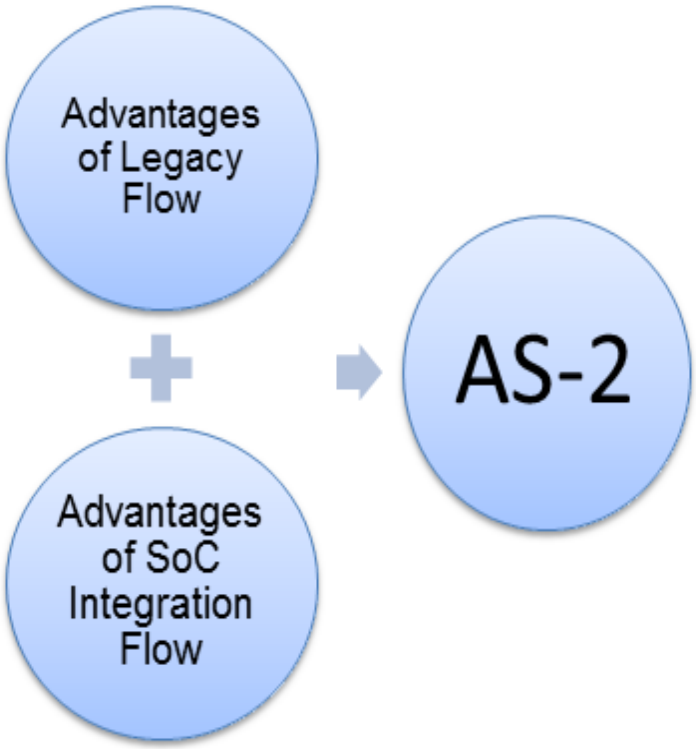


Figure 2.6: AS-2 system implementation [3]

Table 2.1: Flow comparison [3]

Sr. No.	**Legacy Flow**	**Advanced Flow**	**AS-2 flow**
1	Behavioral model	Semi structural model	Structural model
2	Can't extend with new technology (No IP reuse)	Partially IP reuse possible	Completely IP reuse possible
3	For small designs	Not scalable for larger design	For complex designs
4	Only single stage compilation	Not modular flow so hard to proliferate	Two step compilation
5	Slow performance	Slow performance	Moderate performance

2.4.3 Summary

For the loop holes of current validation framework need is to measure performance and improve quality for next generation validation framework.Small improvements in the next generation validation framework results into massive improvement in performance and time to market.

Chapter 3

Next Generation Design Validation Framework

In this chapter brief idea is given about next generation design validation framework. Improvement over previous validation framework is also explained. Framework is explained by block diagram and also functions of all blocks. Loop holes in the framework is listed where improvement is needed which is the goal of the project.

3.1 Introduction

Next generation design validation framework is basically a wrapper which provides environment for electronics design automation tools. It is build on design validation framework to add more powerful capabilities for caching and parallelism. This allows the flow to scale up to higher capacity designs still maintaining compatibility with previous framework. It is a generic front-end build environment because it is project/platform independent and can be deployed and used by any design (soc/ip) with correct design configurations. Framework uses project supplied configuration files to describe a projects design data. The design data can be RTL source files as well as test source files, project custom flow steps/processing, specialized hooks/scripts etc. [4]

3.2 Validation framework architecture

Validation framework flow actually consists of multiple tools/APIs closely interacting with each other. The master tool entry for framework which contains below list of sub tools.

- Flow manager
- Scheduler
- Configuration engine
- Caching interface
- Flow hierarchy
- Operation engine

Input from the command line given to primary executable script to flow manager and it gives to other sub tools as needed. Here flow manager works as the core piece or a mind or the flow which interacts and controls everything. Configuration engine gathers and pre-processes all the data from different input configurations (design configuration files) and feeds processed design data back to flow manager. Scheduler schedules all generated data as per requirement. All processed data is given to operations engine which takes in required objects from scheduler and configuration engine. Here all necessary operation like certain checks are done. [4]

Caching interface works as cloud which is the memory of the flow which remembers previous successful runs and saves precious build time in case no incremental changes detected and File generation works as template expansion API. Flow hierarchy block shown in the diagram mainly handles Flow hierarchy generation and it

is part of design validation tool itself. Flow hierarchy handler and flow manager are mainly tools which interact with configuration engine and pass the extracted design data to tool for further processing and compilation.[4]

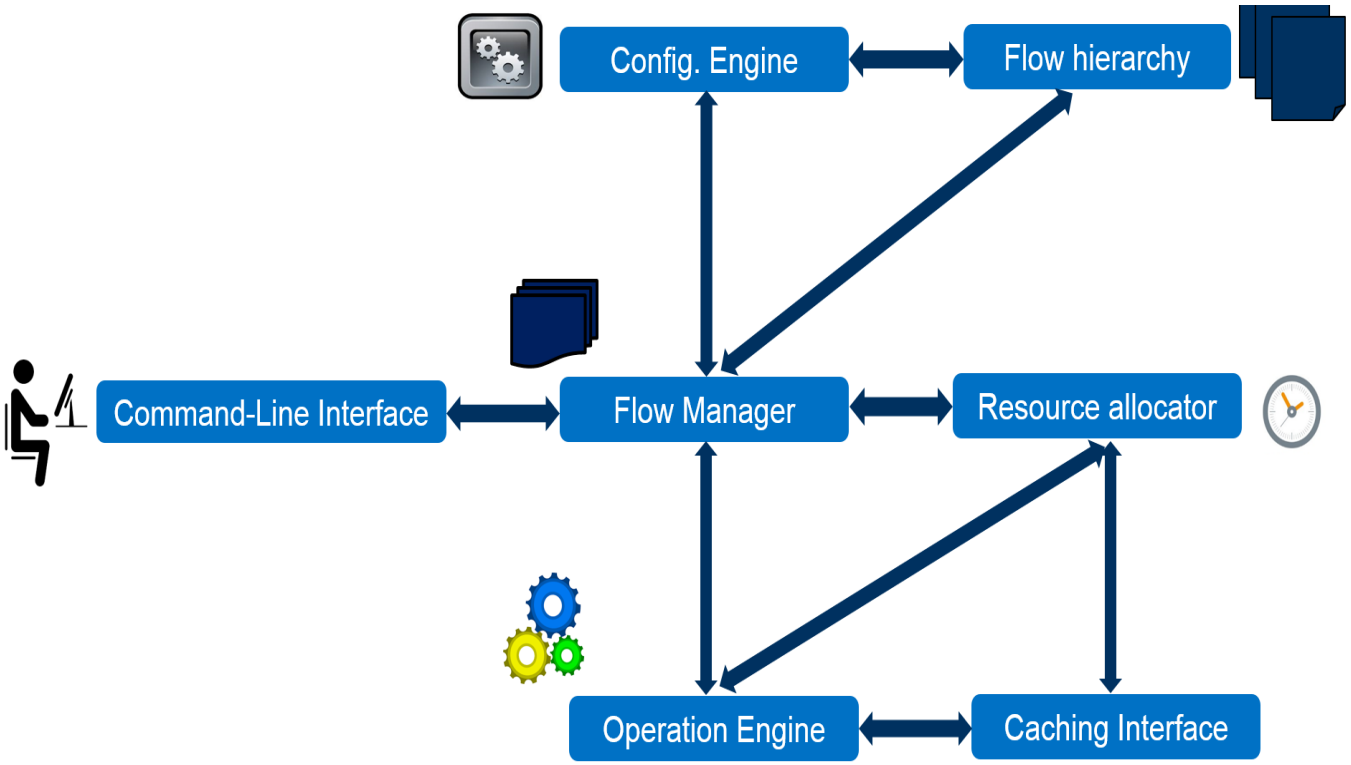


Figure 3.1: Framework Architecture [4]

3.3 Flow Manager

Flow manager is the heart of the framework. This includes the infrastructure that parses the flow specification (Flow hierarchy). Information or specification can be get by command line and configuration options. Here it figures out which design steps/tools to run and interfaces with the scheduler to run them. This infrastructure is agnostic to the actual commands/tools being run by the flow and has a standard API for all the tool wrappers (design steps). Therefore projects using different build flow hierarchy can still share a common flow infrastructure.[4]

This is a Perl module that contains the flow specification. RTL tools divides the entire analysis flow into a set of design steps - each step runs a tool/script to perform one of the functions (code generation/code manipulation/compilation/run static checks) of the flow. Design steps can be logically grouped together into flows. The Flow hierarchy module specifies the set of design steps (flow) that needs to be run and the dependencies among them along with some configuration information. By enabling these configurable flows and design steps via Flow hierarchy. Framework allows projects to customize the flow to suit their methodology.

3.4 Scheduler

Scheduler is one of the important tool of the framework. This runs the design steps/jobs and communicates their status to the flow infrastructure. All design steps or jobs are run on the local machine. It uses special tools underneath to run jobs on processing. Scheduler can also be shared across different projects.[4]

3.4.1 Caching Interface

This feature is the enhanced version from the previous validation framework. RTL tools use this block which can be used to enable incremental and cached results for certain design steps. Caching interface remember the last successful run. If again command line input demands same flow with no incremental changes then it directly provide last successful saved result so that time consumption should be less and also processing for the framework can be saved.Only last successful data is being saved so no extra amount of memory or processing is needed.[4]

3.5 Configuration Engine

Configuration engine is the tool used by flow manager for the framework. In the previous validation framework this block was used with some limitations so here this

block can be used for some module re-usability so that it can compatible with previous version and added new features in the next generation validation framework. Main aim for this module is to make it more sustainable, maintainable so that it can satisfy the increasing need to support much more complex design environments and multiple flow methodologies.[4]

3.6 Flow hierarchy

Flow hierarchy is very important module and connected with flow manager as well as configuration engine. The main intent of flow hierarchy handler is to analyze the design data and prepare an optimized flow network. After that it should convert those to flow hierarchy files which can be loaded into master flow hierarchy while building the flow.

Handler intelligently looks at the library definitions and defined attributes to decide what design steps are relevant for a particular library. It also takes in a flow step configuration (part of flow step packages) as input which gives information on what particular design step in a flow cares about what types of spec attributes. Any flow package which is default loaded by flow, can call flow manager and register a new flow step to flow hierarchy. A design step can be of three types with respect to handler.[4]

- Library type flow step: These design steps are mainly compile/analysis steps which process library data and provide output. Flow hierarchy creates instances of the step for each library and runs it for every library.
- Model type flow step: These steps mainly are elaboration type steps which work at the model level.Handler creates instance of these kind of steps per model.

- Global type flow step: These steps are global and only get run for the particular instance they are called for.

3.7 Areas longing improvement

We saw validation framework and next generation validation framework also. Next generation validation framework have significant improvement in time factor as well as parallelism than previous framework. Still all features of this framework are not perfect. In some parts of the flow improvement is needed for the better performance. Below mentioned some of them:

- Lack of powerful user interface
- User friendly
- Efficient debug mechanism needed
- Less number of test cases
- Not enough debug help
- Missing corner test cases
- Needs improvement to fill quality gap

3.8 Summary

Hence, in this chapter we seen brief description of design validation framework. Function of each block is described and also interconnection between them. Flow of the architecture is shown from starting with user input to the output of complete flow. At each and every step verification,validation and testing is key part. Next generation validation framework shows qualitative improvement in features like time to market, complexity, etc. Also factors mentioned in the last section which needs an improvement. So for those factors quality improvement and debugging is needed.

Chapter 4

Extended debug mechanism

In this chapter, debug mechanism for the next generation validation framework is given. For the current flow debug scenarios are available but some information is till not getting by the framework tool. This need to be explored to get better performance and user friendly environment.

4.1 Need of debugging information

In simple words for current operation of system or computer finding issues and defect resolving methods is called debugging. For a software development, this includes locating errors in the flow. Basically this is the part of the software testing process and an integral part of the software development life cycle. This process starts as soon as some script is written and continues in successive stages with combined to other units of programming to form a software product. For large program that has many lines of code, there the debugging process made work easier by using different techniques.Principles of debugging are mentioned below: [4]

- **Immediate report of error conditions** - Most of the debugging time is spent is on the cause of errors. If earlier an error is detected then that is easier

to find the cause. Statement for incorrect module state detected as soon as it arises then the cause is also determined with minimal effort. But if it is not detected until detected at client interface then it may be difficult to overcome and detect the list of possible causes.

- **Easily interpreted and maximum useful information** - For ease of interpretation maximum useful information is desirable. For data structures ease of interpretation is important. In the entire data structure sometimes small module errors cannot easily be detected. Form of data structure is needed to easily visible information for correctness.

- **Distracting and useless information minimization** - Too much information is messy sometimes. If some flow debugging is like to show all entry and exit form for every procedure then it is very difficult to find useful information from there.

- **Use testing case avoidance** - Avoiding user testing because it is very complex to test entire complex flow. So for this type of work it seems to waste time in debugging. This idea works only when some parts are reusable.

This is only one side of debugging. Not only errors of the flow but also information extraction for the flow is known as debugging. If there is some flow which process something and gives output then there is possibility to extract data at some intermediate stages. For the next generation validation framework debugging is the very important part. Processing complex design data is big task itself so there debugging helps a lot.

Flow have large data to process as moving to complex and big designs therefor processing data and generating results is very time consuming. There are some points where one can extract information from the processed data. If at early stage some data are generated then there is no need to wait until end. At that stage only

one can get data by debugging. SO scripts and programs are written to extract the information from intermediate staged processed data. By this way flow can give important data to user without consuming more time. [4]

4.2 Command line interface

Command line interface is also known as command-line user interface.It is designed with purpose of interacting with a computer program or tool where the user gives commands to the program in the form of some lines of text which is also known as command lines. There are basically two type of command lines are there one is operating system command line and second is application specific command line interface. OS specific command line is very important as all computer have this and can be used by command prompt for interaction. Second is application specific command line. Mainly these command lines support any or all of command line interface mechanism as below[4]:

- **Parameters**: For OS additional information is supposed to pass at launching time, we can say at booting time. At the time of launching program from OS command shell with additional text provided with the program name.
- **Interactive command line:** Once program is launched then some operator is provided independently to enter command in form of text.
- **Inter-process communication for OS:** Like standard streams and pipes mostly OS supports means of inter-process communication.So command lines from user interaction can be processed by any of this method.

In our framework command line is very much important. As we saw in the introduction of the next generation design validation framework input to the framework is by command line only. User give specification of required data or tool by command line interface. With different arguments framework will understand the need

of the user. There are number of categories of EDA tools supported by framework. User can pass data for any of tool he/she wants to process. In the command line user provides tool name to be processed for with design specification or design description. All remaining process is take care by the flow manager.

Flow manager understands command line by dividing into different categories. It requests and pass specific design data to resulted into organized one which is understand by the framework. Next generation design validation framework is very powerful that it supports different category of tools as well as large amount of design data. Command line interface is not only limited to give input to the framework but also uses for extracting information at different stage of the flow from processed data by framework. This basically enhance debug mechanism and user interaction with the framework.[4]

4.3 Debug feature in next generation design validation framework

In next generation validation framework debug feature is supported by command line interface. As mentioned in the previous section by command line one can extract useful information. One of the information type is about tool. In command line at the time of input user mentioned tool name and design data for which deign is to be build. Framework starts processing after getting command from user by command line. First and foremost control is given to flow manager which is the heart of framework. Flow manager's first task is to gather design specific data in organized way. For this purpose configuration engine is used.

Flow manager passes primary data to configuration engine. Configuration engine takes data files of tool, project design data, configuration files and some supportive

data files for the process and generate design database in unique format. After creation of design data specifications it passes data to operation engine. Now some of the data of design specification is important to know for user that which supportive file or libraries are used for the processing. Rather than waiting for complete flow to run it is very convenient to provide this information at the time of creation if demanded by the user so if there is something wrong or different data needed to process than running design, it is very easy to debug at earlier stages and also time consume in gathering information. [4]

4.4 Enable Log::Log4perl support for logging

Log::Log4perl gives powerful logging API for perl program.Logging is better than a debugger as you know what's happening in your code during runtime. Traditionally logging packages are too static and generate lots of log messages in your log files that won't help you.

Log::Log4perl is different than debugger as it allows developer to control the number of logging messages generated at three different levels:

- At a central location of your system may be in startup file or in configuration file developer can specify which components (classes, functions) of system that should generate logs.
- Coder can specify level of details in logging by specifying logging levels.
- Appenders can all be initialized that on screen that log message should on screen and also to feed log messages to log file in specific format.

Basically this mechanism is flexible as user can turn on and off anytime and also it prints message in specific format with specific level of details.

For example take any perl module where turning on detail logging messages can print lot of messages on screen and all are not useful which is waste of time

and memory. With help of Log::Log4perl system can be make to able print only for several type of messages and one can print all messages to different log file by making small change in configuration file.

There are two ways to print info via Log4perl package. One is through command line and one is through configuration file. In configuration file we need to initialize appender with name, log file name, log file layout, log file mode etc. Once initialized in config file then can be use in any command by just initializing package name and info message.

```
package My::MegaPackage;
    use  Log::Log4perl;

    sub some_method {
        my($param) = @_;

        my $log = Log::Log4perl->get_logger("My::MegaPackage");

        $log->debug("Debug message");
        $log->info("Info message");
        $log->error("Error message");

        ...
    }
```

Different log levels present in the log4perl package are following:

```
$logger->trace("...");  # Log a trace message
$logger->debug("...");  # Log a debug message
$logger->info("...");   # Log a info message
$logger->warn("...");   # Log a warn message
$logger->error("...");  # Log a error message
```

```
$logger->fatal("...");  # Log a fatal message
```

4.4.1 Mail support through log4perl

In some cases while running flow at runtime user wants notification if something goes wrong rather than waiting till end. This feature can be enabled by using the Log::Dispatch::Email::MailSend module in configuration file which allows user to specify recipient and subject for emails of configuration file.

```
log4perl.category = FATAL, Mailer
log4perl.appender.Mailer          = Log::Dispatch::Email::MailSend
log4perl.appender.Mailer.to       = drone@pageme.net
log4perl.appender.Mailer.subject = Something's broken!
log4perl.appender.Mailer.layout   = SimpleLayout
```

4.4.2 Enable dereferencing in log message

If any code have hash structure then as log message it prints something like HASH(0x81141d4) which is not desirable. Log4perl have support for dereferencing also to print proper message format.

In Log::Log4perl 0.28, there is a better way to print it like below:

```
$logger->debug( {filter => \&Data::Dumper::Dumper,
                 value  => $ref} );
```

4.4.3 Convert selective debug tags to log4perl format

In next generation design validation framework selective debug mechanism is enabled to print tag information for every working block of modules in flow. This mechanism is useful for printing information but mechanism is very complex and it also takes lots of time to print all messages. One more drawback is there is no

selection of categories for print messages. All log messages are printed on screen irrespective of log message levels.

This drawback can be overcome if log4perl is used instead of selective debug mechanism. Standardize the way of logging and also easy for developers as lots of information and help is available regarding the use of log4perl in perl modules. This support brings all features of standard package like different levels of log messages, enabling specific part of code to print messages, mail user in case of some failure. All these features are not present in current debug mechanism in framework.

This feature can be added in framework by their block structure.For flow manager enable logging messages in particular working module, now that module can be called in one of the unit test. Log4perl is the standard module which enable print messages by -debug switch when running test. That means adding specific tests for specific modules help to print logging messages for particular working block of flow. It will be easy for the developer to see specific information by this way and no need of debugging after failure of test. At different stages of module logging messages can be enabled and easily detect error at different level without passing test for failures. By this way enable log4perl can replace existing non standard and non-efficient debug mechanism with standard logging pattern. Extra features are also provided by standard module which make system more efficient and improve quality of framework.

4.5 Help commands

As mentioned in the section earlier that next generation validation framework supports user interface for extracting interface. Design specification is generated by configuration engine after giving command line data by user in flow manager. Design specification is a complete package where all the data is organized in specific manner. In design specification there are information related to libraries, unified data files used, etc. For every information there are commands designed for ex-

tracting information like show_lib_info show library information used for the specific design. Different parameters are passed to extract detail or specific information from generalized one. These show commands are best as making framework user friendly.

All scripts of the framework is designed in perl. Design data base is particularly generated in hash format with information in the form of hashes of hash. This is an organized form of data. To create any help command there is particular script need to analyze command line options and according to that extract information from the hash of design database. By analyze hash structure generated by configuration engine it is easy to see which information can be extract and by which depth. Enhancing more and more information through command line makes framework more user friendly as well as better performer in time factor.[4]

4.5.1 Enabling new help command

As we seen in the previous section there are number of help commands available in the present framework. But still many more information can be extracted from the design database. One of the important parameter for the flow is library. There are different types of libraries like partial libraries, dependent libraries etc.

- **Library:** For any design lowest level of specification is a library, source files needed, specification that can be used to generate a set of results. The results can be anything from analyzed libraries to generated RTL/C/XML files etc.) Libraries are immutable. A tool cannot be run until the library definition for that tool is known. Once known, it is immutable - it will not change. The library definition may have one or more content types (RTL, C++) and the results might include analysis results, generated RTL files, etc. The generated content can be spread across multiple directories and can include multiple content types.

- **Partial-library:** Currently supported in framework configuration Implies

that the results are a part of the parent library results. Enables merging/-packaging multiple libraries into a single library. You can keep the command lines for the different physical libraries separate while packaging them all into the same logical lib Mostly applies to compilation tools.

- **Dependent libraries:** Dependent libraries include list of libraries that must be compiled before this one.Dependent libraries are very much important as it show need of libraries to compile that design successfully.

All this library information is very much important. Partial libraries and dependent libraries should be known which clears design data dependencies. Rather than tracking entire hash structure if there is some script which can provide information related to libraries then it will be very useful for the user as well as from debug point of view. Following figure shows the flow of show command from user interface to result generation.[4]

Here the complete flow of help command is shown. User gives input via command line. As per the design of framework information is given to flow manager first. Now design data must be in machine understandable language so first it should be compiled.In next generation validation framework verilog compilation is used first. After compilation compiled data passed to configuration engine. Configuration engine takes all the required data like configuration files, compilation files, libraries etc. and generate design database. This design data specification is in hash format. From this complete hash only required data should be extracted out. Different help commands have common script which process same hash and different information extracted as per different arguments.

Not only information extraction is important but generating information in some specific format is also important. Visibility should be good of generated output which is provided to the user. As part of enhanced debug information show_design_lib_info is generated. Purpose of this command is to extract dependent libs and partial libs

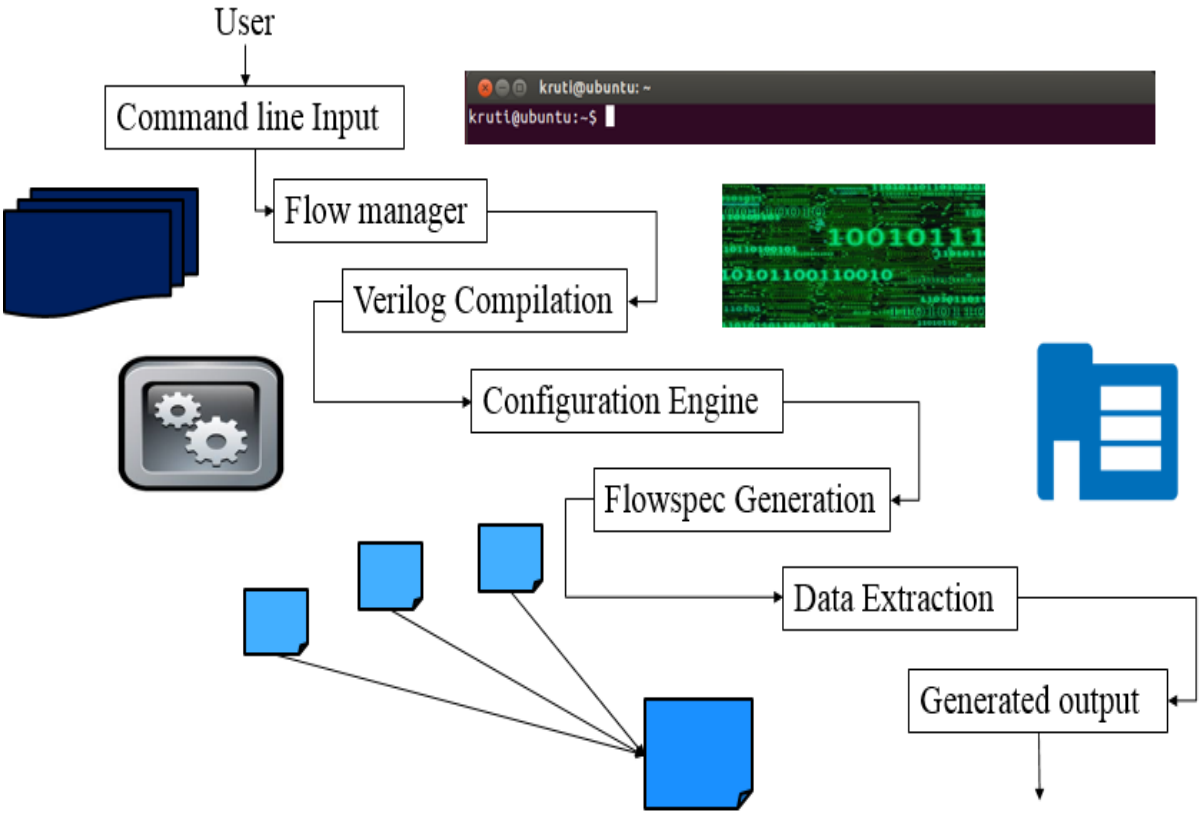


Figure 4.1: Flow of show command [4]

for particular design. Every project have different models. Models have libraries according to design. This information is shown in table format using perl module Text:Table. Result is as show below.[4]

As shown in screen shot of the result, user will give input via command line by typing show_design_lib_info (arguments). Arguments can be given for limiting or specifying particular data. This command will generate scope of particular design, library name, partial library name and dependent library name.Now different arguments can be passed with this command like for only specific scope result should be displayed. For that we have to enable different command line options in the script to filter out data. Hence, by developing such commands performance of the tool can be improved. [4]

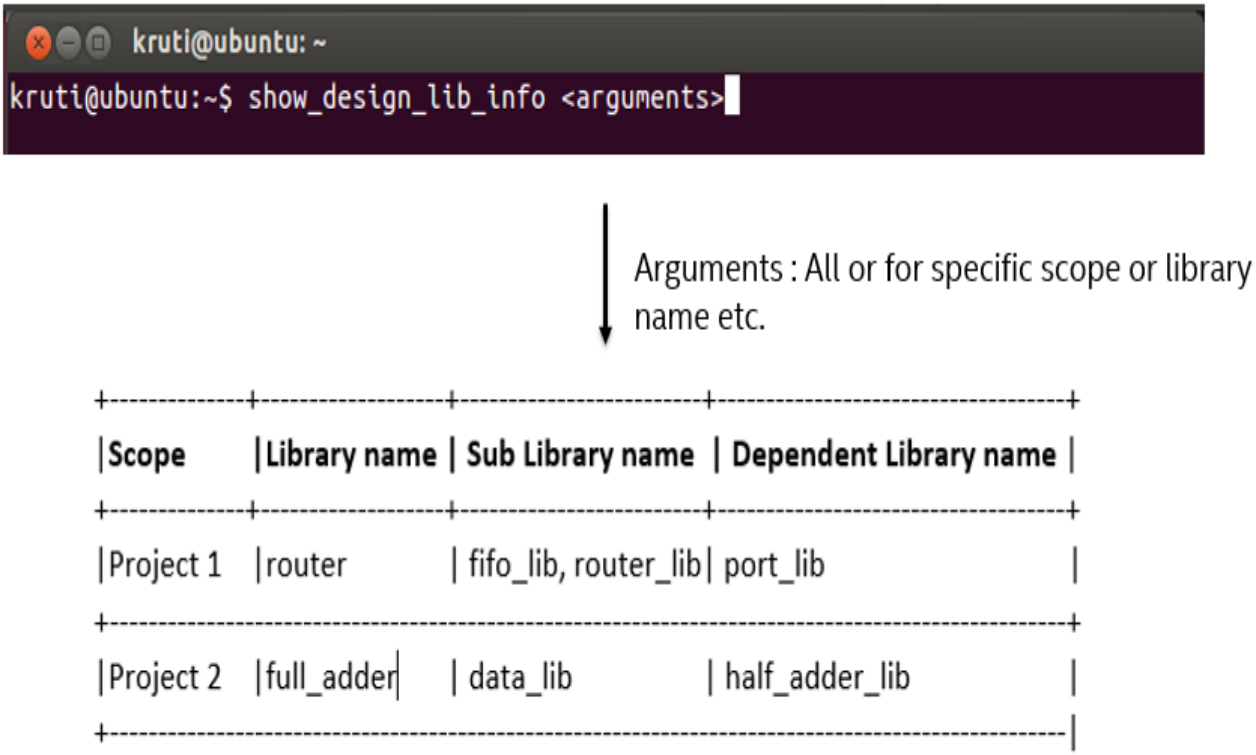


Figure 4.2: Result generated by show command

4.6 Summary

This chapter gives details about extended debug mechanism which is the core part of any framework. There are many ways to extend debug mechanism one is by extracting information from the processed data. Here introduction of the help command is given which provides information of the design to user and after that new help command feature is added in the same. Result of the developed show command is displayed. Also explained importance of command line arguments. Use of standard packages for logging message brings efficiency in framework. By this way debug mechanism of design validation framework can be enhanced.

Chapter 5

Dependency Analyzer Tool

In this chapter, dependency analyzer tool is explained. Need of the tool is mentioned which can do quality improvement for the next generation validation framework. This tool is basically a script which can extract dependencies and compare them from design data. This will help to find any loop holes in flow creation.

5.1 Need of dependency analyzer

As mentioned in the extended debug mechanism more and more information can be provided by extracting data from the design data base. Via help command user can know about files, stages or libraries used for particular stage from design data. Purpose of this extraction is to give user specific data without waiting to run whole flow. Now for the further processing after design database generation control is given to flow hierarchy generator which generates complete structure for the flow. This hierarchy have all the attributes and elements for particular stage of the design. There are file lists for the particular stage available in the tool as demanded by command line.[4]

Flow hierarchy generates hierarchy for the design data and provide it further to flow manager for compilation and operations. If there is some fault in the genera-

tion of the flow hierarchy then for particular design data wrong dependencies passes to flow manager for the process. It is possible that generation of the flow data may have extra dependencies, missing dependencies or some other design's dependency. This can not be tolerated as this further processed through all over flow by flow manager in the framework.

Second problem with the framework is when user gives design data through command line it generates file list. If user is not aware of exact dependencies of the design then it might be possible that user can put some extra dependencies or less than required dependencies as input. If there is the case then for extra dependencies also flow will run and time consumption is more in this case. While if dependencies mentioned are not enough required for the design then after running whole flow it shows an error that missing dependencies or flow termination for not finding data required to process. This will take too much time to generate output in the framework.

So for the problems mentioned above solution is script which can identify correct dependency for the design so that framework shows dependencies and can be checked before final generation of the result. This tool is powerful as more and more features can be added related to comparison of elements for stage instance. Tool currently can show dependent library information and compare them. It can be enhanced by adding more and more extracted information from the flow. [4]

5.2 Functionality

Dependency analyzer tool is basically a perl script to show dependency information. In the design validation framework flow manager is taking data from user via command line. Now this data is fed to configuration engine for generating design database. Flow hierarchy is the block which interact with this configuration engine. Basically flow hierarchy generator generates the flow database which have

all information regarding design. When user pass information in command line it is supposed to pass arguments necessary for design data to process further. This information is stored as file list with arguments and attributes. Flow manager generates it's own data base which have all data related to design needed to compile and build for particular tool. This data generation is done by flow generator. So output of the flow generator must be verified in some case before processing further. [4]

Dependency analyzer tool is supposed to take two input files for comparison. One is file list where all the design data is located. This file has list of all stages, models, libraries and some other attributes. Second input is flow generated data for specific or demanded design. This file have lesser attributes than file list of the design. Now first tool extract dependent library information from both the files. Tool is written in perl script which process hash format of both the files. From the hash particular libraries and their dependent libraries are extracted first for mentioned design. Later on this data is send for comparison.

While comparison first library names should be compared and if library name is matching then further dependent libraries should be compared. If library name matches then it will take data for that particular library. For data comparison various regular expression are used. After getting final data from hash structures it is showing as output in two different categories like different dependent libraries and same dependent libraries. For ideal result all libraries are mentioned in same libraries which shows both design data and flow generated data matches in correct dependent libraries used for the process by framework. If some result is showing as different libraries then that must be going for debugging to correct the functionality of the framework. This script is powerful in user interaction as it can take design files (input files) and library name from the user. So that user can see debug information for particular attribute.[4]

5.3 Generated results

For the generated tool script expect input data from the user and if user data is empty then show error at output. Input format for the tool is as shown below:

Figure 5.1: Dependency analyzer input

This way tool expect three input data from the user two input files and one library name. After getting this data, tool extract information from the file and show difference in dependent libraries for input library name as shown in figure below:

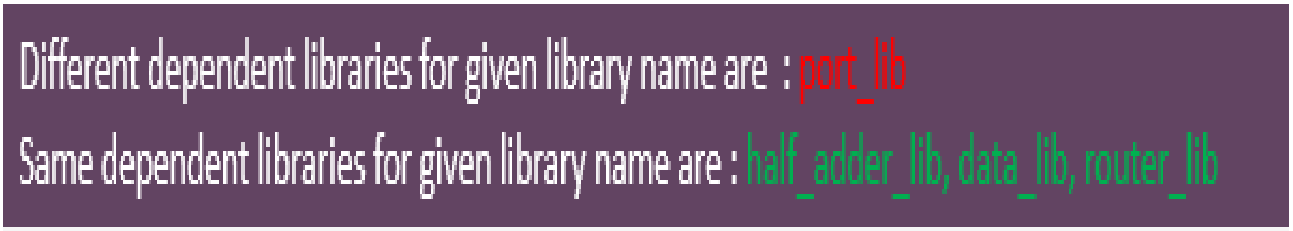


Figure 5.2: Dependency analyzer output

As shown from the figure same libraries which is the expected output is shown in green color and different libraries which shows some problem with the flow or input data is shown in red color. In this way dependent libraries for particular library are generated in form of comparison which saves time for user as well as flow in case of any fault in flow generation. If user gives some wrong input then output is showing error to correct the name. In this way tool shows proper errors in case of wrong input and shows output for the correct data input.

Figure 5.3: Dependency analyzer error

5.4 Summary

In this chapter complete flow of dependency analyzer tool is explained. Possible problems with the flow are mentioned and what could be the solution by dependency analyzer tool is given. This tool can be time saver for very complex design data processing. More and more functionality can be added to enhance the scope of the tool in the framework.

Chapter 6

Performance Analysis

In this chapter, introduction of performance analysis is given. Further, performance analysis for particular commands at different stage of the flow is shown and also performance improvement in the framework. There are main three factors for any VLSI design, they are Testing, verification and validation. In electrical engineering domain VLSI testing is generally used.While computer science mainly used verification and validation concepts. But at industry level all three factors are treated and processed differently. There is one common similarity in all three is underlying the framework build.

6.1 Need of performance analysis

Verification includes some part of performance analysis. Code coverage is one of the measure to analyze code utilization for the flow. This is done to see how well the design is working and verified. Functional coverage is different than code coverage as functional coverage checks functionality of the design. In next generation validation framework there are many advantages but still lacking at some performance points.[1]

- Integration problem is not shown as per requirement.

- From previous version some improvement is not bringing performance improvement. This affect while processing complex design data as it takes large time to process.
- Some code quality checks are not up to the mark.
- While building data flow there are many recursive blocks or hierarchy need to be removed for better performance.

The proling analysis on the next generation validation framework is done by Perl profiling tool NYTProf which is being introduced in the next segment. This tool is efficient and accurate in showing timing information.[1]

6.2 NYTProf - Perl profiling tool

Perl profiling tool is used and important to provide information related run time statistics like time consumed by each function, number of calls for particular function and mentioned line numbers in particular file. These features known as profiling. Every run of the command starts running with the profiler and at the end it generates result of profiling. [2]

- Profiler done two types of profiling one is statement profiling and other is subroutine profiling.
- Supports small resolutions up to nanosecond.
- Result is generated in HTML format having coverage of code, statements, number of calls in organized manner.
- It maintains extra information in the file for further analysis.
- Record keeps data file names, calls, subroutines etc.
- Different processes can be shown differently in hierarchy tree so that all processes can seen differently with performance and avoid any overlapping.

- Analysis can be done for single output profile file and after that it can be merged with multiple files.

6.2.1 Subroutine Profiling

Basically this type of profiling evaluates the time between starting of a subroutine and ending it. For call of any subroutine starting time is noted down by tool and also call count is measured. Every line number and location is recorded for analysis.Different subroutines are mentioned with their name. Total time taken by the subroutine gives us information about time consumption as well as which subroutine call redundantly can also be extracted from the analysis. [2]

6.2.2 Statement Profiling

This type of profiling measures time between starting of one Perl statement and starting of another statement. Statement coverage fundamental is similar to this. Call for statement is also counted and shown into result.[2]

Figure 6.1: Statement Profiling

For the looping conditions first call of the statement and last call when it exists the loop is recorded. So basically number of time the condition becomes true is shown by analyzer.

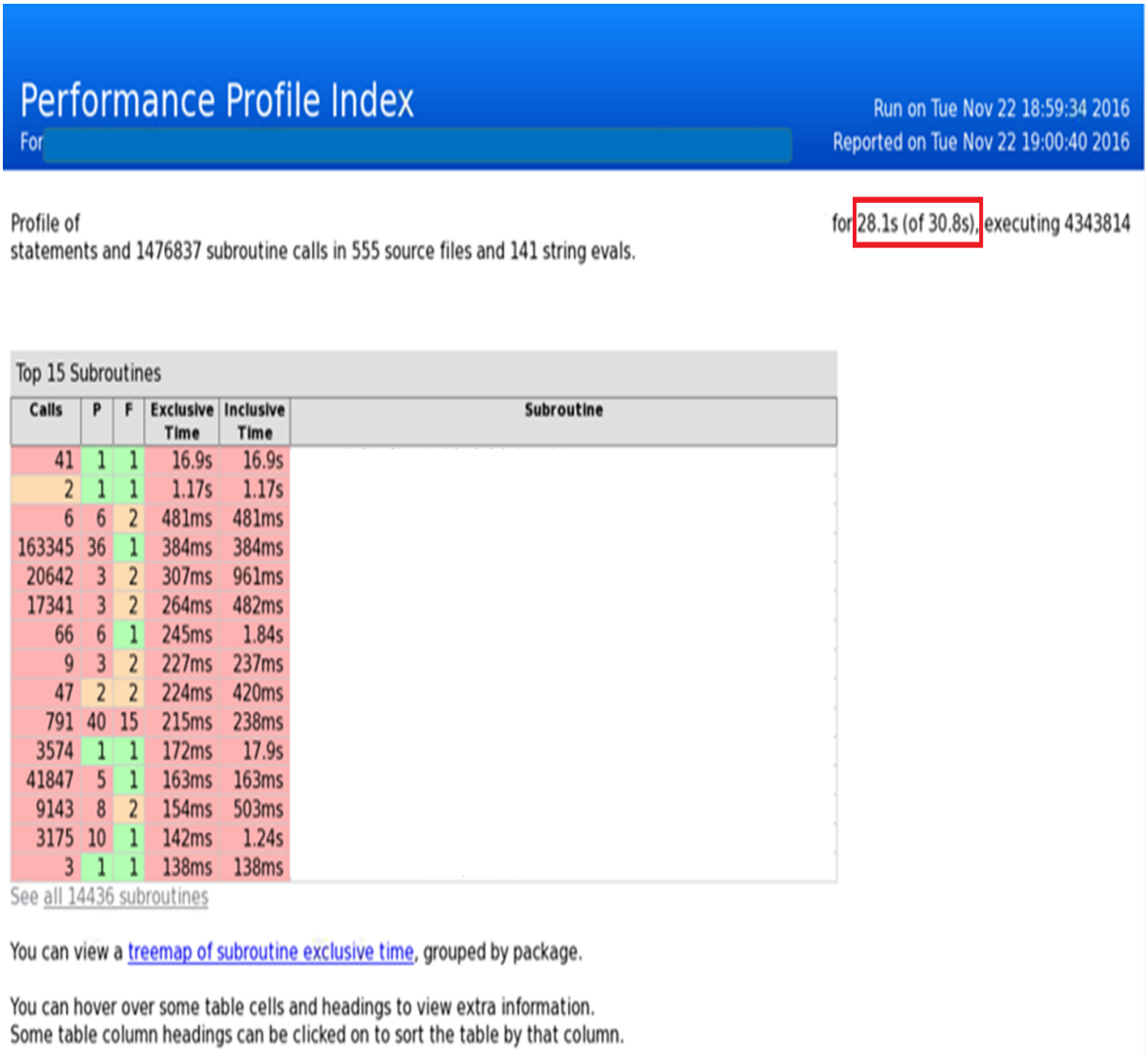
6.3 Performance analysis for positional change of block in flow

Next generation validation framework have many blocks which perform their function and passes data in the flow. Flow manager handles all these activities. User gives input from the command line which is passed to flow manager. Flow manager gathers information related to design from the user and passes it to configuration engine.Configuration engine takes input data and generate design data base. This design data base have all the design attributes required to process. [1]

As we saw in previous section that help command is one of the utility of the tool where user can extract data before final result. These help commands uses data of database generated by configuration engine. Now some of the help commands generated output which is available before generation of design database. One of the help command is to show user defined files. This command does not need information from design database, it can be extracted before generation of design data specification at API level.

Previously this command extracts information after generation of design data base. There is no point to generate result after database when information can be extracted before. For small design processing it is not making much difference but when design is large design database creation taking too much of time. Those designs consume more time for showing user defined files. Now in the framework this block of code which show user defined files is moved forward before design data base creation. By this way performance of the block improves. For this improvement performance analysis needs to be done for analyze time improvement in the flow.

For the previous design flow which have specific block after design data base creation, analysis result is showing below [2]:



Performance Profile Index

For

Run on Tue Nov 22 18:59:34 2016
Reported on Tue Nov 22 19:00:40 2016

Profile of
statements and 1476837 subroutine calls in 555 source files and 141 string evals.

for 28.1s (of 30.8s), executing 4343814

Top 15 Subroutines

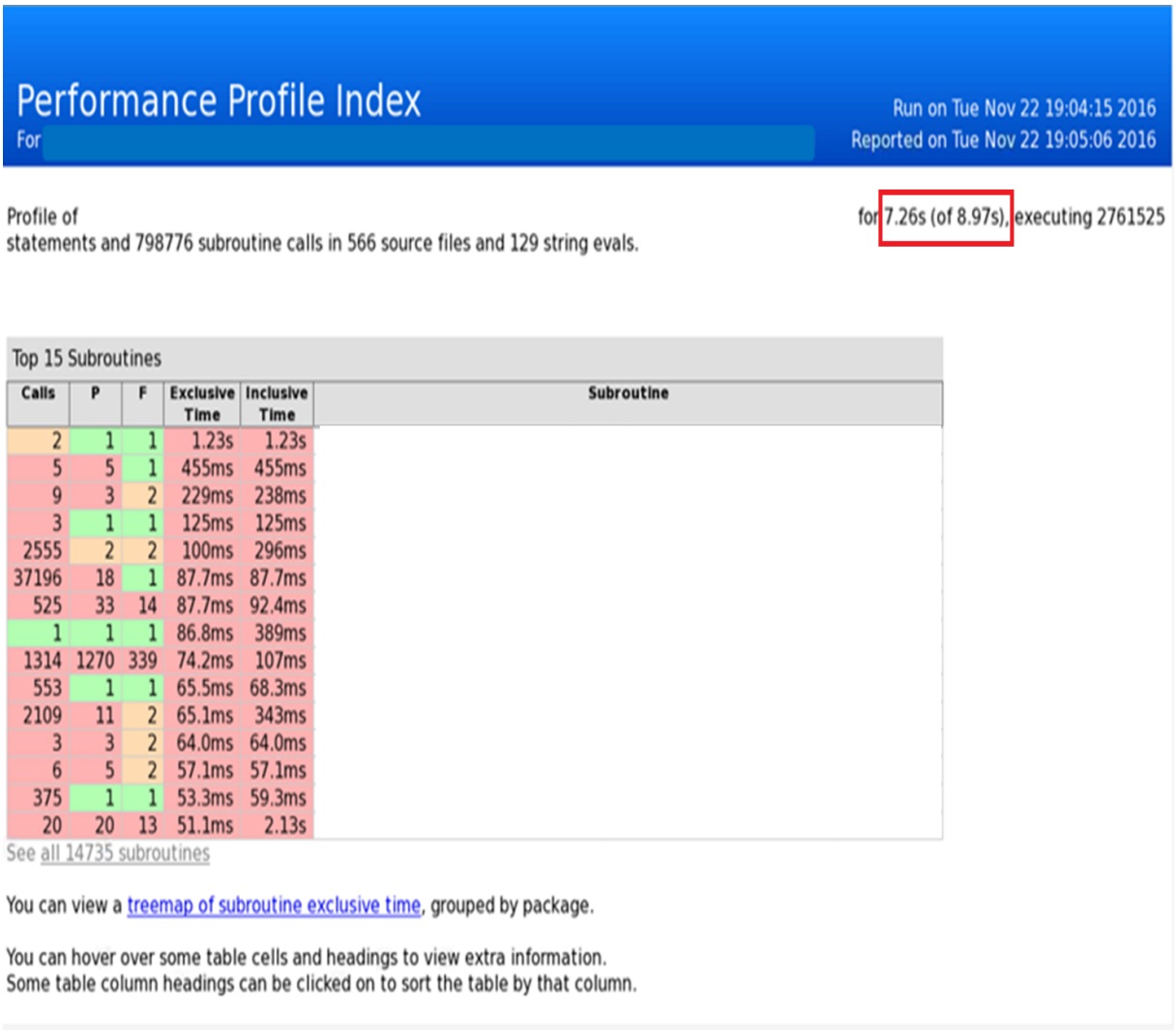
Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
41	1	1	16.9s	16.9s	
2	1	1	1.17s	1.17s	
6	6	2	481ms	481ms	
163345	36	1	384ms	384ms	
20642	3	2	307ms	961ms	
17341	3	2	264ms	482ms	
66	6	1	245ms	1.84s	
9	3	2	227ms	237ms	
47	2	2	224ms	420ms	
791	40	15	215ms	238ms	
3574	1	1	172ms	17.9s	
41847	5	1	163ms	163ms	
9143	8	2	154ms	503ms	
3175	10	1	142ms	1.24s	
3	1	1	138ms	138ms	

See all 14436 subroutines

You can view a treemap of subroutine exclusive time, grouped by package.

You can hover over some table cells and headings to view extra information.
Some table column headings can be clicked on to sort the table by that column.

Figure 6.2: Analysis report before changes



Performance Profile Index

For

Run on Tue Nov 22 19:04:15 2016
Reported on Tue Nov 22 19:05:06 2016

Profile of
statements and 798776 subroutine calls in 566 source files and 129 string evals.

for 7.26s (of 8.97s), executing 2761525

Top 15 Subroutines

Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
2	1	1	1.23s	1.23s	
5	5	1	455ms	455ms	
9	3	2	229ms	238ms	
3	1	1	125ms	125ms	
2555	2	2	100ms	296ms	
37196	18	1	87.7ms	87.7ms	
525	33	14	87.7ms	92.4ms	
1	1	1	86.8ms	389ms	
1314	1270	339	74.2ms	107ms	
553	1	1	65.5ms	68.3ms	
2109	11	2	65.1ms	343ms	
3	3	2	64.0ms	64.0ms	
6	5	2	57.1ms	57.1ms	
375	1	1	53.3ms	59.3ms	
20	20	13	51.1ms	2.13s	

See all 14735 subroutines

You can view a treemap of subroutine exclusive time, grouped by package.

You can hover over some table cells and headings to view extra information.
Some table column headings can be clicked on to sort the table by that column.

Figure 6.3: Analysis report after changes

After changes, moved code block before design database creation improvement in the performance is shown as above.

We can see from the report that there is one table which shows overview of the analysis. File name is mentioned on the top of the report. Total time for running that particular command is showing after file name. In the table number of calls, exclusive time, inclusive time and subroutines are mentioned.

- Exclusive Time : Best for bottom up approach.
  - Total time in the code consumed by subroutine.

  - Location where actually time spent in flow.
  - Suits for localized optimization.

- Inclusive Time : Best for Top Down Approach
  - Shows time spent in and below of subroutine.
  - Suits for structural optimization.

Second report shows an improvement in the result. For bigger design improvement is also bigger and performance of framework is increased. By this way performance of the framework can be analyzed by the tool.

6.4 Basic terminology of next generation validation framework

- **Library**

  The lowest level of design specification. A library definition is a set of design collateral (source files) and sub-libraries (defined next) and a build specification (compile options, tool options) that can be used to generate a set of results. The results can be anything from analyzed libraries to generated RTL/C/XML files, HDL spec, etc.) Libraries are immutable. A tool cannot be run until the library definition for that tool is known. Once known, it is immutable - it will not change.

- **Sub-Library**

  Enables merging/packaging multiple libraries into a single library.

- **Model**

  A model definition is a grouping of one or more libraries with a set of specific construction and execution options applied to it component for e.g.: Simula-

tion models would elaborate and simulate, static check models would elaborate and run checks.

- **Stage**

  A stage is the lowest level of granularity for accomplishing work in flow. It typically is a Perl module which provides a wrapper around a specific tool, which does following tasks:

  - Basic setup for the tool
  - Construct the tool command line based on the data model
  - Execute the tool command line
  - Post processing of the results/files
  - Log the STDOUT/STDERR
  - Report the execution status to the flow manager

  Dependencies can be specified at stage level (stage A depends on stage B). This is also currently the granularity at which incremental/caching behavior is supported All stage code writes to a shell script, which is then executed. The shell script will have everything needed (environment variable setup, commands, etc) to replicate its action. This allows for debugging single stages/-tools, without needing to know the sequence of stages that lead up to this point. Multiple instances of a stage can share the same stage definition. Each can be configured separately. Each stage does a discrete re-usable quantum of work. Stages communicate with the outside world through well-defined APIs. They will fetch their inputs through the well-define configuration engine API, and provide logging and error messaging outputs through other well-defined APIs. This allows stages to be re-used in other contexts, by re-mapping the APIs.

- **Flow**

  A flow is a named grouping of stages. Every stage must be a part of a flow.

Flows are hierarchical i.e. a flow could include other flows. In such cases, the included flow is called a subflow of the top level flow. Example: If flow A includes flow B, flow B is a subflow of flow A. A flow can have multiple instances of a stage or subflow each configured differently (diff options, netbatch resources, etc).

6.5 Need of run time analysis

According to the latest statistics next generation validation is used 87% in latest project and 13% by other projects. From the latest project during one month 1000 users approximately run flow approximately 99984 times.

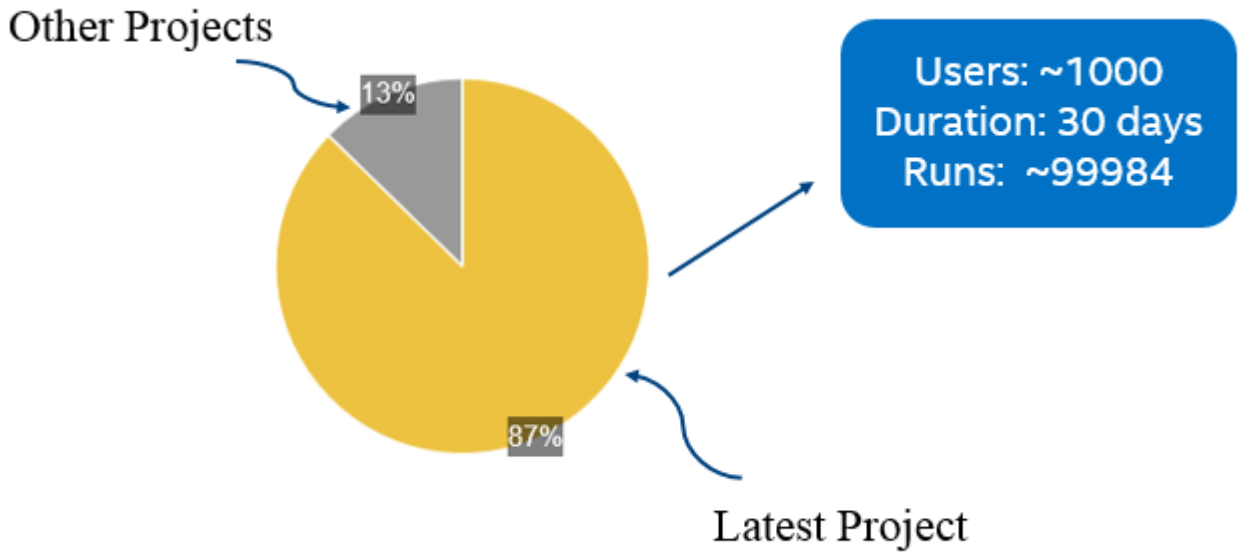


Figure 6.4: Usage of next generation design validation framework

As show in figure importance of run time is very critical factor. Small improvement in run time can cause lot more saving of time for all the projects. Flow run time analysis is important and it is done at stage level in the flow.

6.6 Run time analysis at Stage level

6.6.1 Initial results

Run time analysis is first done on next generation design validation framework which gives 24.42% more time consuming than previous generation design validation framework. Basically this analysis is done by nytprof profiling tool at stage level.

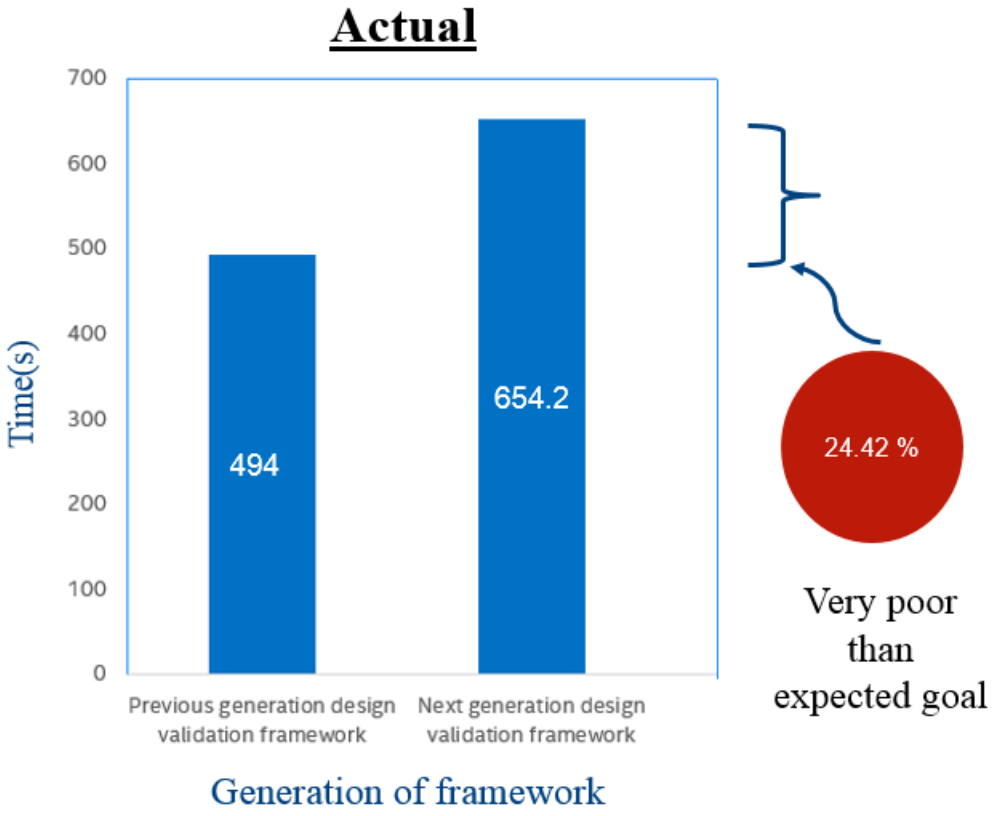


Figure 6.5: Initial profiling result for next generation design validation framework

In above figure graph shows statistics of time of next generation design validation framework with respect to previous generation design validation framework. In the bars seconds are mentioned which is total time to run that particular stage. As seen in initial result we can clearly see poor results of next generation design validation framework.

6.6.2 Expected results

Next generation design validation framework is suggested with some goals for improvement from previous generation design validation framework. Ultimate goal with new features is to make it's performance 2x. So 2 times faster than previous generation design validation framework which means that it should take half time than previous generation design validation framework. Expected results are shown as below:

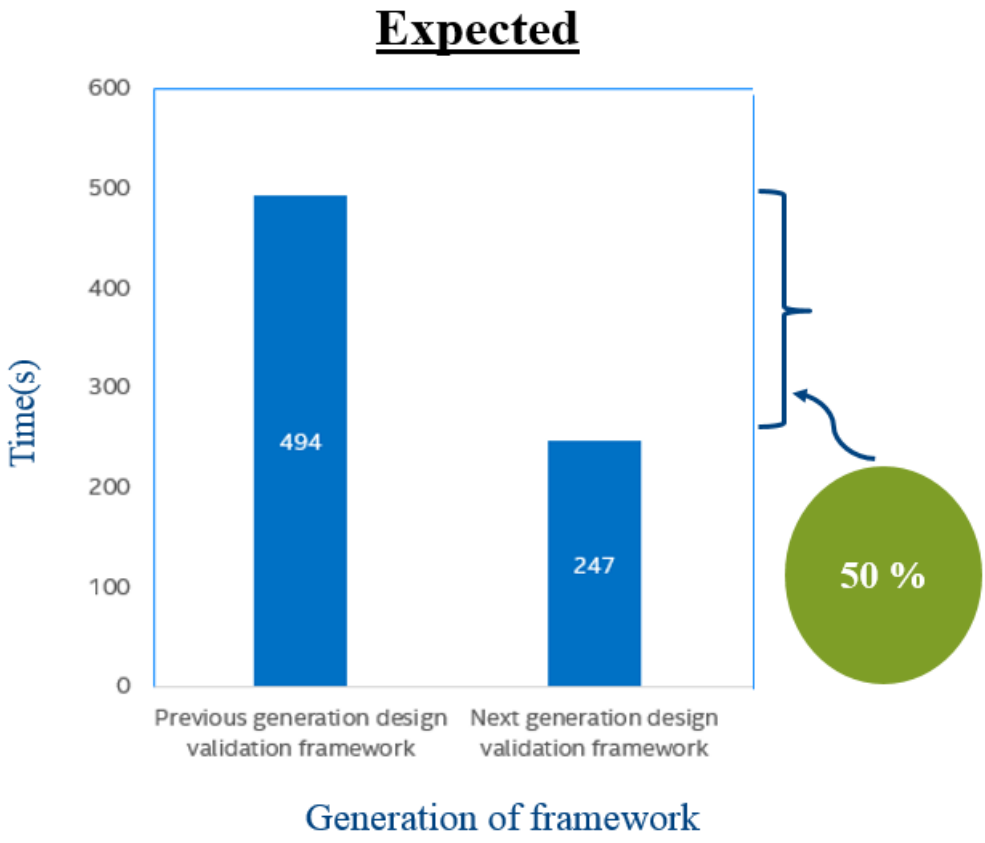


Figure 6.6: Expected profiling result for next generation design validation framework

As figure shows to meet to meet the actual goal time should be half than the previous generation design validation framework.To meet this result several methods need to apply for better performance in terms of run time.

6.7 Methods of improvement

To meet goal of next generation design validation framework in terms of run time, some methods of improvement are needed. These methods are derived from the profiling results. From nytprof profiling results, time taken by every part of the flow is analyzed and some methods of improvements are applied. Some part of the flow unnecessary taking more time than required. Filtering out those specific cases some suggestions are made. To verify these methods, all approaches are implemented on different IPs and verified.

Here is the list of methods suggested to improve run time of next generation design validation framework.

- Critical netbatch resource
- Incremental run
- Cache disabling
- Single library unfolding

6.7.1 Critical netbatch resources

In the next generation design validation framework, for better CPU utilization each and every job is scheduled to different machine. But while analysis cases came where dependent processes also run parallel and then it was taking long time to sync up between different machines. This sync up time increases run time for particular stage. To avoid such situations dependent processes filtered out and manage to run on same machine and only independent processes are scheduled on parallel machines. By this approach following result is achieved.

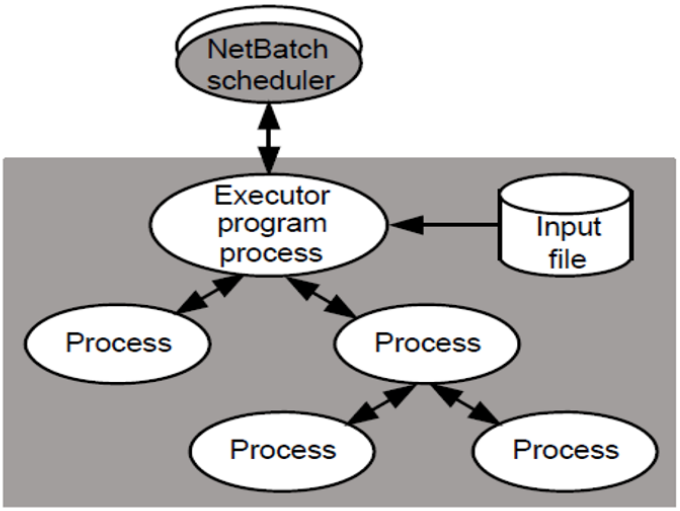


Figure 6.7: Critical netbatch resources

6.7.2 Incremental Run

Incremental run means same request done repeatedly. In next generation design validation framework request is coming from user though command line. Basically design data is passed through command line and then flow generates data needed for EDA tools. There are situations where users pass same data to build run without any change.

For this type of scenarios cache mechanism is provided in flow. Every successful run is saved to cache and when same data is requested by user without any incremental change then results generated previously can directly provide as output via cache.

As shown from figure compilation time can be saved through this approach so huge improvement can be achieved as in repeated request can handle quickly without wasting time and memory and that is 19.46% than the previous generation framework.

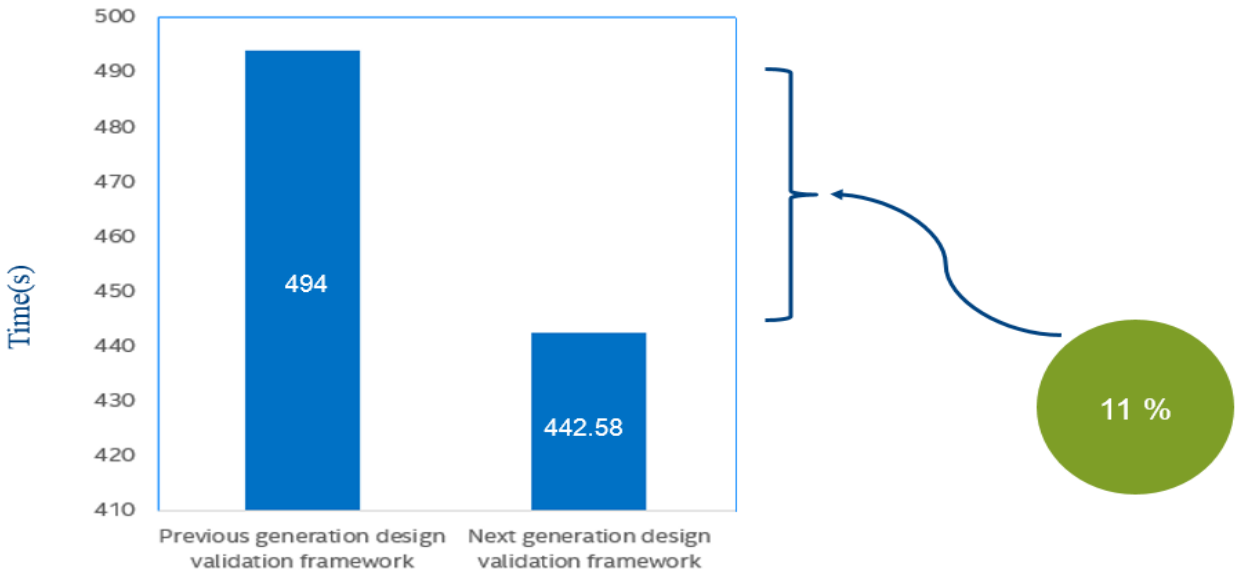


Figure 6.8: Results achieved by critical netbatch resource approach

6.7.3 Cache disabling

As described in the previous section cache mechanism is very important to save computations as well as time. But unfortunately this is not correct in all situations. In next generation design validation framework there are situations where retrieving data from the cache is taking more time than start a fresh run.

Challenging part is to identify special cases and compare actual fresh time and cache retrieval time for possible cases. After finding those special cases job is to disable cache from that part and that can be done by switch and make it off there otherwise by default caching mechanism is always on. By this approach 22% improvement in the result is seen.

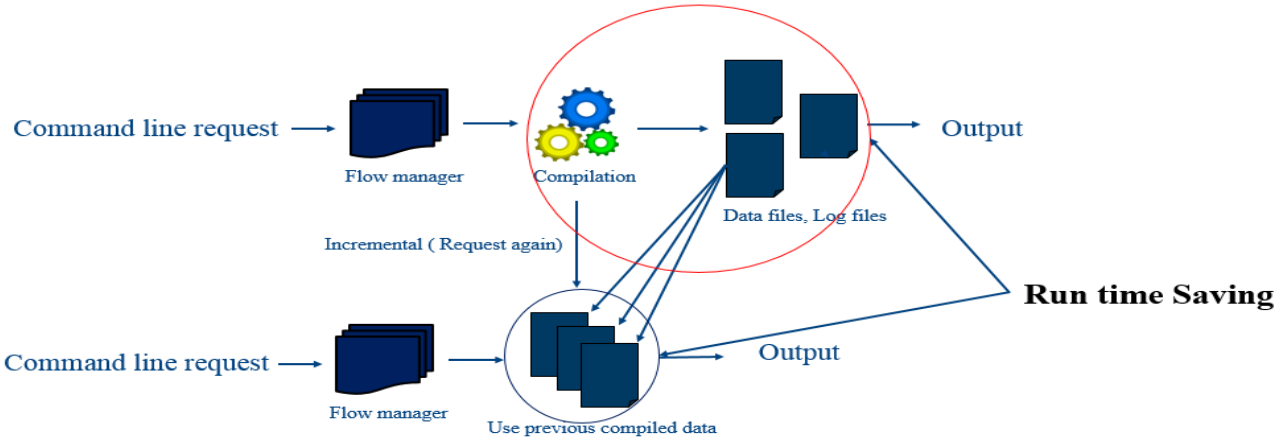


Figure 6.9: Incremental run method

6.7.4 Single library unfolding

Flow of next generation design validation framework starts with request from the user by command line. Once getting design request from the user all required data is started processing by flow manager. Smallest entity of design is library so all libraries are first initialized during this process. But is it really necessary? For many designs only few libraries are actually required from the data files. There is no need to process all libraries.

In such situations there is huge loss of time for compiling all libraries. To overcome this situation flow is not processing all libraries at compile time but only required libraries are getting processed at run time. This approach not only save time but also reduces machine's load when several requests are coming parallel. As shown in figure 5% improvement can be achieved.

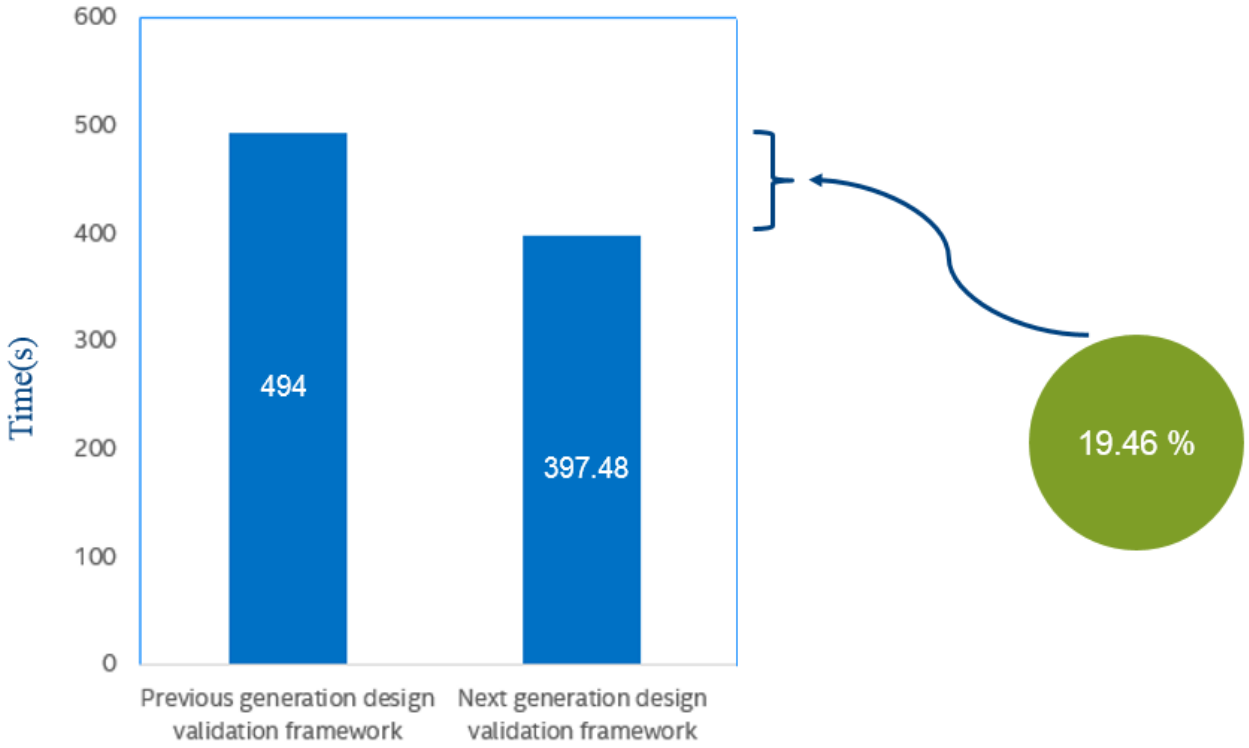


Figure 6.10: Results achieved by incremental run approach

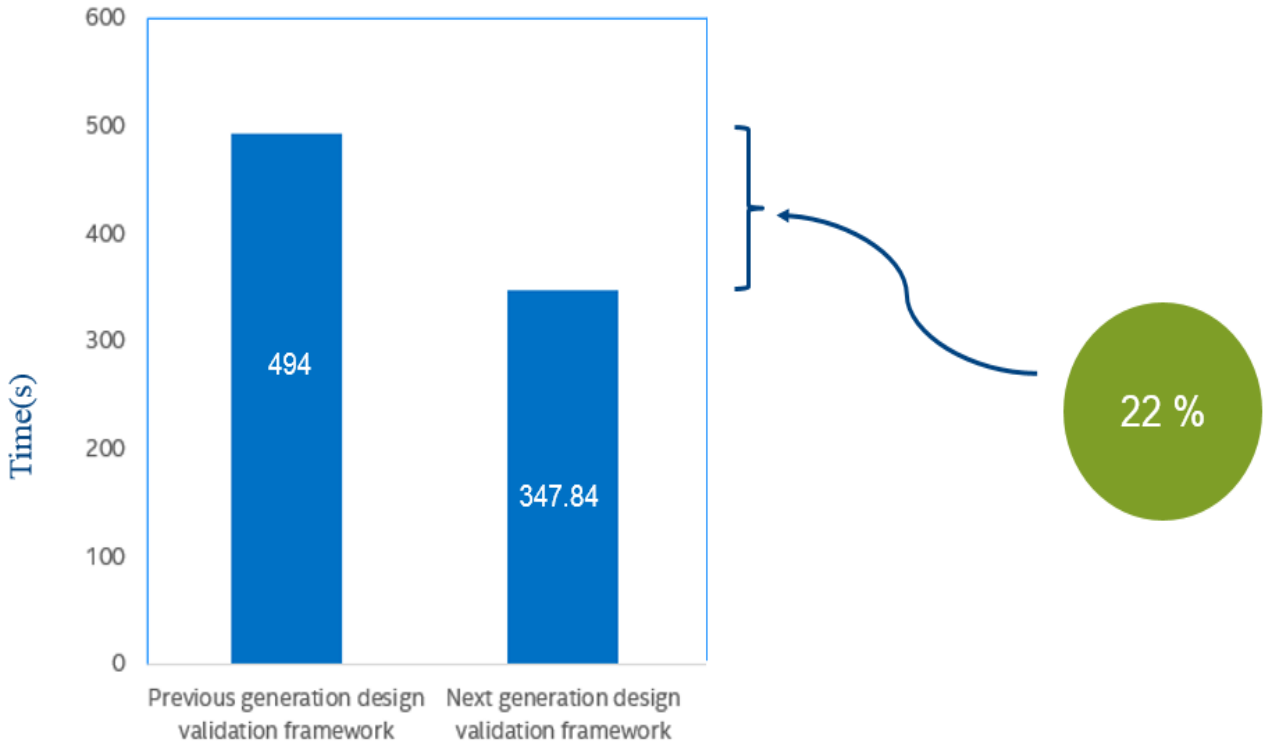


Figure 6.11: Results achieved by cache disabling approach

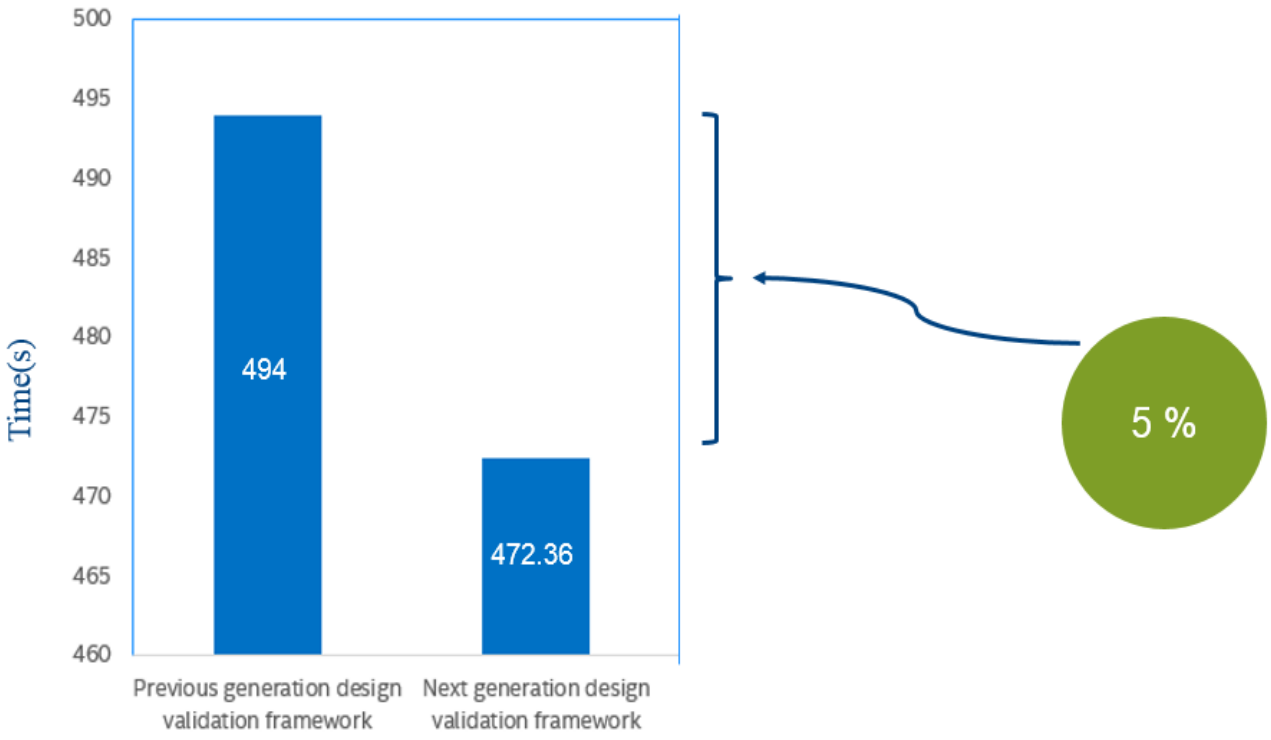


Figure 6.12: Results achieved by single library unfolding approach

6.8 Best result achieved by combining all methods

In previous section various methods are discussed with their results. All these methods need to combine to see overall result achieved. Goal of next generation validation framework is double the performance. Combining all methods 32.38% improvement is achieved. First result without implementing these methods is very poor as 24% bad than the previous generation. So from negative results to positive results which is very near to expected results.

To verify methods in general designs, solutions are applied on three different IPs and results are very positive. Specially for time critical projects this approach is useful. By this way overall run time analysis is done and improved in next generation design validation framework.

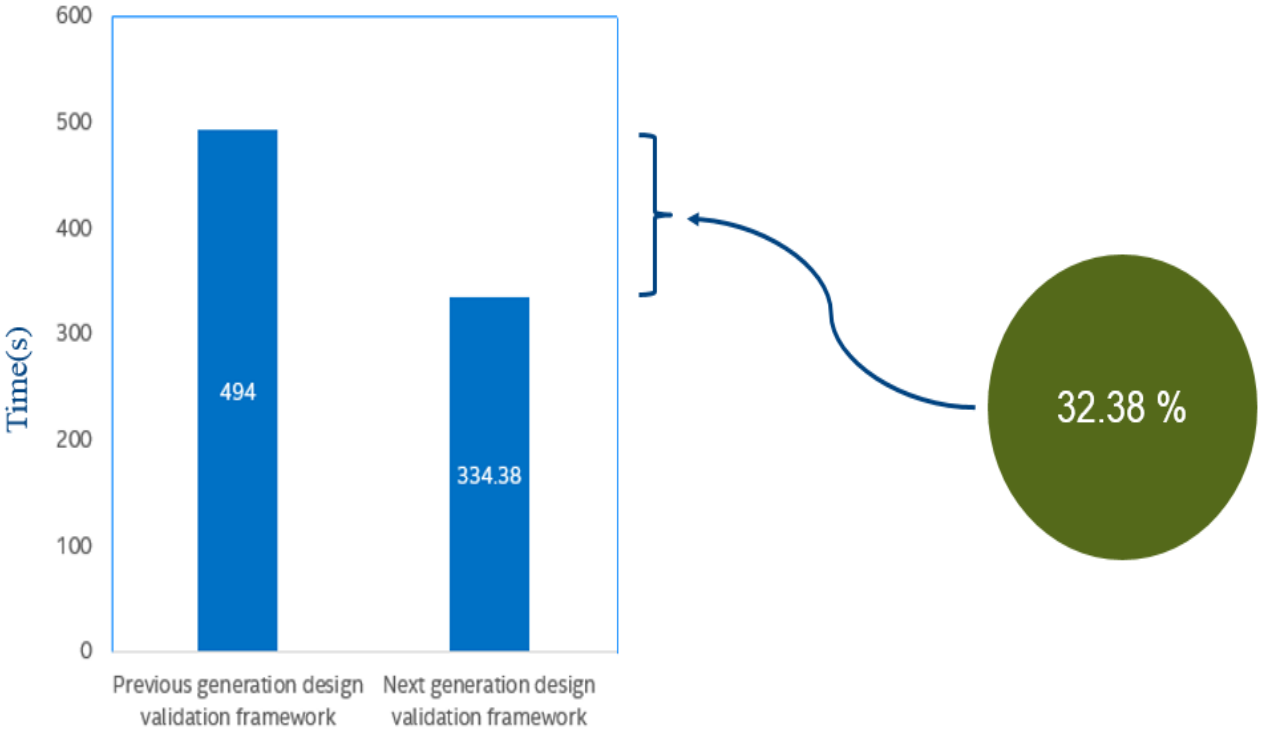


Figure 6.13: Results achieved by combining all methods

6.9 Summary

This chapter gives overall summary of performance analysis. Showing analysis for the block of framework and generate the results with nytprof perl profiling tool. All the fields generated in report are explained. For any change or improvement this is the best way to show overall performance improvement. Run time analysis is done at stage level where initial results were very poor. But after suggesting new methodologies, results are improved and 32.38% faster performance is achieved in new generation design validation framework than previous generation design validation framework.

Chapter 7

Testing in Perl

This chapter describes importance of testing as well as levels of testing in software. Here scripts are written in perl so rest of the chapter explains perl testing modules and methodologies.Improvement in the performance with testing is to save total time rather than testing after design development.

7.1 Test levels

Testing have different levels during the process. There are many methodologies included in levels of testing that is used during software testing. Main two levels of testing are:

- Functional Testing
- Non-functional Testing
- **Functional testing:** Functional testing can be seen as black-box testing which is based on the specifications of the software needs to be tested. Main aim of functional testing is to testing provided inputs and for that examination of result which tests functionality of the application intended for. This type testing is conducted on a complete integrated system to examine the system's

compliance with its specified requirements.There are specific steps for checking functionality as below [2]:

- Functionality of the application should determined which it intended for.
- Test data creation based on specification of the application.
- Output must be based on specification.
- Actual and expected results should be compared based on test cases.

- **Unit testing:** Before the setup handed over to software team this type of testing is performed by developers. Testing is performed on individual units or blocks of source code assigned areas. Test data used by developer team and quality assurance team are different.Main goal of this testing is to check correctness of each individual blocks of the design. But there are some limitations of unit testing. This testing can not catch each and every bug in tool. It's impossible to evaluate each and every part of software application. Always test scenarios are in limited number.

- **Integration testing:** As name suggest this testing is testing on combined parts of an application to test functionality of the design. It can be either bottom-up integration testing or top-down integration testing. [2]

  - Bottom up integration: Testing begins with unit testing followed by higher level of combinations of models.
  - Top down integration: First test is done on higher level modules and then come to lower level of modules.

  In normal scenario bottom up approach is taking first and then followed by other method. There are multiple tests for complete application and situations.

- **System testing:** System as a whole tested under system testing. To check system as a whole to meet its expected standards all components are tested once combined. Special design team is doing this type of testing. [2]

  - First step of software development life cycle includes software testing.
  - To meet functional and technical specifications this test is designed.
  - Location for testing is selected where it is possible to deployed.
  - Application architecture and business requirements both can be verified.

- **Regression testing:** When some change in application is done some other areas might affecting by it in functionality. This testing is done to verify that a fixed bug hasn't resulted in some other blocks or business rule violation. Purpose of testing is to verify effect of new changes in application is not faulty.[2]
  - On change of application gaps should be minimum.
  - Another area should not be affected by changes.
  - Risk should be minimum for application.
  - Coverage of test should be increased without compromising time lines.
  - Speed of market should be increased.

- **Non functional testing:** Testing is done on its non-functional attributes. Basically testing on attributes which are non functional in nature but important to test like performance, interface, user, security, etc.There are number of sub testing available within this like stress testing, performance testing, load testing etc. [2]

7.2 Unit testing

As we saw in the previous section unit testing is done by developers. It is tested before handed over to testing team. This one is applied on the individual units of the code to verify functionality. Perl supports all types of testing. Unit test is done in perl to test each and every part of the code to verify functionality. [2]

7.2.1 Perl modules for unit testing

There are different modules design in perl for unit testing. They all have their unique functionality and testing support. Some of them are as shown below:

- **Test::Simple** - Simplest type of module in perl is Test::Simple. It is one of the CPAN module which is used for writing basic tests. Further features are added in this module and make it separate module for testing with high functionality. Basic testing is done by 'ok' function. If test passes then it prints ok otherwise not ok. like [5]

  ```
  ok( $foo eq $bar, $name );
  ok( $foo eq $bar );
  ```

  Basically this is simple test to print ok or not ok for passing status of the test.For this test to work you must pre-declare number of test you plan to run script so that in case of some error it can loop out rather than continuous testing in loop.

- **Test::More** - Wide range of testing utilities are provided by this module. There are various ways to print "ok" with better diagnostics, skip tests in case of some fault, test features and comparison of complex data structures. Only ok is not provide enough support for testing. There are number of functions supported in this module. Print test name, ok , not ok , done_testing function when testing is completed, is and is_not for comparing variables or expressions, like, unlike for matching outputs, cmp_ok to test comparison is ok or not, can_ok for checking methods are supported or not, subtest, pass, fail etc. many functions are supported in this module. Some examples are as shown below[5]:

  ```
  ok( $foo eq $bar, $name );
  ok( $foo eq $bar );
  is  ( $got, $expected, $test_name );
  ```

```
isnt( $got, $expected, $test_name );
like( $got, qr/expected/, $test_name );
unlike( $got, qr/expected/, $test_name );
cmp_ok( $got, $op, $expected, $test_name );
can_ok($module, @methods);
my $obj = new_ok( $class );
subtest $name => \&code, @args;
pass($test_name);
fail($test_name);
require_ok($module);
```

- **Test::NoWarnings:** For any script in general it shouldn't produce warnings. This modules is used to capture warnings. It automatically adds one extra test which will run when script ends to check for checking that there were no warnings. In case there are any warnings, result produced is "not ok" and diagnostics of where, when and what the warning is for debugging. There are several functions supported by module described below.[6]

  - had_no_warnings : This function checks that warnings have been emitted by test scripts. It automatically calls self script for checking when script finishes.
  - clear_warnings : To clear the warnings generated by script this is used.If the array result is null then function had_no_warnings() will produce a pass result.
  - Warnings: This function will return all warnings captured in script. Element of array return information about the warning. Some of the methods used in warning are: $warn-¿getMessage to get message, $warn-¿getCarp to get stack crap, $warn-¿getTrace to get stack trace, $warn-¿getTest to get number of tests, $warn-¿getTestName to get test name.

7.3 Improve efficiency of existing test framework

7.3.1 Introduction

As mentioned in above section next generation framework is built it perl format and has standard test support through perl modules.Each and every feature needs to be tested for correctness of the framework. For this testing standard test framework is built in current flow to test each and every block of the flow. Developers generally add test in framework with every new feature. In test framework there are two types of testing methods present one is unit test and one is system test. Any new command line feature are tested by system framework and for that command line feature added modules are tested by unit testing.

7.3.2 Flaws in existing test framework

Test framework should be powerful and efficient as this is the way to validate working of the flow. Existing test framework have many features and tests added to test features of the flow but this is not 100% efficient. Some of the major flaws found in framework is as below:

Poor performance:

During analysis getting poor performance in run time factor of framework. Time is the major concern for the flow as each and every time when any user add new feature, it must have to pass through all tests again to ensure that new feature doesn't affect existing tests present in framework which is called regression testing. If regression testing is taking more time then for even small change developers have to wait and pass through all tests again which should not be done ideally. More over after local testing there are two phases where any change is tested with existing tests one is testing in local repository and one is when pushing changes in central repository. During all these processes run time is very big factor for the flow. Analysis confirms

that run time is very huge that by that time even small IP can be run on the framework. Ideally framework is the wrapper only and it should not take much time but which can not be seen in current scenario.

Missing corner test cases:

Ideally in any test framework each and every scenario should be covered.But during study seen that some of the cases not at all covered through existing framework. Framework is used by so many customers and they are passing flow with expectation of working all features. Now if there is something broken from customer point of view then it is very hard to find exact spot to looking at. This will take time and also take huge effort to resolve. To overcome this one and only one approach should apply as with adding every new feature or module there must be test condition for each and every possible case. That way we can make sure that whatever present in the flow is well tested and no more issues can come through testing point of view.

7.3.3 Case study of test framework for improvement

As mentioned in above section there are two main flaws present in framework. During analysis there are different cases encountered and different approaches made to overcome scenarios. All different cases mentioned below with problem statement and solution approach.

- **Case 1**:

  Problem encountered - In existing framework large number of test cases are present. Mostly test cases are added within flow when new feature is added by user to validate that particular feature. At time of adding new test case within framework, developer is not aware of the existence of similar kind of test in present framework. New test can be extension of existing test case. In that case test duplication is happened where same checks are present in more than two test cases. Run time can be major concern in this type of case.

Solution - As mentioned in the problem statement duplication is concern point. For this problem duplication should be removed by combining similar type of test cases. During study many test cases encountered where some of the test cases can be combined. All the tests are analyzed by their purpose through release notes. After study there are several test cases are there which have same first phase of testing. Those type of test cases are combined and reduced run time as common testing time is less than run on two different test cases. This way similar kind of test cases can be combined to reduce run time.

- **Case 2**:

  Problem encountered - There are several test cases which test specific test scenarios like generation of log file, stage passing conditions, all input files are passed correctly or not. In this type of scenarios only several conditions need to check. But in present test framework almost every test checks for whole flow generation and then extract data from that and flow generation takes major time.

  Solution - In scenarios like above we need to understand the purpose of test cases and checks applied on that test case. Only required data of checks should be generated rather then running whole flow. Framework have strong command line feature with tags. Test cases can use those switches to pass only limited flow data through it. This type of approach is used when user don't want to reduce number of test cases but still need of efficient test case is there. There are many test cases suffer from this type of conditions and this solution can be used there which helps to reduce run time drastically.

- **Case 3**:

  Problem encountered - Next generation design validation framework is basically wrapper around different EDA tools. In the framework majority runs are around stage level where each stage is one type of tool like static check tool, design tool, etc. Different models are design with different data and

checks like with library and checks for design data. Some models have all data and some have very basic and limited data which all are designed according to their need of usage. But this models are not used efficiently by test cases present in framework. Developers directly use biggest model with all facilities irrespective of their usage in test checks. Big model takes more time to build than smaller model which increase run time of test.

Solution: From looking at the flow of next generation design validation framework use of models should done very carefully and efficiently. Basically all models are designed for different purpose so they should be used accordingly. By smart use of models run time can be reduced where only required data can be generated through running specific stage only. By analyzing test purpose many test cases can be improved and make them more efficient.

- **Case 4**:

  Problem encountered - During testing in some of the cases where only command line checks are done, mostly passed on dummy data only to check working of particular test scenario. Those dummy data present in test case are not in optimized form. That is almost similar to real data. Passing test on that kind of data is taking more time just to test normal working of command from user.

  Solution: To solve above mentioned problem generation of dummy data should be made very careful. Required data have only specific data for all categories. So during running that test limited amount of data is getting processed and purpose of test can be full-filled by saving run time and use that test more efficiently. In framework 40% of tests are system test from which 50% of them uses dummy data to check functionality of the flow. All these tests can run efficiently and faster.

7.4 Summary

This chapter covers various test methodologies and levels for perl. Perl modules and their functionality which is supported to test any script. Testing is important concept and powerful test cases must be designed to check functionality of the application or design of system. Test framework is designed very well but still needs lot of improvement. Through case study all those scenarios are analyzed and improved in terms of run time which brings efficiency in test framework of flow.

Chapter 8

Conclusion and Future scope

In this chapter, conclusion of the project work is described.

8.1 Conclusion

Next generation design validation framework is a wrapper for the design automation tools. For quality gaps and lack of debugging information present in current design validation framework, help commands for debugging information and run time analysis is done. For the purpose of the framework performance improvement steps are discussed and overall 56% improvement is achieved. Help command for library information is one of the feature added for library information to user. Testing is as important as development so study of test cases is done and by applying one or more modules debugging can make more powerful. Testing at early stage can save time and cost for the design. Case study of test framework is done and solutions are discussed to reduce run time and increase overall efficiency. In this way debug mechanism and performance improvement brings overall quality improvement in framework.

8.2 Future scope

In the future of this project further improvements can be done. Testing is the biggest area where starting from improvement in existing framework to developing corner test cases. Other than that new performance methods can be suggested not only from run time point of view but also from machine load, efficiency point of view. Some debugging approaches are also improved and added to current generation validation framework. Improvements are on going thing and by this project next generation design validation framework can be used for more and more complex design with better usability for user and efficient computation for all data.

References

[1] Oudjida, A. K., D. Benamrouche, and M. Liem. "Front-end IP development: Basic know-how."Design & Technology of Integrated Systems in Nanoscale Era, 2007. DTIS. International Conference on. IEEE, 2007.

[2] Weidendorfer, Josef, and F. Zenith. "The KCachegrind Handbook."

[3] Framework for system design, validation and fast prototyping of multiprocessor system-on-chip: Applied to Telecommunication Systems from "Architecture and design of distributed embedded systems"by Bernd Kleinjohann

[4] Intel internal documents

[5] Netbatch manual by HP Nonstop.

[6] "Test::Moremodule ",$http : //search.cpan.org/ exodist/Test - Simple - 1.302049/lib/Test/More.pm$,Date: 11/09/2016, Time: 14:30

[7] "Doxygen Manual ", $http : //www.stack.nl/ dimitri/doxygen/manual/index.html$, Date:11/09/2016, Time:14:30