

# Highly Optimized 2-Step Design Validation and Integration System Delivery to a Multicore Project

## Major Project Report

*Submitted in partial fulfillment of the requirements  
for the degree of*

Master of Technology  
in  
Electronics & Communication Engineering  
(VLSI Design)

By

**Poornima Khullar**  
(14MECV24)



Electronics & Communication Engineering Branch  
Electrical Engineering Department  
Institute of Technology  
Nirma University  
Ahmedabad-382 481  
May 2016

# Highly Optimized 2-Step Design Validation and Integration System Delivery to a Multicore Project

## Major Project Report

*Submitted in partial fulfillment of the requirements  
for the degree of*

**Master of Technology  
in  
Electronics & Communication Engineering  
(VLSI Design)**

By

**Poornima Khullar**

(14MECV24)

Under the guidance of

External Project Guide:

**Mr. Sandip U Rajput**  
Component Design Engineer,  
Intel India Technology Pvt. Ltd.,  
Bangalore.

Internal Project Guide:

**Dr. N. M. Devashrayee**  
Professor, EC Department,  
Institute of Technology,  
Nirma University, Ahmedabad.



Electronics & Communication Engineering Branch  
Electrical Engineering Department  
Institute of Technology  
Nirma University  
Ahmedabad-382 481  
May 2016



## Certificate

This is to certify that the Major Project entitled “**Highly Optimized 2-Step Design Validation and Integration System Delivery to a Multicore Project**” submitted by **Poornima Khullar (14MECV24)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date:

Place: Ahmedabad

**Internal Guide**

**Program Coordinator**

**Dr. N. M. Devashrayee**  
(Professor EC ,VLSI)

**Dr. N. M. Devashrayee**  
(Professor EC ,VLSI)

**HOD**

**Director**

**Dr.P.N.Tekwani**  
(Head of EE Dept.)

**Dr.P.N.Tekwani**  
(Head of EE Dept.)

## Project Completion

This is to acknowledge that Project entitled “**Highly Optimized 2-Step Design Validation and Integration System Delivery to a Multicore Project**” submitted by **Poornima Khullar (14MECV24)**, towards partial fulfillment of requirements for the degree of Master of Technology in VLSI Design, Nirma University, Ahmedabad is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination.

for Intel Technology India Pvt. Ltd.,

**Project Manager**

**Mr. Venkatesh K Elayavalli**  
(Engineering TD Manager)

Date:

Place: Bangalore

## Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in VLSI Design at Nirma University and has not been submitted elsewhere for a degree.
2. Due acknowledgment has been made in the text to all other material used.

**- Poornima Khullar**  
**14MECV24**

## Disclaimer

“The content of this paper does not represent the technology, opinions, beliefs, or positions of Intel Technologies India Pvt. Ltd. Company, its employees, vendors, customers, or associates.”

## Acknowledgements

I take immense pleasure in thanking my Mentor Mr. Sandip U Rajput, Component Design Engineer, Mr. Venkatesh K Elayavalli, Engineering Manager Intel Technologies India Pvt. Ltd., Bangalore, for having permitted me to carry out this project work. Through-out the training, they had given me much valuable advice on project work which I am very lucky to benefit from.

I also wish to express my deep sense of gratitude to my guide Dr. N. M. Devashrayee, Institute of technology, Nirma University for being a source of inspiration and for timely guidance during the project.

I would like to express my gratitude and sincere thanks to our Director Dr. P. N. Tekwani, Head of Electrical Engineering Department for allowing me to undertake this thesis work and for his guidelines during the review process.

I would like to express my gratitude and sincere thanks to Mrs. Ruttika Jaju, Component Design Engineer, Intel technologies India Pvt. Ltd., for guiding me throughout my thesis work at Intel, and for making project at Intel more enjoyable.

I wish to thank my classmates for their delightful company which kept me in good humor throughout the journey. Last, but not the least, there are no words to thank my family for their constant support and sacrifices, because of which I'm able to complete the M.Tech. degree successfully.

**- Poornima Khullar**  
**14MECV24**

## Abstract

With the rapid growth in semiconductor technology, always in alignment with Moore's prediction, the designers face large integration capacity than they can consume. Furthermore, the cut throat competition in electronic innovation is constraining the designers more and more to minimise the time and meet the "time to-market" window. Without change and evolution in the design process and testing process, it will not be able to simply reduce reasonable amount of time. One possible methodology to reduce this productivity problem is the "design-reuse", which consists in the re-exploitation of the already designed modules in different context, well-known under the name of IP-reuse. In the concept of IP reuse, the IP module is once compiled individually and then with the entire SoC modules, thus it is being validated twice, leading to the concept of 2-step compilation. This report details about compilation flow used for front end design and verification through which time required and the complexity of front end process can be reduced.

In Intel earlier different design teams used their own legacy flows. A new flow was proposed which was a mixture of all the flows and can be used many design teams. Advanced Streamlined System (AS-2) is a mixture of the legacy flows used for validation of the SoC or design modules. It also provides a common environment to many projects while providing project-specific customization. Flows that were used earlier for validation of the IP's were not compatible with today's generation SOC's. Advanced Streamlined System is converged, 2-step but it has performance gap and integration bottlenecks. Performance analysis of the current system is done to reduce the time for validation in best possible ways and finally flow migration to the new highly optimized system.



# Contents

Certificate	i
Certificate(Intel)	ii
Declaration	iii
Disclaimer	iv
Acknowledgements	v
Abstract	vi
List of Tables	ix
List of Figures	x
List of Abbreviation	1
<b>1 Introduction</b>	<b>2</b>
1.1 Motivation . . . . .	2
1.2 Problem Statement . . . . .	3
1.3 Thesis Organization . . . . .	3
1.4 VLSI Design Flow . . . . .	3
1.5 Front End VLSI Flow . . . . .	5
<b>2 Literature Review</b>	<b>7</b>
2.1 Behavioral v/s Structural Modeling . . . . .	7
2.2 2-Step Compilation . . . . .	8
2.3 Legacy and SoC Integration Flow . . . . .	9
2.3.1 Legacy Flow . . . . .	10
2.3.2 Advanced flow or SoC Integration Flow . . . . .	10
2.4 Advanced Streamlined System (AS-2) . . . . .	11
2.4.1 Flow Mechanism . . . . .	11
2.4.2 Integration Bottlenecks . . . . .	12

2.4.3	Summary . . . . .	13
<b>3</b>	<b>Performance Analysis</b>	<b>14</b>
3.1	Importance of Performance Analysis . . . . .	14
3.2	Perl Profiling Tool – NYTProf . . . . .	15
3.2.1	Statement Profiler . . . . .	15
3.2.2	Subroutine Profiler . . . . .	16
3.3	Performance Analysis of AS-2 . . . . .	16
3.3.1	Experiments . . . . .	16
3.3.2	Critical Observation in AS-2 . . . . .	22
<b>4</b>	<b>Design Prototyping and Results</b>	<b>24</b>
4.1	Importance of AS-2 Prototype . . . . .	24
4.2	Perl Testing Module . . . . .	26
4.3	Test Cases added to AS-2 Prototype . . . . .	27
4.3.1	Test for loading various checks for the SoC or IP . . . . .	28
4.3.2	Test to check attributes being passed from parent to child design configuration for the SoC or IP . . . . .	30
4.3.3	Test for Flow Equivalence Checking for the SoC or IP . . . . .	31
4.3.4	Test for checking whether AS-2 modules are Perl tidy compatible . . . . .	31
4.3.5	Test for Performance Checking of AS-2 prototype . . . . .	33
4.3.6	Improvement in Test . . . . .	34
4.3.7	Unit Test for Environment setup for AS-2 Flow . . . . .	37
4.4	Summary . . . . .	39
	<b>Conclusion</b>	<b>40</b>
	<b>References</b>	<b>41</b>

# List of Tables

2.1 Flow Comparison Table . . . . . 11

3.1 Results of Performance Analysis . . . . . 20

# List of Figures

1.1	VLSI Design Flow . . . . .	4
1.2	Front End VLSI Design Flow . . . . .	6
2.1	Behavioral Modeling Example . . . . .	7
2.2	Structural Modeling Example . . . . .	8
2.3	IP Classification . . . . .	9
2.4	Implementation of AS-2 . . . . .	10
2.5	Flow of AS-2 . . . . .	12
2.6	Integration Flow of AS-2 . . . . .	13
3.1	Statement Profile Example . . . . .	15
3.2	Experiment 1A . . . . .	17
3.3	Flow Graph of Experiment 1A . . . . .	17
3.4	Experiment 1B . . . . .	18
3.5	Experiment 1C . . . . .	18
3.6	Experiment 1D . . . . .	19
3.7	Experiment 1E . . . . .	20
3.8	Reoccurring Loop . . . . .	21
3.9	Experiment 2A . . . . .	21
3.10	Experiment 2B . . . . .	22
3.11	Color Coding in NYTProf . . . . .	23
3.12	Call Tree in NYTProf . . . . .	23
4.1	AS-2 Work Flow with IP . . . . .	25
4.2	AS-2 Work Flow with Prototype . . . . .	26
4.3	Test Driven Development . . . . .	28
4.4	Test for loading checks on SoC 1 . . . . .	29
4.5	Test for loading checks on SoC 2 . . . . .	29
4.6	Test for checking attributes . . . . .	30
4.7	Test for Comparing Flow Specification . . . . .	31
4.8	Before Perltidy . . . . .	32
4.9	After Perltidy . . . . .	32

---

4.10 Test for Performance Checking of AS-2 prototype[1] . . . . . 33  
4.11 Test for Performance Checking of AS-2 prototype[2] . . . . . 33  
4.12 Test for Performance Checking of AS-2 prototype[3] . . . . . 34  
4.13 Logging of AS-2 Flow test . . . . . 35  
4.14 Function of Screen Filter . . . . . 36  
4.15 Function of Unit Test . . . . . 38

# List of Abbreviation

RTL	Register Transfer Logic
SoC	System on Chip
CAD	Computer Aided Design
HDL	Hardware Description Language
AS-2	Advanced Streamlined System
FEDV	Front End Design Validation
PERL	Practical Extraction and Report Language
EDA	Electronic Design Automation
DUT	Design Under Test

# Chapter 1

## Introduction

All practical VLSI circuit are very, very large or very, very complex in nature like a processor, which are large and very complex applications. A complex circuit may have hundreds of inputs, outputs and millions of gates, in order to create complex circuit consisting of millions of transistors manually, can be difficult. For such problem a CAD tool is required which will help to automate the design for given specification.

CAD tools provide several advantages like it has an ability to assess complex conditions in which solving a single problem which may creates other problems. It use analytical methods to evaluate the cost of a decision and uses synthesis methods to provide a solution for the same. It concede the process of introducing and investigating the solutions at the same time.

### 1.1 Motivation

The entire project covers working on AS-2 which is a front end design IP validation system. It is converged, 2-step, and based on next generation build flows. It is a common solution used for both big and small IP. This project gives a good idea of methodology used in front end IP validation. It involves development and enhancing features required in the front end part of complex SoC integration. It reduces the design time, time to market and can handle design complexity easily.

Performance analysis of the current system is done to find the integration bottlenecks and enhancement can be done accordingly. With that developing a prototype for AS-2 as it was running on an IP which runs maximum qualification checks but not all. The prototype would imitate the design and contain test case for different IP blocks and perform all qualification checks.

## 1.2 Problem Statement

Designing a module which can be an SOC or IP that are optimized along the three axis of area, power and time is a difficult task. Achieving optimum balance along any two axis is a simpler task than across the three. To handle the rising time to market pressure and increasing design complexities we need efficient validation methodologies that can give maximum coverage check of the design in less amount of time. AS-2 is very efficient in the concept of IP reuse and compiling an entire soft SoC design.

But AS-2 has performance gap and integration bottlenecks which hinders the goal of reduction in time drastically. Performance analysis of AS-2 reveals the various gaps which when removed can enhance performance and finally moving to optimized AS-2 which would be much faster n efficient.

## 1.3 Thesis Organization

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

- \* Chapter 1: This chapter gives information about VLSI flow and the front end flow used in SoC design process.
- \* Chapter 2: It describes the various legacy flow that are used for soft IP validation.
- \* Chapter 3: This chapter explains the performance analysis results of the AS-2 system in detail along with its advantages and fault.
- \* Chapter 4: Implementation of Prototype which is used to save the compile time of the design.
- \* Chapter 5: Conclusion

## 1.4 VLSI Design Flow

The design process of any integrated circuit at various levels is evolutionary in nature. The VLSI design flow generalizes the step, which in itself are complex task and require various tools at different design level. The VLSI design flow is partitioned into two parts: Frontend design flow and Backend design flow. With the two flows together, a functional chip could be created from scratch to fabrication.[5]

Specification describes abstractly the functionality, architecture, and the interface of the digital IC circuit to be designed. Here, vendors get feedback from potential customers on their requirement. When this is completed, a final specification sheet with all major technical details is constructed. Behavioral description is then build to evaluate the design



in terms of performance, functionality, compliance to given standards, and other needs specified. RTL description is done using HDLs. RTL level analysis is then reformed to a gate-level netlist with the help of logic synthesis tools. A gate-level netlist is a description of the design in terms of gates and connections between them, which fulfils the timing, power and area specifications. These parameters of timing are checked by performing static time analysis. Finally a physical layout is created, which will be verified and validated and then sent to fabrication or production.

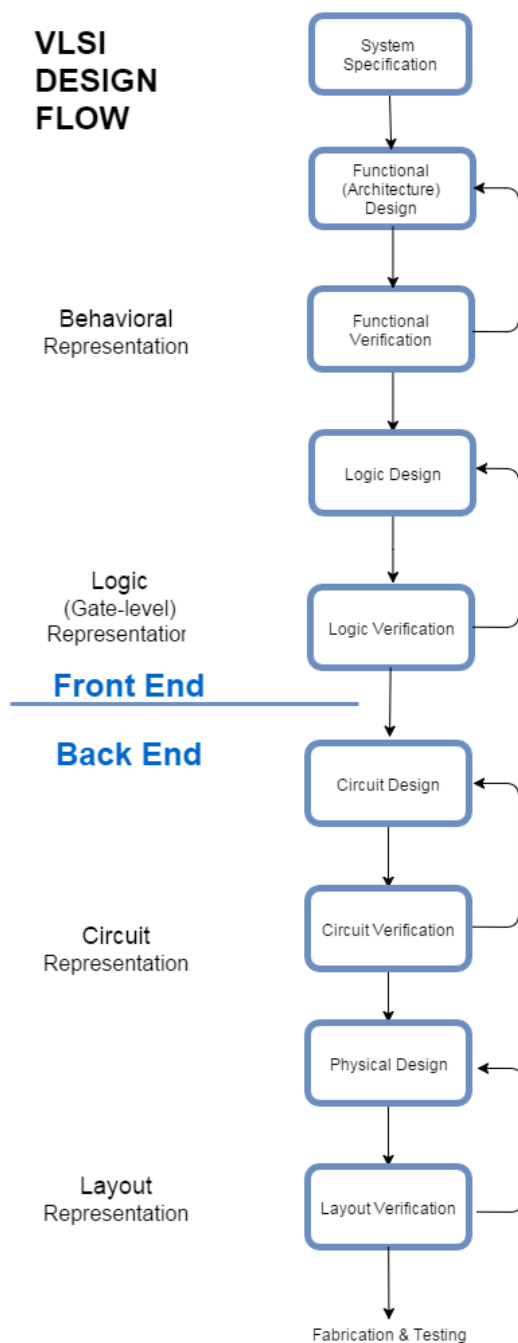


Figure 1.1: VLSI Design Flow

## 1.5 Front End VLSI Flow

The frontend flow obtain a solution for a user problem or it converts specification into an RTL circuit description. The flow starts from specification, and mostly involves verification at each step to achieve good initial design confidence. Later on the process is handled over to back end flow, where a number of process required for physical implementation of design are carried out.(Figure 1.2)

The design complexities are growing day by day so a single tool is unable to provide good remedy for all the design, compilation and verification related issues. With increasing System-on-Chip (SoC) complexity, the software content combined with it and the rising time-to-market pressures are increasing the need for an advance automated system for design, compilation and verification solution which saves time as well as efforts, during the designing period. The term different VLSI flows means the different design, compilation verification flows of tools by different CAD tool provider companies.[5]

The AS-2 gives higher degree of design confidence and bringing down the risk of re-spin or repeated efforts is the main aim of the system, which ultimately contributes for less time to market. The system is:-

- \* Converged: Used by almost all front end flows.
- \* Amalgam: It makes design compilation and validation of SoC easier and faster and takes all the advantages of the earlier used legacy flow.
- \* Streamlined: It is a new generation build flow compatible with today's SoC and IP block validation.

My work goes around the different periphery of the AS-2, performance analysis of AS-2 and enhancing features of the system.

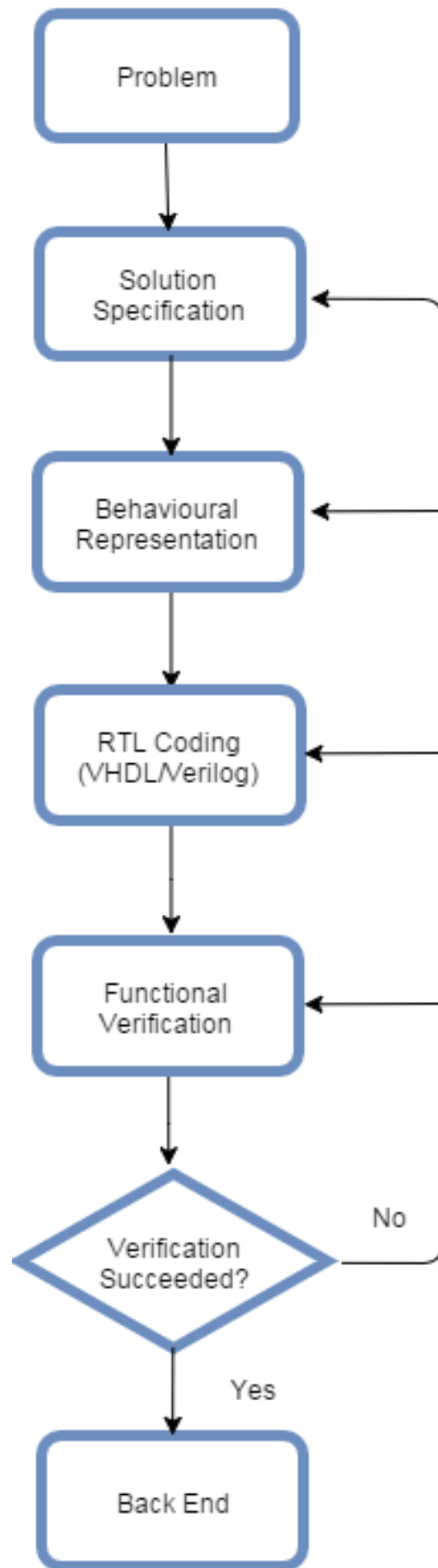


Figure 1.2: Front End VLSI Design Flow

# Chapter 2

## Literature Review

One possible methodology to increase productivity of design is to accept the "design-reuse" concept, which require re-exploitation of the already designed modules in different context. This idea meant building up new IP or modules using the existing ones. This results in a new concept well-known under the name of IP-reuse. Moreover, an IP is nothing other than a module implemented with reuse capabilities (IP = Module + Reuse Capabilities), which leads to the concept of 2-step compilation.[2].

### 2.1 Behavioral v/s Structural Modeling

When you are modeling hardware, you can write code in different ways, independent on the HDL you are using (i.e., either VHDL or Verilog). Behavioral modeling refers to a way to write code (more precisely, to model your hardware design) based on its functionality: it's like writing the algorithm that solves your problem. Certain disadvantages of behavioral modeling are that it does not support the concept of component re-usability. The entire design is loaded, elaborated and simulated at the same time which takes large amount of compile time hence increasing time to market. Moreover it gives a non-modular design description which is low quality and not optimized.

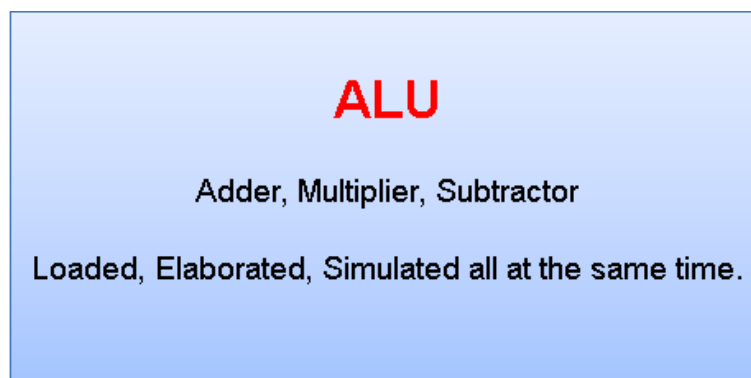


Figure 2.1: Behavioral Modeling Example

With structural code, on the other hand, you are connecting different parts together to get the final design. In some way you will generally use a mixture of the two: if you think to bottom-up approach, you first create behavioral/algorithmic code for lowest-level blocks (e.g., flip-flop) and once you move up across the abstraction levels you mix different blocks together and connect them in a structural code (e.g., shift register). Structural modeling gives modular design description and support the concept of design re-usability. It's a well-known fact that it's easier to optimize small design rather than big ones. As it's a modular design technique optimized small modules that can be inherited in larger modules, thus giving a more optimized overall design and brings the concept of 2-step compilation.

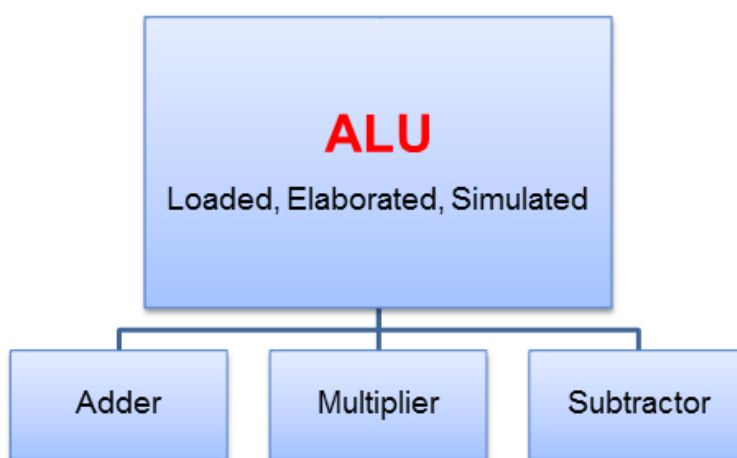


Figure 2.2: Structural Modeling Example

## 2.2 2-Step Compilation

In the concept of IP reuse, the IP module is once compiled individually and then linked with entire SoC modules, thus it is validated twice leading to the concept of 2-step compilation. The 2-step support parallelization of process and linking easily with SoC easier. Here the IP reuse is only for the RTL collateral not for the Validation collateral. Moreover, to be reusable, a module must be:-

- \* Configurable- constructed to solve a general user problem.
- \* Portable- designed independent of technology and CAD tools.
- \* Debugable- verified and validated with a high degree of confidence "bug free".
- \* Readable- clearly documented on the basis of applicability, restrictions, and defined interfaces.

An IP or module can be classified as a reusable module by these of quality:

- \* Functional- it is the lowest level of a module which can be reached to be able to use it.
- \* Maintainable- it means being functionally correct, a maintainable module which is well documented, with clean and commented coding.
- \* Reusable- A reusable module would have many improvement on a maintainable module or IP.

”For the purpose of module re-usability, an industrial corporation called VSIA (Virtual Socket Interfaces Alliance) was established in 1997. It’s aim is to create a set of standards for making IP reusable. These standards are well defined, described and accepted in today’s semiconductor world”.[2].

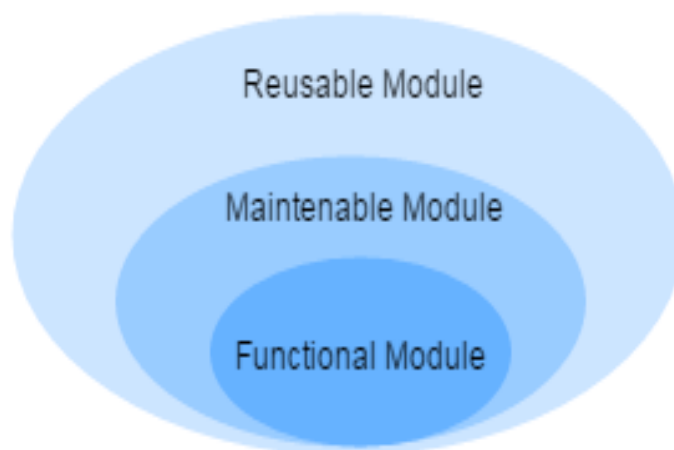


Figure 2.3: IP Classification

## 2.3 Legacy and SoC Integration Flow

Flows that are being used for validation of Soft IP’s are not compatible with today’s generation SoC’s and IP’s. The two flows that were used earlier are **Legacy Flow** and **SoC Integration Flow or Advanced Flow**. Both had certain advantages and disadvantages, but the common disadvantage of them was the time taken to compile the entire SoC design which was large, thus increasing time to market.

### 2.3.1 Legacy Flow

It worked on the concept of behavioral modeling design description (Figure 2.1) i.e. non-modular design description. It didn't supported the concept of IP reuse and thus no 2-step compilation. As the design is not optimized thus limiting the quality of the checks done on it. The compilation time was very high thus limiting the performance of the flow.[6].

### 2.3.2 Advanced flow or SoC Integration Flow

It supported modular design concept, IP reuse, saved design time also but didn't had a proper qualification step. It has high maintenance cost and difficult to deploy or proliferate to different segment of design system. If the flow failed on certain checks it was difficult to debug the cause of failure. It was not manageable either and cannot be extended to the growing technology because of its incompatibility.[6]

Solution to the above problem is AS-2 which takes all the advantages of the above flows and reduces the compilation time drastically thus reducing time to market and increasing the performance drastically

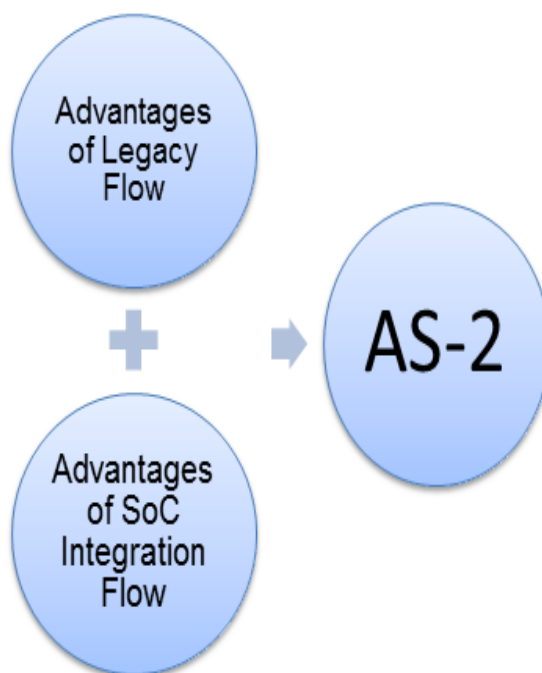


Figure 2.4: Implementation of AS-2

Table 2.1: Flow Comparison Table

S.no.	Legacy Flow	SoC Integration Flow	AS-2
1.	Behavioral Modelling	Partial Structural Modelling	Structural Modelling
2.	No IP Reuse (Cannot be extended to new technology)	IP reuse possible(not completely)	IP reuse possible (completely)
3.	Suitable for small designs	Not scalable for larger designs	Scalable for high capacity designs
4.	No 2-Step Compilation	Flow not modular, thus not easy to proliferate	Enabled 2-Step Compilation (For new technology)
5.	Performance Limiter (Slower)	Performance Limiter (Slower)	Performance Limiter (Slower)

## 2.4 Advanced Streamlined System (AS-2)

AS-2 is a mixture of both legacy flow and SoC integration flow. Taking advantages of both flows and having some new features of itself, it reduces the compile time drastically. It also support the concept of IP reuse and thus it is converged, does 2-step compilation and is much more advanced as compared to other system. AS-2 is facilitated both for small as well as complex IP's. As AS-2 is a mixture of various legacy flow it supports all earlier flow used by different design system at Intel.

### 2.4.1 Flow Mechanism

The various step of hybrid flow are as following:-

- \* **User Specification Request** –This stage specifies what action the user wants to perform on the design. It can be running only a particular test, compiling a certain part of the design, compiling a certain library etc., using command line interface. Validation of the user commands is also done in this step.
- \* **Data Extraction** – Based on the user request the AS-2 decided the input configuration, the design data which would be multiple RTL source code, library needed to compile for the design and the constraints given by the user and according to the environment set. Design reuse in one of the feature of the system which is done in this step. A large numbers of algorithms are used in this step, which makes the data extraction easy and fast.
- \* **Compilation** - the actual sets of commands are executed on the design. Various checks and test cases are also deployed on the design.



- \* **Execution** - The output of the compiler is fed into the executor that generates the output log files which can be looked if anything fails.

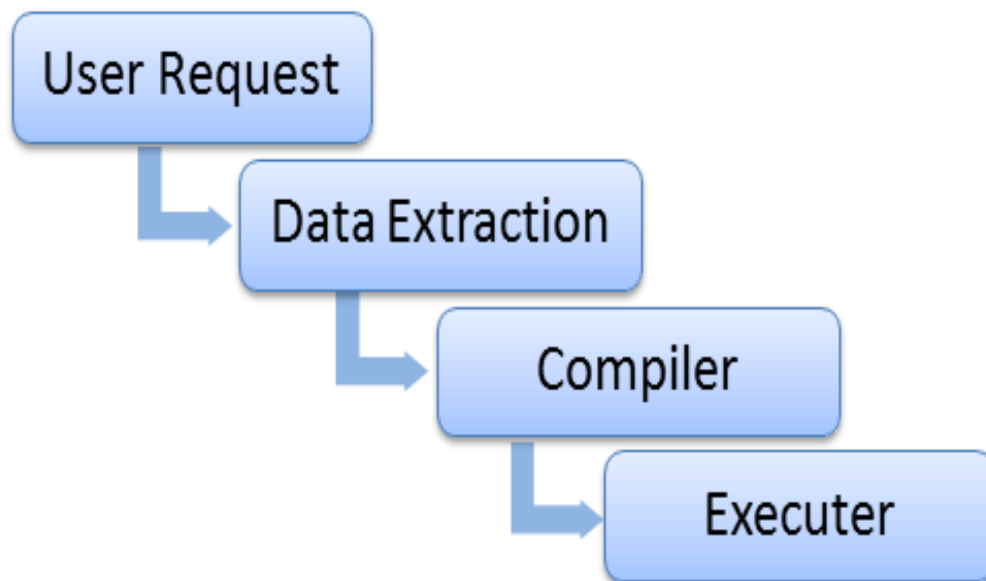


Figure 2.5: Flow of AS-2

### 2.4.2 Integration Bottlenecks

Besides having various advantages AS-2 still has some problem. It does not address the integration problem as per requirement. On the data extraction step of the flow mechanism it fails to pass data to different compiler. Moreover it produces large amount of data for analysis while running complex algorithm and are time consuming and hence increases time to market. It had no provision of flow testing and prototyping.

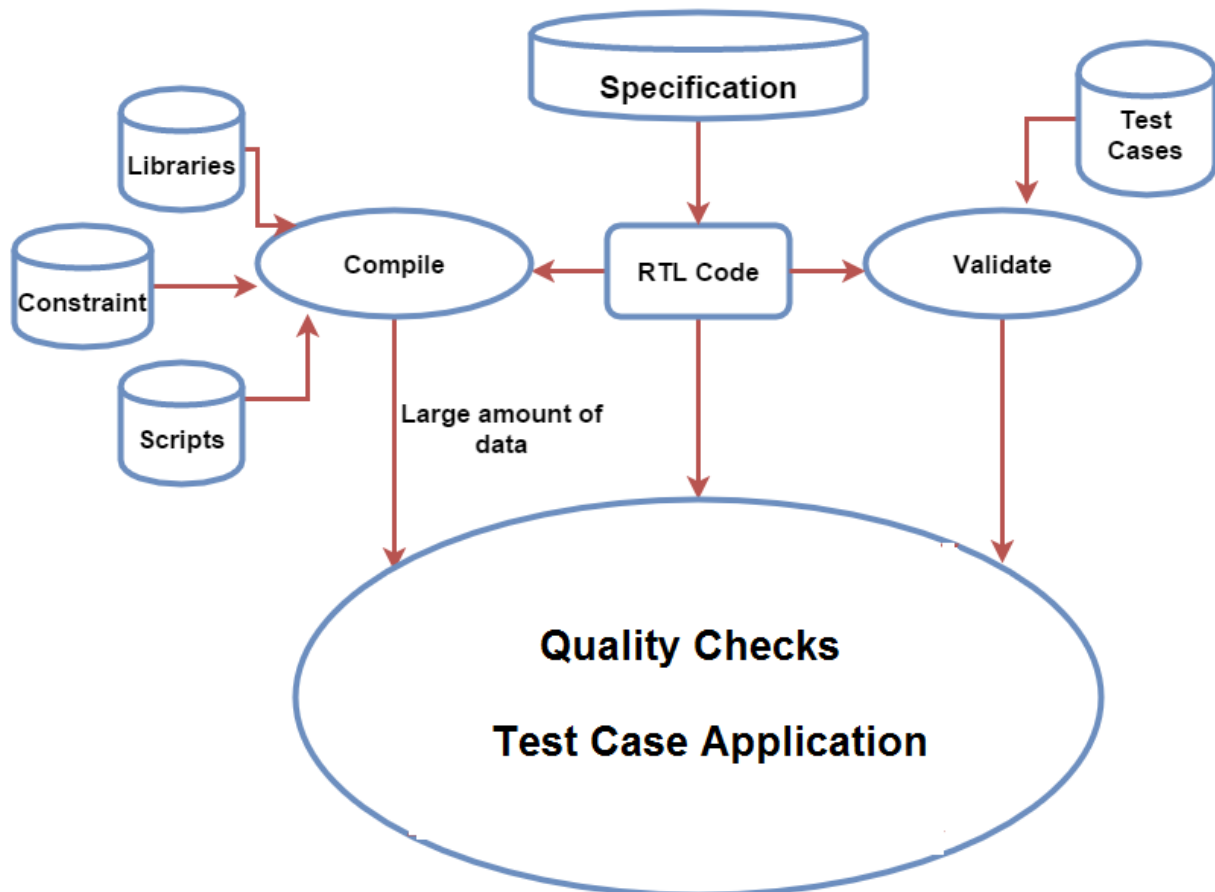


Figure 2.6: Integration Flow of AS-2

### 2.4.3 Summary

In order to find the performance and integration gap performance analysis of the current system needs to be done and finally migration to Optimized AS-2 system.

# Chapter 3

## Performance Analysis

Testing, verification and validation problems are generally consigned by separate communities - VLSI testing is generally assumed to belong to the Electrical Engineering domain, while verification and validation are assumed of as Computer Science subjects. While in industry, the methodologies deployed by testing, verification and validation engineers are also different. While the application area of these problems is surely not same, there is one similarity among all of them - and that is the underlying mathematical build or framework.

### 3.1 Importance of Performance Analysis

Performance analysis constitutes certain part of verification. Coverage is a measure to evaluate the progress of functional verification. This gives a clear picture on how well the design has been verified and also gives away the design corners still left uncovered. Code coverage and functional coverage are the different coverage methods used in functional verification. Besides having various advantages AS-2 still has some problem. It does not address the integration problem as per requirement.

- \* On the data extraction step of the flow mechanism it fails to pass data to different compiler.
- \* Moreover it produces large amount of data for analysis while running complex algorithm and are time consuming and hence increases time to market.
- \* It had no provision of flow testing and prototyping.
- \* Code quality checks are slower compared to other flow.
- \* While doing the analysis of the flow it had many recursive flows or flow hierarchy which need proper convergence to have proper dependent flow hierarchy.

The profiling analysis on the AS-2 is done by Perl profiling tool NYTProf which is being introduced in the next segment. It's a very accurate and efficient system.

## 3.2 Perl Profiling Tool – NYTProf

For sequential code, it is important to collect statistical data of the programs and certain run-time characteristic like time spent in each functions, number of times the function is called and code lines. This is called Profiling. The design or system is run under the control of a profiling tool, which at the run end gives the summary of an execution.[7].

- \* NYTProf is effectively does the work of two profilers. It acts both as statement profiler, and a subroutine profiler.
- \* It has resolution of nanosecond.
- \* It generates HTML report which included code coverage, statement flow and interactive Tree Maps. (Figure 3.12)
- \* NYTProf maintains extra information in the data file to capture each and every detail of the run that may be useful when evaluating the performance.
- \* It also records all the file-name and number of calls of all the subroutines.
- \* NYTProf can profile applications and system that fork between different processes, and does it with so efficiency so that there is no loss of performance. (Figure 3.3)
- \* `nytprofhtml` only works with a single output profile file. So in order to merge with multiple files use `nytprofmerge`.
- \* It also reveals the code coverage of entire flow part by part.

### 3.2.1 Statement Profiler

The statement profiler evaluates the time between entering one Perl statement and entering the next statement. It is similar to Statement coverage. The number of times statement is executed can be calculated to enclosing block and enclosing function.

```
while (<>)  
{ ...  
  1;  
}
```

Figure 3.1: Statement Profile Example

Looping for the first time around the loop, the more time spent analysing the condition would be recorded as the time spent on the last statement executed in the loop!

### 3.2.2 Subroutine Profiler

The subroutine profiler evaluates the time between entering a subroutine and leaving it. When the subroutine is called the call count is incremented and the duration is noted. Each time the subroutine is called, separate counts and durations are recorded for every location where the subroutine is called.

## 3.3 Performance Analysis of AS-2

- \* Step 1- The first step to carry out performance analysis of AS-2 is to deploy it on some design or IP Block, while running NYTProf on it simultaneously.
- \* Step 2- The NYTProf run gives a number of nytprof.out files which need to be analyzed further. The nytprof.out file is read by nytprofhtml. If there is more than one nytprof.out file we could use nytprofmerge. Next step is thus analysis.
- \* Step 3-During the analysis we need to track Master Violation that consumes large amount of time during compile.
- \* Step 4- Look the code of **Master Violation**. Reproduce the same violation using a test case in prototype and re-run the flow.
- \* Step 5- Optimize the flow part and re-run to see the reduction in time. The optimization can be anything either reducing number of calls or removing a reoccurring loop.
- \* Step 6 – Analyze the HTML report again. Track, look and optimize the flow. Repeat the above to get maximum reduction in time.

The branch taking large amount of time should be tracked down and unnecessary code should be optimized. For the optimization in the actual flow, some experiments were done locally before applying to flow. While optimization we need to make sure the functionality is completely restored, like a mistake that generally done when the actual purpose of keyword is not known, eg. : Croak can be misunderstood by print but both have different meaning. Experiments done are as follow.

### 3.3.1 Experiments

From a given design file, I have to extract given tags that is given as input by the user and display the information for the user input in optimized way.

- \* **Using three loop** - Wrote a script to read a design file using three nested loop and then acting along user's wish. Profiling results are as follow:

# Performance Profile Index

For ./exp1\_1.pl

Profile of ./exp1\_1.pl for 15.4s (of 15.4s), executing 14582 statements and 3752 subroutine calls in 9 source files.

Top 15 Subroutines

Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
6	2	1	15.4s	15.4s	main::CORE:readline (opcode)
1	1	1	2.69ms	2.72ms	Config::BEGIN@10
10	2	1	2.26ms	2.26ms	List::MoreUtils::firstidx (xsub)
1	1	1	2.10ms	15.0ms	main::BEGIN@2
5	1	1	1.95ms	1.95ms	DynaLoader::CORE:ftdir (opcode)
1	1	1	1.88ms	7.50ms	List::MoreUtils::BEGIN@6
3537	1	1	1.11ms	1.11ms	main::CORE:match (opcode)
1	1	1	1.02ms	5.58ms	DynaLoader::BEGIN@22
1	1	1	1.00ms	1.00ms	List::MoreUtils::BEGIN@5
1	1	1	852µs	1.75ms	Config::BEGIN@11
1	1	1	760µs	803µs	List::MoreUtils::BEGIN@4
1	1	1	740µs	813µs	vars::BEGIN@7
1	1	1	707µs	707µs	DynaLoader::CORE:ffile (opcode)
88	3	1	670µs	670µs	main::CORE:print (opcode)
1	1	1	436µs	436µs	main::CORE:open (opcode)

See [all 82 subroutines](#)

You can view a [treemap of subroutine exclusive time](#), grouped by package. NYTProf also generates call-graph files in [Graphviz](#) format: [inter-package calls](#), [all inter-subroutine calls](#).

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 3.2: Experiment 1A

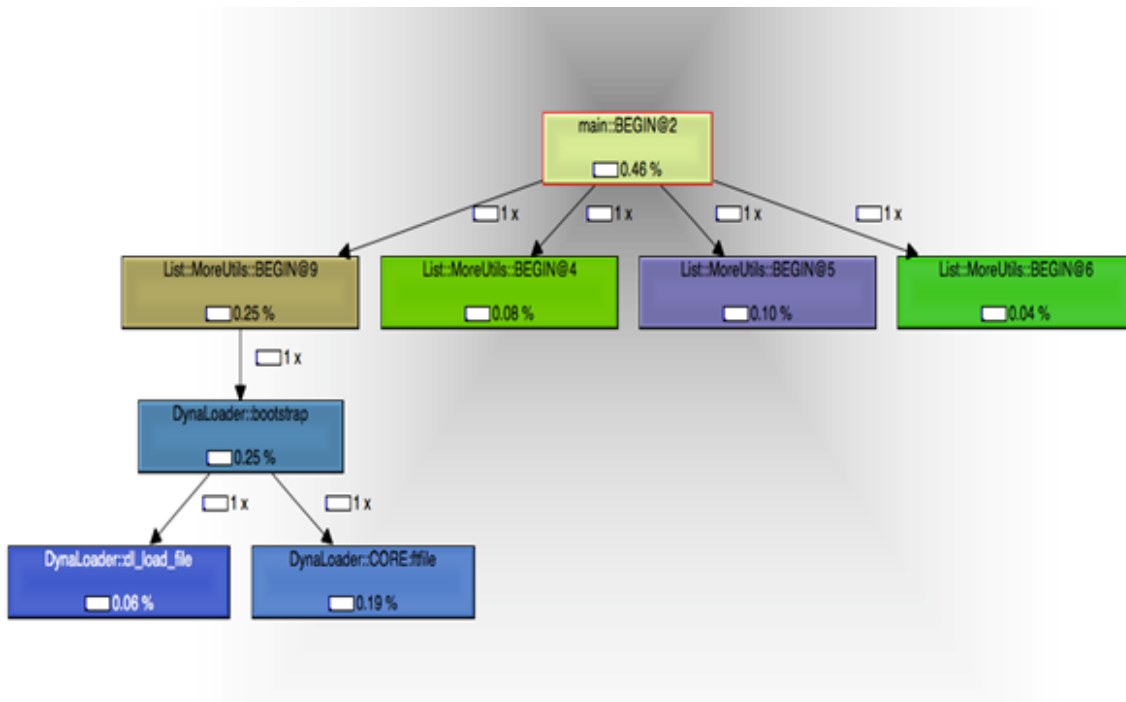


Figure 3.3: Flow Graph of Experiment 1A

- \* **Using Array** – Wrote a script to read a design file using array and then acting along user's wish. Profiling results are as follow:

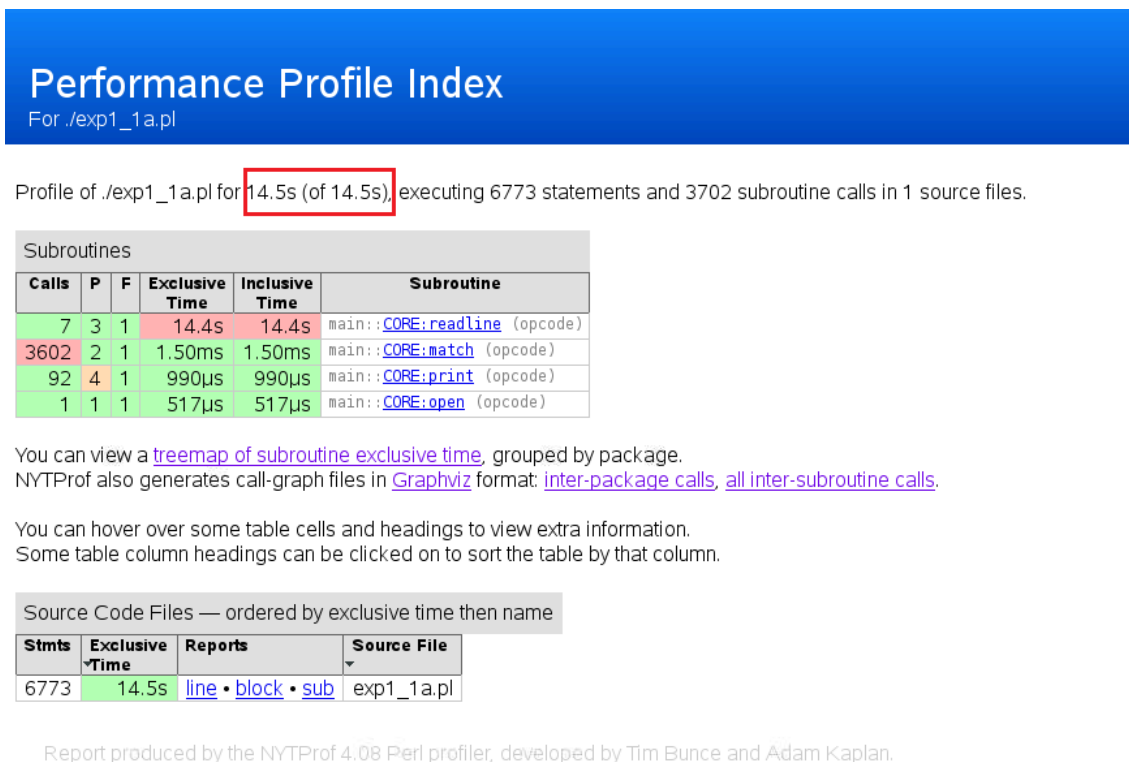


Figure 3.4: Experiment 1B

- \* **Using Hashes** - Wrote a script to read a design file using hashes and then acting along user's wish. Profiling results are as follow:-

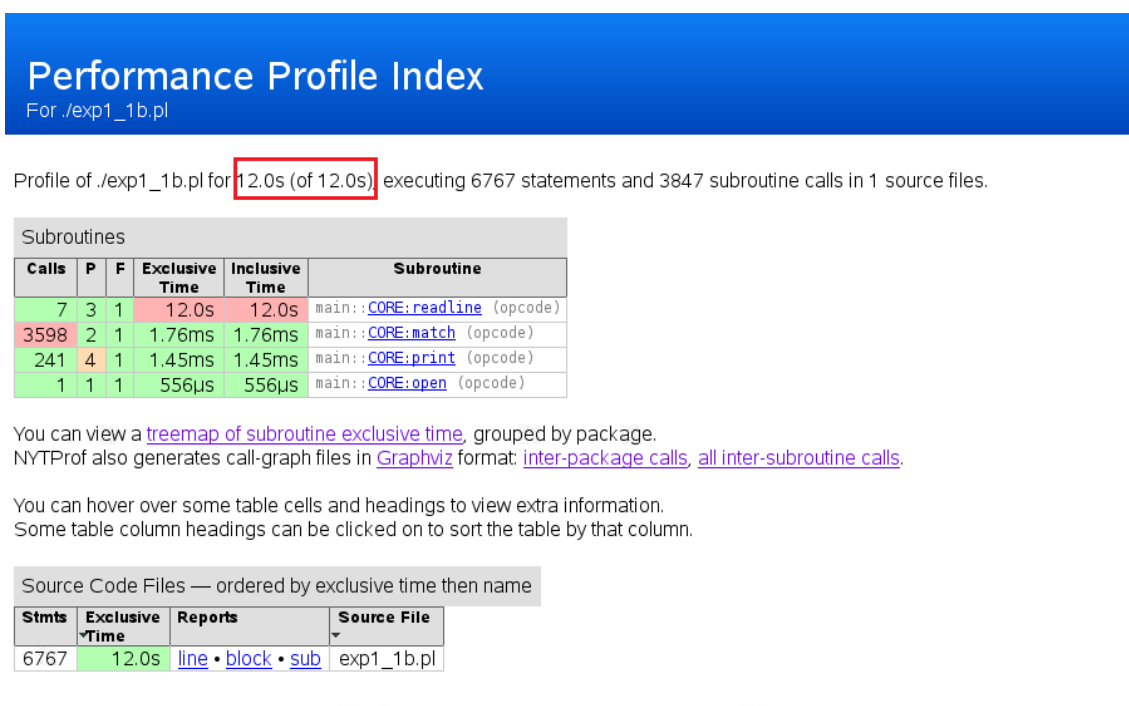


Figure 3.5: Experiment 1C

- \* **Using Regular Expression** - Wrote a script to read a design file using regular expressions and then acting along user's wish. Profiling results are as follow:-

## Performance Profile Index

For ./exp1\_1.c.pl

Profile of ./exp1\_1.c.pl for 222ms (of 272ms) executing 64321 statements and 102876 subroutine calls in 14 source files

Top 15 Subroutines

Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
53006	3	1	19.0ms	19.0ms	main::CORE:match (opcode)
49408	1	1	12.7ms	12.7ms	main::CORE:regcomp (opcode)
1	1	1	8.60ms	17.0ms	main::BEGIN#3
1	1	1	5.97ms	16.3ms	main::BEGIN#2
1	1	1	3.68ms	7.03ms	Data::Dumper::BEGIN#20
1	1	1	2.36ms	2.39ms	Carp::BEGIN#4
1	1	1	2.35ms	2.97ms	Getopt::Long::BEGIN#208
1	1	1	2.00ms	2.08ms	Exporter::as_heavy
1	1	1	1.49ms	1.49ms	XSLoader::load
1	1	1	1.35ms	1.73ms	Getopt::Long::BEGIN#19
130	4	1	1.30ms	1.30ms	main::CORE:print (opcode)
1	1	1	965µs	1.08ms	overload::BEGIN#147
1	1	1	786µs	831µs	Carp::BEGIN#3
7	7	2	666µs	823µs	vars::import
1	1	1	630µs	634µs	Data::Dumper::BEGIN#683

See [all 210 subroutines](#)

You can view a [treemap of subroutine exclusive time](#), grouped by package.

NYTProf also generates call-graph files in [Graphviz](#) format: [inter-package calls](#), [all inter-subroutine calls](#) (probably too cor

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 3.6: Experiment 1D

- \* **Using Hashes and Regular Expression** - a script to read a design file using hashes and regular expressions and then acting along user's wish. Profiling results are as follow:-



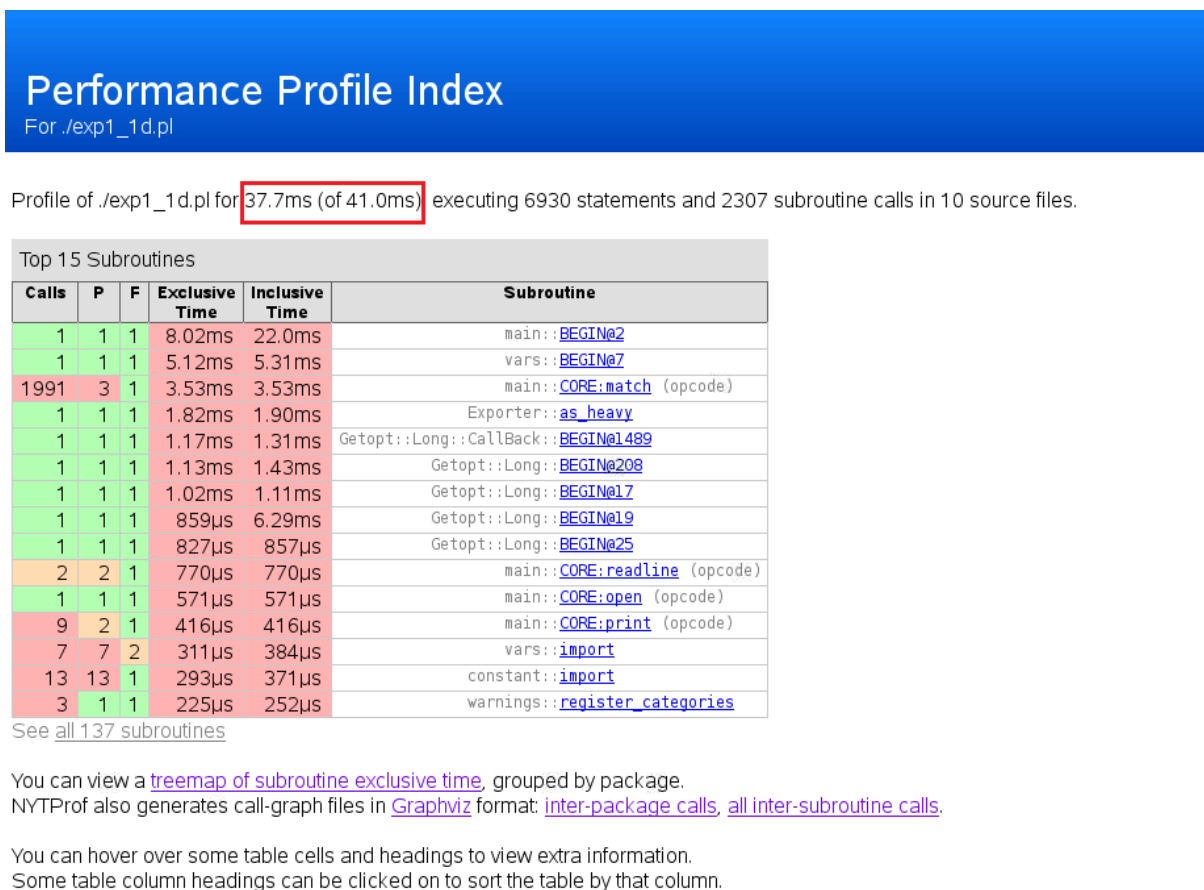


Figure 3.7: Experiment 1E

\* **Exclusive Time** - It is best for Bottom Up Approach

- It gives the amount of time spent “in the code of the subroutine”.
- It gives the location where the time actually gets spent.
- It is appropriate for localized optimization. (Figure 3.11)

\* **Inclusive Time** - It is suited for Top Down Approach

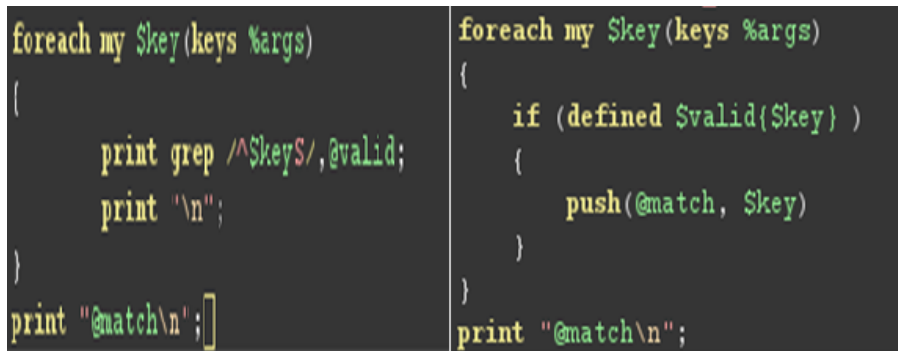
- It gives the amount of time spent “in and below this sub”.
- It is appropriate to prioritize structural optimizations.

Table 3.1: Results of Performance Analysis

S.no	Experiment	Before	After
1.	Using Three Loop	-	15.4s
2.	Using Array	15.4s	14.5s
3.	Using Hashes	14.4s	12s
4.	Using Regular Expressions	12s	222ms
5.	Using Regular Expressions and Hashes	222ms	41ms

Percentage Reduction in time is 98.55% reduction.

- \* **Removal of recurring loop** - This Loop was taking a long time as it used to go each and every element of array. But after the optimization the array was converted into hash and going through it keys takes less time. Earlier the time was 47.3ms and now its 13ms with a reduction of 86.37% in time. The Actual reduction in AS-2 flow because of this was 9.37% but is significant. Profiling results of the above experiments are as follows:-



```

foreach my $key (keys %args)
{
    print grep /^$key$/, @valid;
    print "\n";
}
print "@match\n";

foreach my $key (keys %args)
{
    if (defined $valid{$key} )
    {
        push(@match, $key)
    }
}
print "@match\n";

```

Figure 3.8: Reoccurring Loop

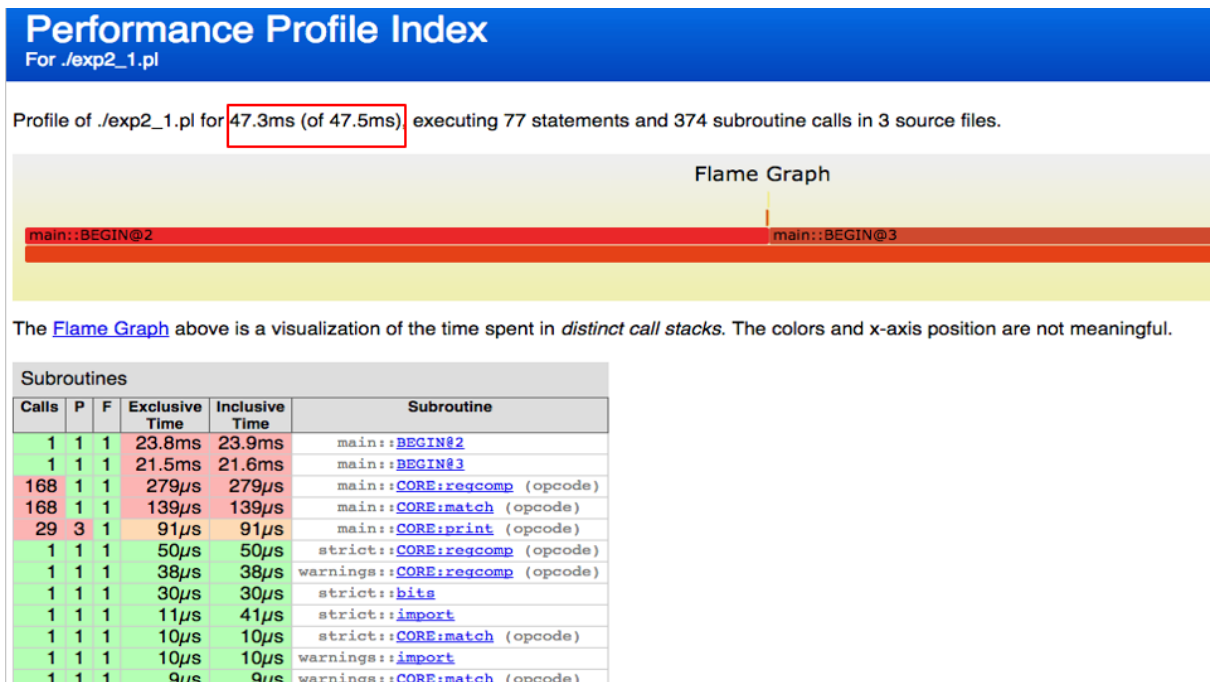


Figure 3.9: Experiment 2A

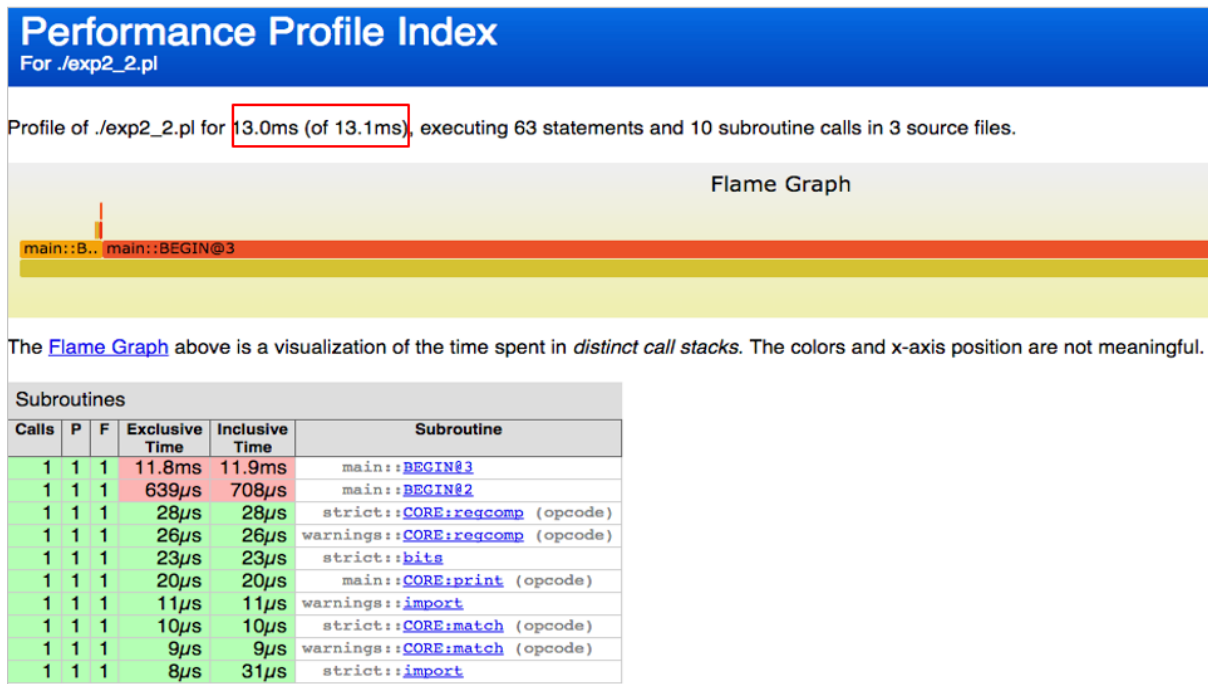


Figure 3.10: Experiment 2B

### 3.3.2 Critical Observation in AS-2

By analyzing the profiling result some observation were made and tried to optimize certain part of it

- \* The current flow was getting slower because of calling a particular subroutine a number of times.
- \* Based on my experiment the calls were reduced by modifying the PERL code using regex, avoiding loops and using hashes.
- \* AS-2 is slower than earlier flow because it was doing more design data analysis.
- \* Running profiling on an IP takes approx. 8hrs which is not feasible for early optimization in the AS-2 flow.

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.

Source Code Files — ordered by exclusive time then name			
Stmts	Exclusive Time	Reports	Source File
1236	32.1ms	line • block • sub	exp1_2.pl
1718	26.6ms	line • block • sub	utf8_heavy.pl
275	14.4ms	line • block • sub	Getopt
101	6.83ms	line • block • sub	Encode
10	3.60ms	line • block • sub	PerlIO/
400	3.60ms	line • block • sub	Encode
352	2.50ms	line • block • sub	constant.pm
168	2.40ms	line • block • sub	vars.pm
61	2.28ms	line • block • sub	base.pm (including 1 string eval)
8	1.96ms	line • block • sub	unicore/Heavy.pl
11	1.71ms	line • block • sub	PerlIO.pm (including 1 string eval)
5	1.42ms	line • block • sub	utf8.pm
29	1.33ms	line • block • sub	Exporter/Heavy.pm
59	1.24ms	line • block • sub	Exporter.pm
296	777µs	line • block • sub	warnings.pm
2	746µs	line • block • sub	XSLoader.pm
67	656µs	line • block • sub	Encode/Encoding.pm
11	571µs	line • block • sub	Encode/Config.pm
23	517µs	line • block • sub	overload.pm
110	462µs	line • block • sub	strict.pm
18	66µs	line • block • sub	warnings/register.pm
1	10µs	line • block • sub	unicore/lib/Perl/Word.pl
4	10µs	line • block • sub	bytes.pm
1	8µs	line • block • sub	unicore/lib/Perl/SpacePer.pl
4966	106ms	<i>Total (-5 statements are unaccounted for)</i>	
206	4.41ms	<i>Average</i>	
	1.33ms	<i>Median</i>	
	0.00116	<i>Deviation</i>	

**Color coding based on Median Average Deviation relative to rest of this file**

Figure 3.11: Color Coding in NYTProf

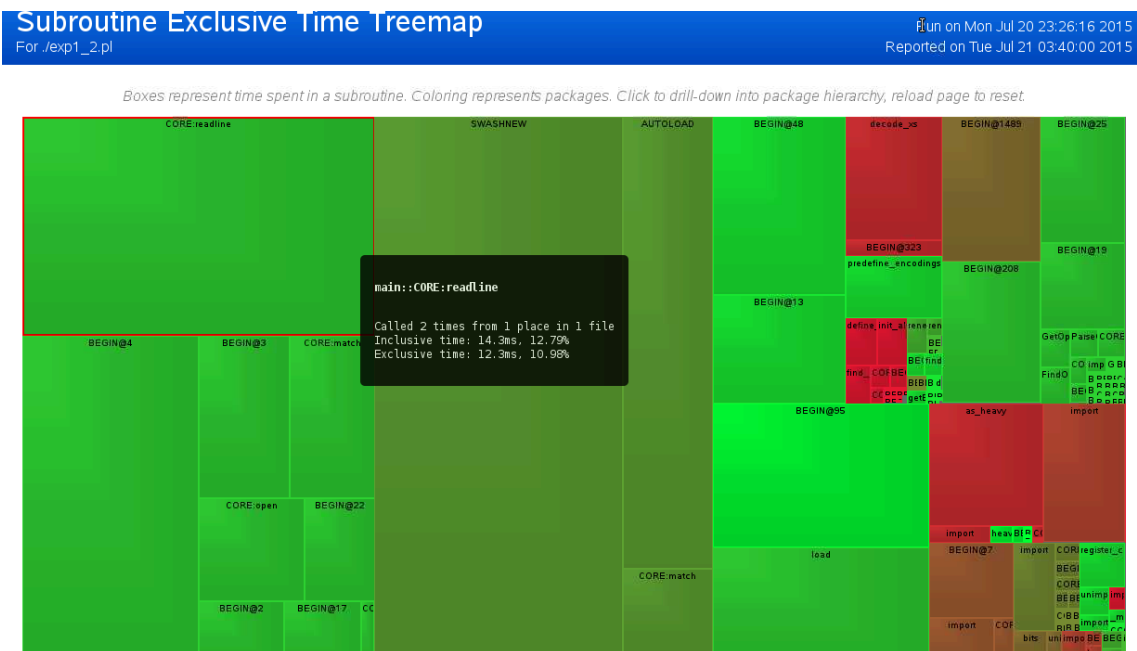


Figure 3.12: Call Tree in NYTProf

# Chapter 4

## Design Prototyping and Results

Running profiling on a SoC takes approx. 8hrs which is not feasible for early optimization in the AS-2 flow. Moreover this is only the run time, analysis of the results takes more time which hinders the requirement to meet time to market window. AS-2 run all the regression on a big IP which has maximum quality checks but not all. Thus it would be better to have a prototype which runs all the quality checks by having proper test cases.

### 4.1 Importance of AS-2 Prototype

”Just as verifying a SoC design on FPGA prototype is a secure way to ensure that it is functionally correct and prevents re-spin[9] of the product saving time to market window, similarly running AS-2 flow on prototype saves compile time.”

Similarly to meet the time to market constraint, AS-2 need to be run on small and effective design rather than big IP that consume large amount of time. Thus a dummy design which takes less amount of time and runs all quality check and gives maximum coverage are needed. AS-2 prototype inherits all the quality checks and run small test-cases to validate actual IP test scenarios. It creates design black-box which may not mimic SoC functionally but has all the flow related collateral's.

For testing critical part or write the test cases a Test Driven Development approach is needed. First a test is written which fails so as to reproduce the error (the tests go 'Red'). Next test are written that make code quality checks pass (the tests go 'Green'). Now the developer takes the opportunity to optimize or refactor the code (tidy it up, improve the readability, remove duplication etc.). The developer had to keep in mind that while refactoring a bug may inadvertently be introduced and the tests will go 'Red' to highlight the problem.

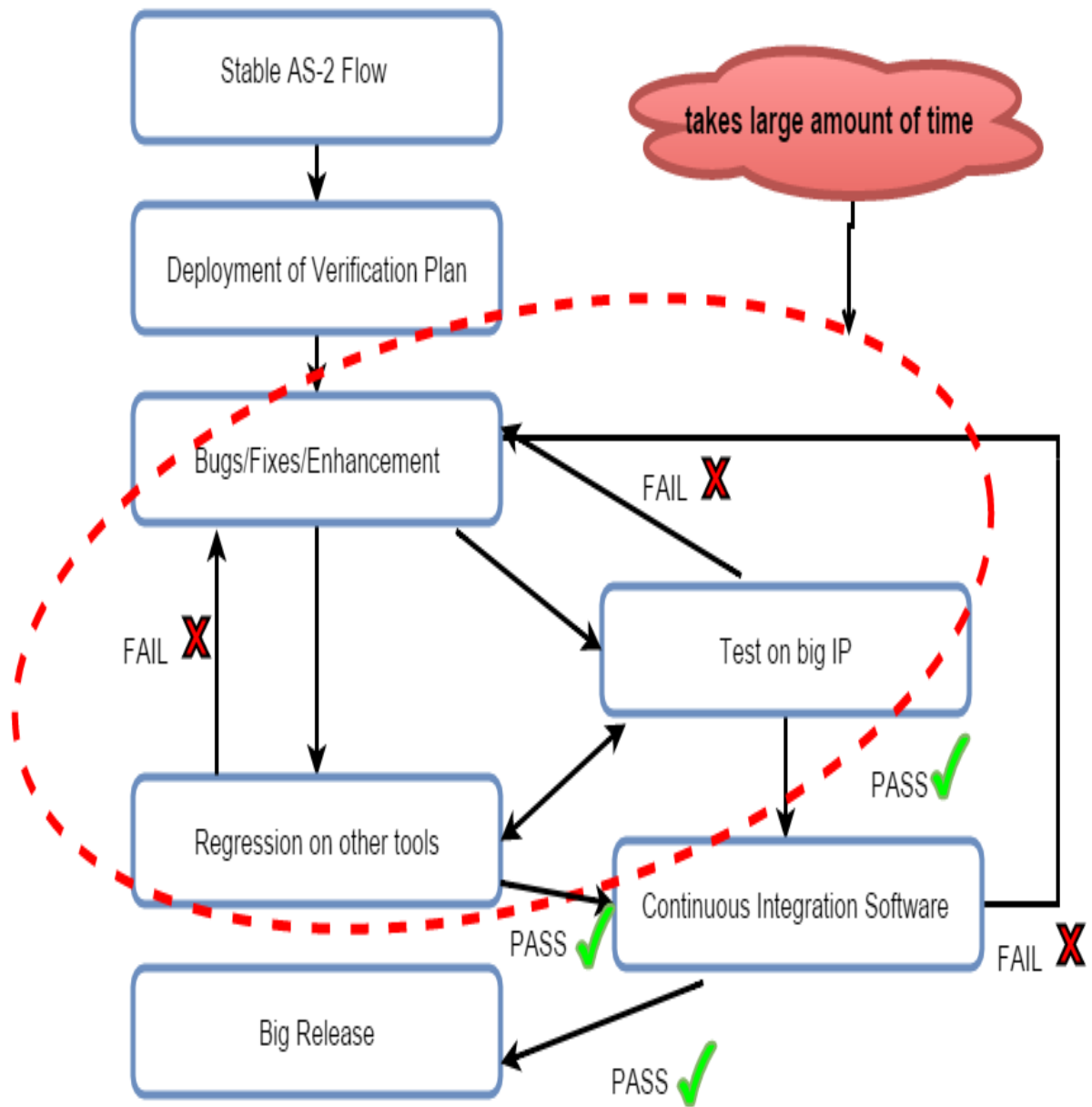


Figure 4.1: AS-2 Work Flow with IP

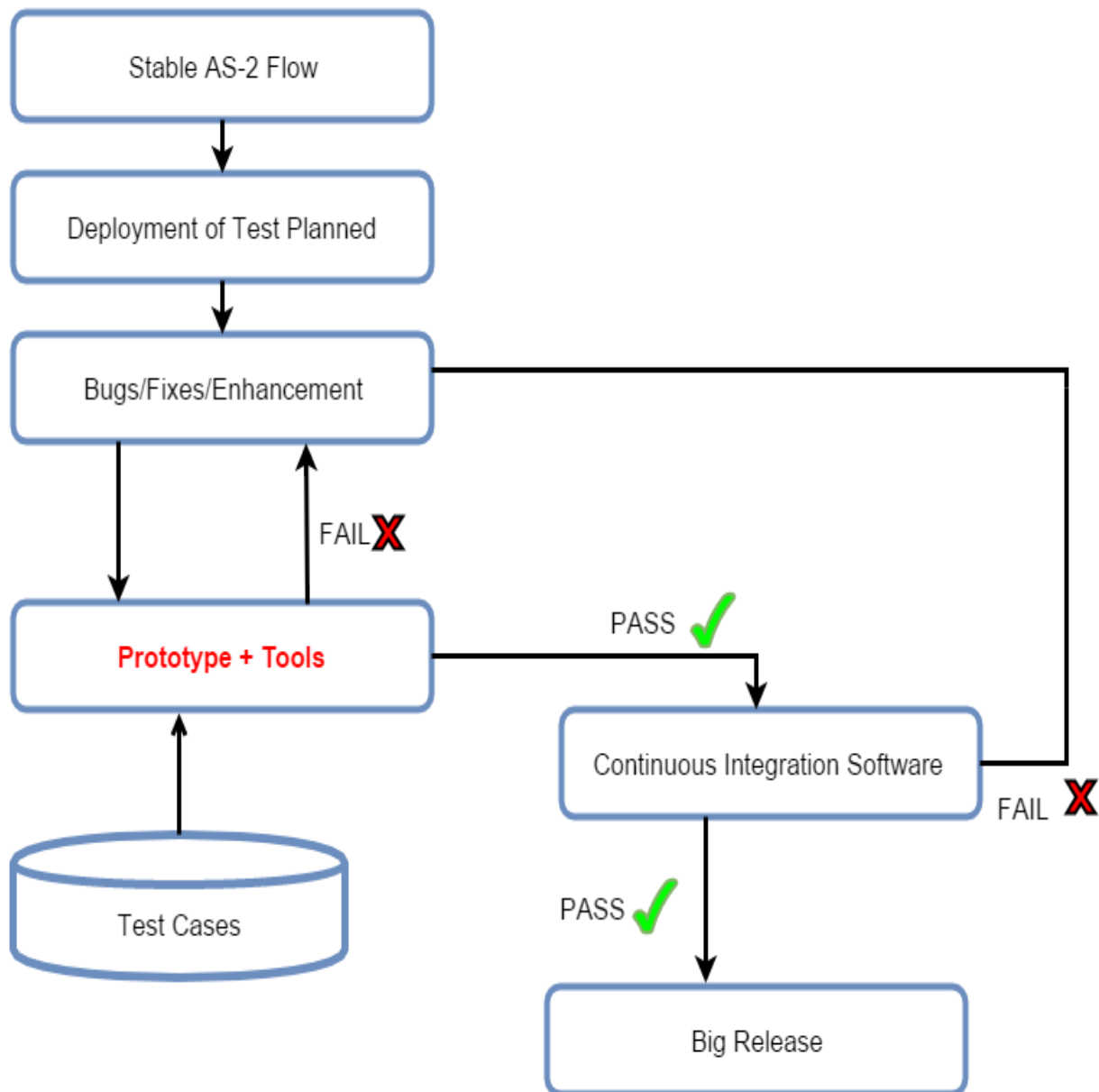


Figure 4.2: AS-2 Work Flow with Prototype

As it's clear from Figure 4.1 and 4.2 the complexity of the system has been reduced and it takes less compile time overall giving results early. The test-cases that are being added to prototype model check different parts of the system through the dummy design, in our case AS-2 Prototype.

## 4.2 Perl Testing Module

The test cases added to prototype are done with help of PERL test modules. Some advantages of it are :-

- \* Can check memory

- \* Speedy way to run the test.
- \* Ease of development.

Basic steps for writing Perl test is:-

- \* Create tests with Test::More.
- \* Prove me command is used to run the tests.

Test::More is the most efficient Perl testing module. It provides many useful subroutines like 'use\_ok' , 'compare\_ok' etc. It is build on Test::Builder framework, so it can be used with any other testing utility module that is based on Test::Builder. Test::More outputs data in the Test Anything Protocol (TAP), so this fits very well in TAP-friendly systems like Test::Harness, and prove.

"All of the test methods return a true or false value depending on whether the given test passed or failed. This helps in adding conditional logic to your test code. Most of the test methods also take the test name as their final argument so as to report when the test is executed and also when the test fails".[8].

### 4.3 Test Cases added to AS-2 Prototype

Test-driven development (TDD), is a method in which a source design is repeatedly tested. The concept on which it is based is to "make something work now and perfect it later." After each test is passed, re-factoring is done on it and then the test is run again. The process is repeated many times until each section of test is functioning according to needs. TDD can produce test cases of high quality in less time as compared to older methods. Proper implementation of TDD requires the testers and developers to accurately analyse implementation of test case and its features in the real world. Problems are approached in an incremental way and tests intended for the application may have to be redone many times. TDD approach ensures that all the section in an application have been tested for correct functionality, both individually and in sync with one another. Tests should be conducted from the very beginning of the design cycle as it saves time and money spent in debugging at later stages.[4].

One of the major limitations of TDD is the fact that tests can sometimes be incorrectly conceived or applied. Because of this the sections may not perform as expected in the real world. Even if all the section work as expected in isolation and integrated with other modules, end users may have a situations not imagined by the developers and testers. The final approach of TDD are only as good as the tests, its thoroughness with which they have implemented and the extent to which they mimic conditions encountered by users of the final product. Similar approach is needed for test-case generation of AS-2 prototype.



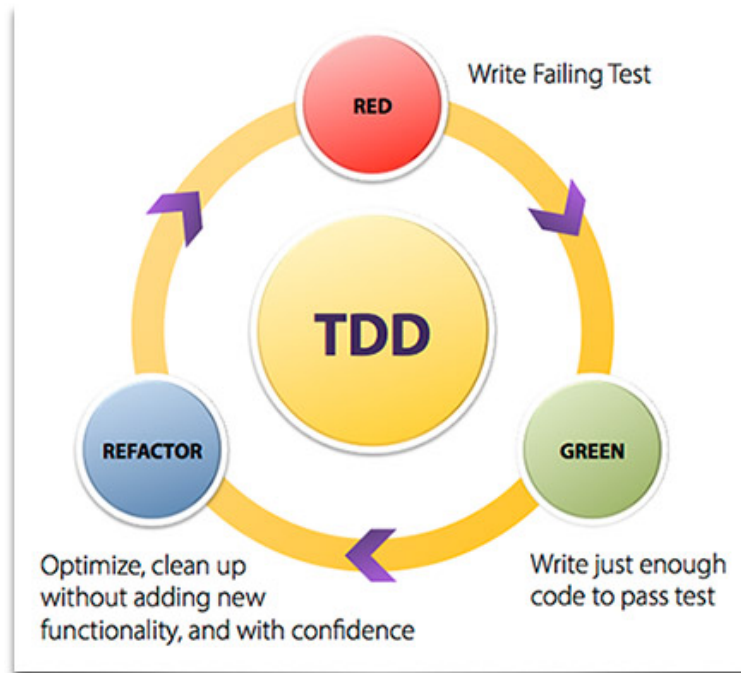


Figure 4.3: Test Driven Development

Added new Design Under Test (DUT-2) to mimic the functionality of the IP. From one of the legacy flow a design was taken and integrated with the AS-2. All the necessary libraries and attributes and quality checks collateral's were added on which a AS-2 can be deployed for various reasons. The prototype is a dummy design, so here the aim will be to even check the corner cases that can come up during the run and fail the design. The test based approach will be based on to get maximum coverage of the design so that the designs have to go for re-spin and save time to market.

### 4.3.1 Test for loading various checks for the SoC or IP

This test load the various checks needed to verify on the IP. There may be quality checks, flow checks, for static check, configuration check. The system read the design files that may be in verilog or system verilog configuration. If the test fails that means anything breaks in loading the configuration of the design. The performance results of the test are as follows:-

# Performance Profile Index

For `t/` /stages.t

Profile of `.` for 15.1s (of 15.2s), executing 14233 statements and 4147 subroutine calls in 79 source files and 23 string evals.

Top 15 Subroutines					
Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
356	1	1	14.9s	14.9s	<a href="#">::CORE:readline</a> (opcode)
1	1	1	13.9ms	15.5ms	<a href="#">::BEGIN@5</a>
1	1	1	10.4ms	10.4ms	<a href="#">::CORE:mkdir</a> (opcode)
1	1	1	8.92ms	11.4ms	<a href="#">::BEGIN@39</a>
1	1	1	8.78ms	9.97ms	<a href="#">::BEGIN@22</a>
1	1	1	7.82ms	23.2ms	<a href="#">::BEGIN@48</a>
1	1	1	7.66ms	8.77ms	<a href="#">::BEGIN@45</a>
1	1	1	6.75ms	19.9ms	<a href="#">::BEGIN@9</a>
1	1	1	6.70ms	27.2ms	<a href="#">::BEGIN@9</a>
8	8	8	5.79ms	5.79ms	<a href="#">::load</a>
1	1	1	5.04ms	62.6ms	<a href="#">::BEGIN@12</a>
1	1	1	4.98ms	14.9s	<a href="#">::execute</a>
2	2	2	4.73ms	8.50ms	<a href="#">::module_available</a>
1	1	1	4.65ms	4.98ms	<a href="#">::BEGIN@7</a>
1	1	1	4.59ms	5.95ms	<a href="#">::BEGIN@47</a>

[See all 1286 subroutines](#)

Figure 4.4: Test for loading checks on SoC 1

Profile of `/project_2.t` for 318s (of 318s), executing 19969 statements and 6298 subroutine calls in 79 source files and 23 string evals.

Top 15 Subroutines					
Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
1073	1	1	318s	318s	<a href="#">::CORE:readline</a> (opcode)
1	1	1	14.6ms	318s	<a href="#">::execute</a>
1	1	1	11.7ms	13.2ms	<a href="#">::BEGIN@5</a>
1	1	1	9.09ms	9.09ms	<a href="#">::CORE:mkdir</a> (opcode)
1076	5	2	7.86ms	9.52ms	<a href="#">::_ANON_/usr/intel/pkg/perl/5.14.1/lib64/module/r1/Log/Log4perl.pm:138]</a>
1	1	1	7.71ms	8.83ms	<a href="#">::BEGIN@45</a>
1	1	1	6.81ms	22.7ms	<a href="#">::BEGIN@48</a>
8	8	8	6.70ms	6.70ms	<a href="#">::load</a>
1	1	1	5.83ms	27.3ms	<a href="#">::BEGIN@5</a>
1	1	1	5.55ms	8.55ms	<a href="#">::BEGIN@39</a>
1	1	1	5.52ms	23.2ms	<a href="#">::BEGIN@9</a>
2	2	2	5.47ms	9.16ms	<a href="#">::module_available</a>
1	1	1	5.10ms	5.76ms	<a href="#">::BEGIN@22</a>
1	1	1	4.84ms	63.3ms	<a href="#">::BEGIN@12</a>
1	1	1	4.69ms	5.00ms	<a href="#">::BEGIN@7</a>

[See all 1286 subroutines](#)

Figure 4.5: Test for loading checks on SoC 2

- \* The prototype contains many Design Under Test units which need to qualify certain quality checks.
- \* While enabling the quality checks for DUT-2 it happens that same quality checks are applied to DUT-1 as both the DUT have some common library and dump output at the same place which causes a race condition.
- \* This Race Condition was removed by dumping the output of the DUTs at different places or running DUTs sequentially.

### 4.3.2 Test to check attributes being passed from parent to child design configuration for the SoC or IP

Certain attributes were there in parent design configuration that were getting propagated to child design configuration which is not valid and should not be happening. The child attribute can be inherited by the parent but vice-versa is a bug.

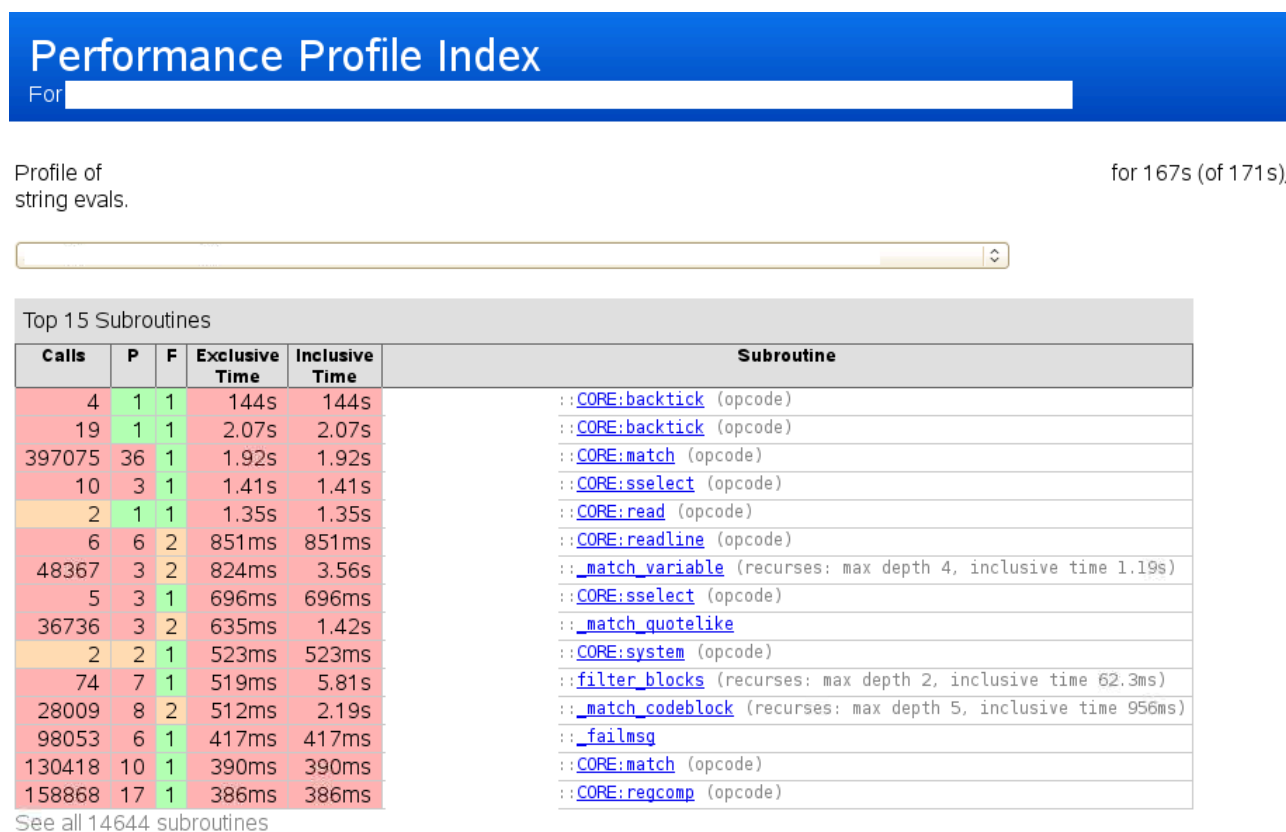


Figure 4.6: Test for checking attributes

- \* The test in this case passed correctly successfully but didn't justified the sub-routine written in a flow part.
- \* Further debugging revealed that the subroutine for calling parent attribute was not working as intended and attributes were not loaded correctly.

### 4.3.3 Test for Flow Equivalence Checking for the SoC or IP

Equivalence Checking is always carried out using two inputs and result comes out by comparing the functionality of these two input designs. This test compares the golden generated flow specification with the flow specification generated after the compilation of the design. The test fails if anything extra is attribute gets added or deleted during the compilation. With this the debugging gets easy and error can be removed in a much faster way.

## Performance Profile Index

For tflowspec.t

Profile of tflowspec.t for 304s (of 304s), executing 46429 statements and 20127 subroutine calls in 85 source files and 168 string evals.

Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
1071	1	1	304s	304s	::CORE:readline (opcode)
4054	2	1	120ms	120ms	::CORE:readline (opcode)
4026	1	1	16.9ms	16.9ms	::CORE:subst (opcode)
28	2	1	16.7ms	166ms	::read_and_filter_handle
1	1	1	16.3ms	16.3ms	::CORE:mkdir (opcode)
1	1	1	14.1ms	304s	::execute
28	2	1	12.9ms	12.9ms	::CORE:open (opcode)
4026	1	1	12.2ms	29.1ms	::trim
1	1	1	10.1ms	11.7ms	::BEGIN#5
14	1	1	9.79ms	12.8ms	::diff
1	1	1	9.65ms	11.8ms	::BEGIN#45
1	1	1	9.00ms	10.3ms	::BEGIN#7
1	1	1	8.58ms	9.52ms	::BEGIN#4
2	2	2	7.98ms	11.5ms	::module_available
1	1	1	7.90ms	10.8ms	::BEGIN#39

See all 1426 subroutines

Figure 4.7: Test for Comparing Flow Specification

- \* This test passed initially but it failed when deployed on actual flow.
- \* The test used “compare\_ok” module which also compared the leading trailing white-spaces, which should be ignored.
- \* Added a trim function to make the test pass successfully.

### 4.3.4 Test for checking whether AS-2 modules are Perltidy compatible

This test run Perltidy checks, which is a standard linting tool on AS-2 flow modules. The test take a configuration file and checks whether the modules qualify the lint checks. The test passes if the files are according to the rules specified and fails if the files are not up-to the mark generating a .tdy file. The user can diff both the files and when satisfied with changes can push in main line.

```

for my $word (keys %{$word{$len}}){
chop(my $prefix = $word);if ($opt{g}){
while( $prefix ){
if(my $words=delete$chain{ $prefix} )
{ $chain{$word} = [ @$words, $word ];
$maxcount=max ($maxcount,@$words+1); last;}

chop $prefix; }
}else{ if (my $words = delete
$chain{$prefix}){$chain{$word} = [@$words,
$word]; $changed = 1;} }}

```

Figure 4.8: Before Perltidy

```

for my $word (keys %{$word{$len}}) {
chop(my $prefix = $word);
if ($opt{g}) {
while ($prefix) {
if (my $words = delete $chain{$prefix}) {

$chain{$word} = [@$words, $word];
$maxcount = max($maxcount, @$words + 1);
last;
}
chop $prefix;
}
}
else {
if (my $words = delete $chain{$prefix}) {
$chain{$word} = [@$words, $word];
$changed = 1;
}
}
}
}

```

Figure 4.9: After Perltidy

This test passed but while manually running Perltidy from command line the .tdy file generated by test and command line were different. With further debugging it was found that this was because of the version difference of Perl used. By using PERL5LIB to set proper perl version this problem was solved.

### 4.3.5 Test for Performance Checking of AS-2 prototype

This test runs profiling using nyptrprof on AS-2 prototype and monitors the time taken to run AS-2 on DUT-2 which is a small design in AS-2 prototype. The test generates a warning whenever a warning threshold is crossed and generates an error whenever failing threshold. The warning and failing threshold is decide by successive runs of the flow on AS-2 prototype.

```
$nyptrprof file = "$nyptrprof dir/nyptrprof/index.html";  
my $run_threshold = 40; #seconds  
my $failing_threshold = 70;  
my ( $fh1, $run_time );  
open( $fh1, "< $nyptrprof_file" ) or die "unable to open file '$nyptrprof_file' for reading : $!";  
while ( my $line = <$fh1> ) {  
    if ( $line =~ m/(Profile\s+of\s+S\s+for\s+)(.*)\s+(of.*s)\s+executing/ ) {  
        $run_time = $2;  
        last;  
    }  
}
```

Figure 4.10: Test for Performance Checking of AS-2 prototype[1]



Figure 4.11: Test for Performance Checking of AS-2 prototype[2]

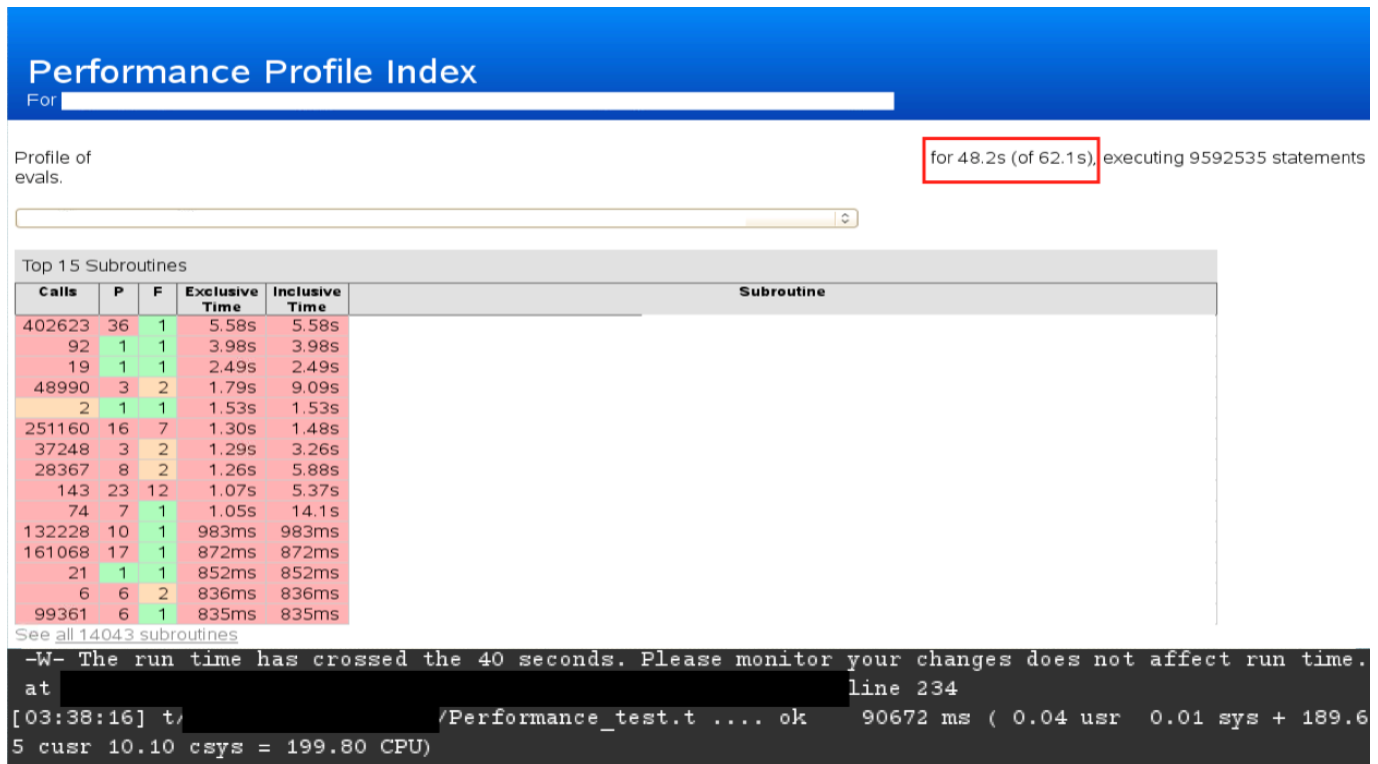


Figure 4.12: Test for Performance Checking of AS-2 prototype[3]

## 4.3.6 Improvement in Test

### 4.3.6.1 Redirecting Test output to a log file

Logging beats a debugger when you want to know what's going on in the code or flow during run-time. However, traditional logging packages are too static and generate large amount of log messages in your log files that won't help you. "Test::Simple and Test::More have proven to be popular testing modules, but they're not always flexible enough. Test::Builder provides a building block upon which to write your own test libraries which can work together." [8].

One of such flexibility is to redirect my test output to a log file. This would help in easy debugging whenever a test failed. By using Test::Builder option, the output of my Test::More print outs to a log file instead of STDOUT / STDERR. The test have many "is/ok/like" calls in the code running in loops and the output is often thousands lines of tests long. Some examples are as follows:-

\* output

- \$Test::output(\$fh);
- \$Test::output(\$file);
- Where normal "ok/not ok" test output should go. Default output is STDOUT. \$Test is a object for Test::Builder.

- \* failure\_output

- `$Test::failure_output($fh);`
- `$Test::failure_output($file);`
- Where diagnostic output on test failures and `diag()` should go. Defaults is set to `STDERR`.

- \* todo\_output

- `$Test::todo_output($fh);`
- `$Test::todo_output($file);`
- Where diagnostics about todo test failures and `diag()` should go. Defaults to set to `STDOUT`.

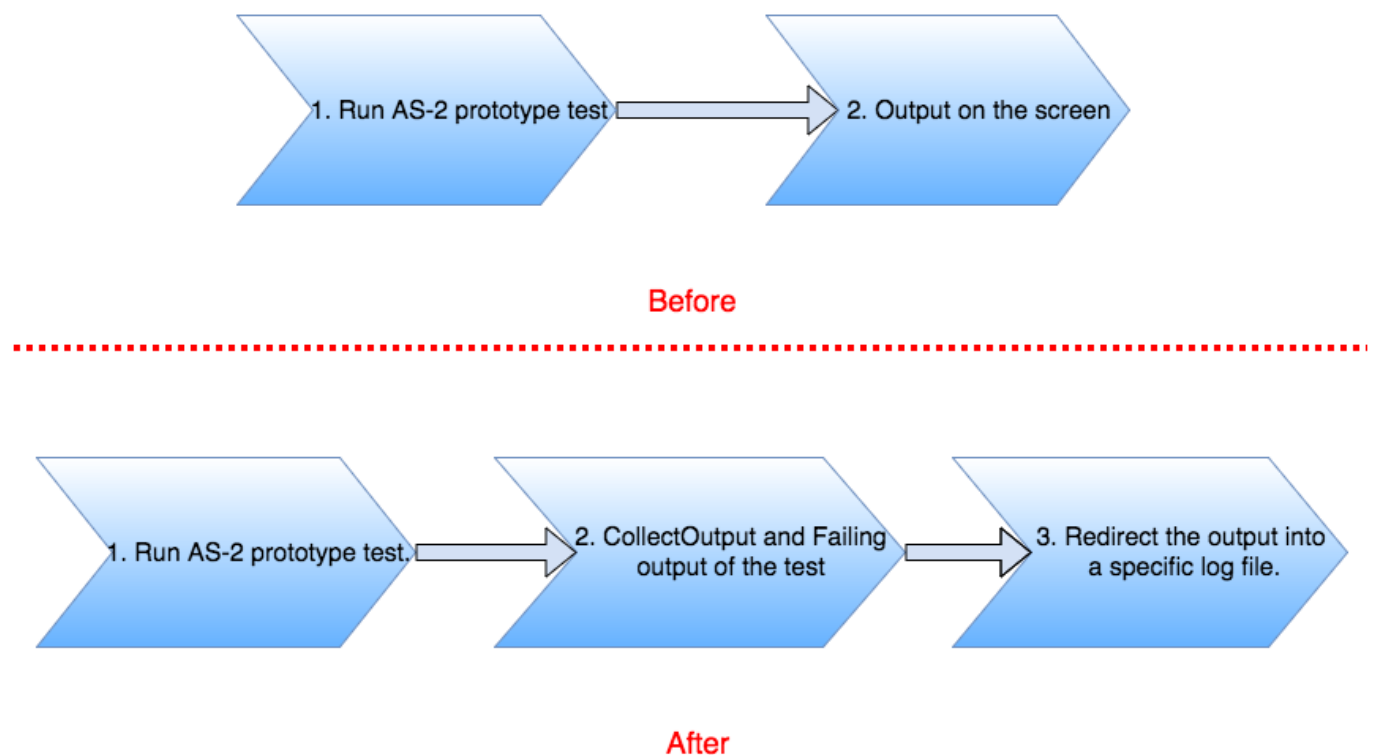


Figure 4.13: Logging of AS-2 Flow test

- \* The AS-2 prototype tests generated a Log file for each test. This helped the user with easy debugging of failing test without clouding the screen.
- \* The standard function `STDOUT` and `STDERR` does not work in test environment. Therefore an object for Builder module needs to be created for redirecting the output.



#### 4.3.6.2 Enable Screen Filter for AS-2 flow

The message getting printed on screen or log files belong to different categories such as an info or a warning. A screen filter allows you to control the number of logging messages generated at these different levels:-

- \* TRACE(-T-)
- \* DEBUG(-D-)
- \* INFO(-I-)
- \* WARN(-W-)
- \* ERROR(-E-)
- \* FATAL(-F-)

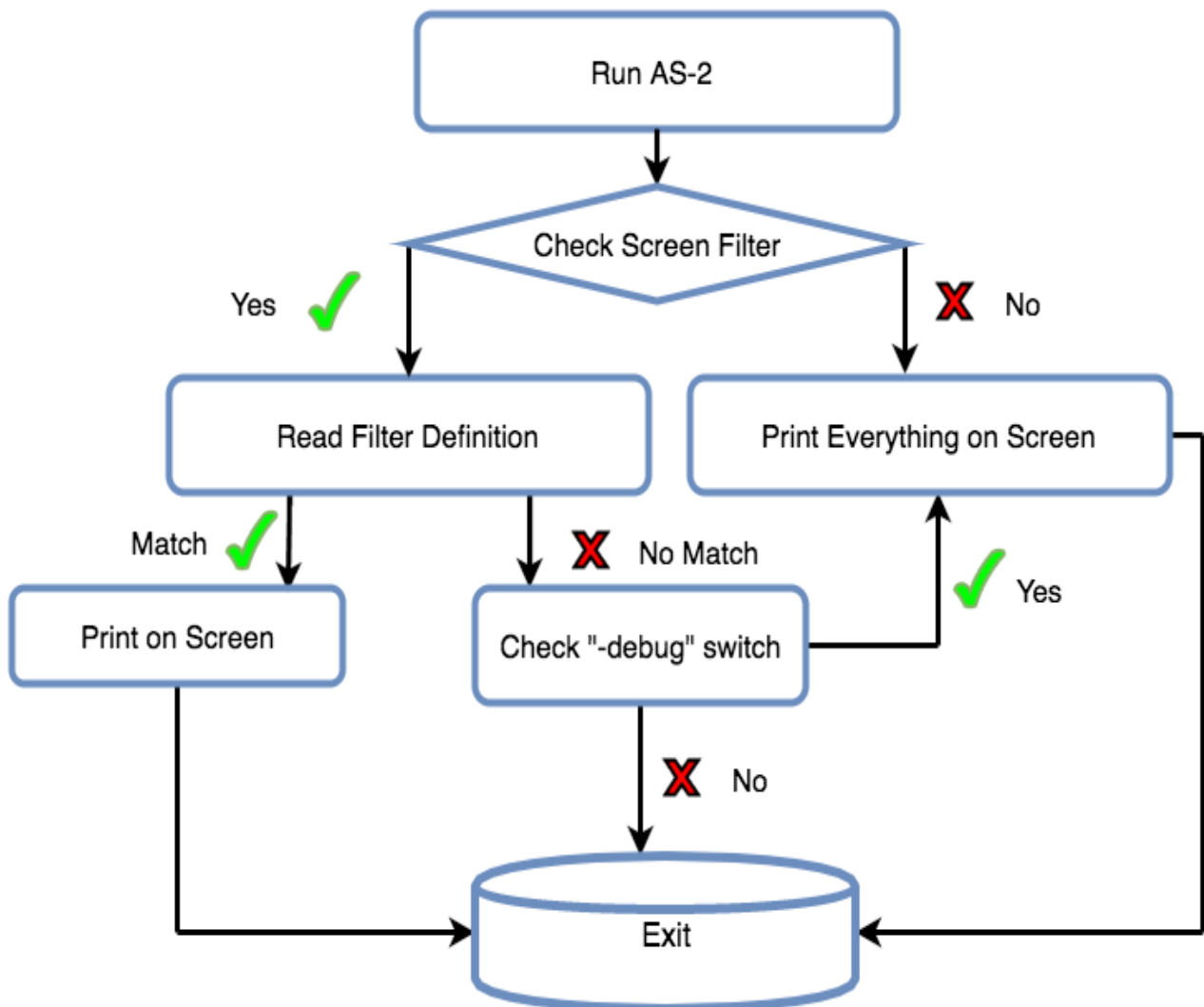


Figure 4.14: Function of Screen Filter

- \* AS-2 flow from its a central location and corresponding tools would generate logs. The messages in the log file belong to the above defined categories. When the screen filter switch is enabled the flow reads the filter definition and prints on the screen whatever matches in the definition.
- \* Screen Filter allows you to control the amount of logging messages displayed on screen very effectively. The logic of screen filter gives highest priority to ”-debug” switch and whenever that switch is encountered, everything is printed on the screen.
- \* A new filter definition can be written according to our requirement. Whenever we need only Die message or error message to be printed we could modify accordingly the filter definition.
- \* While enabling screen filter for AS-2, even warning and fatal messages from AS-2 subtools were printed as an info message.

### 4.3.7 Unit Test for Environment setup for AS-2 Flow

Unit testing is a way to test individual components of code with automatic verification. With normal testing we usually forget to verify that a change in one function didn't break another function but with unit test that is not the case. The basic steps for unit test writing are as follows:-

- \* Define the input, output, and process.
- \* Setup the project (create stub functions and unit test contracts).
- \* Modify and run the test until it passes.

**Specification:** For running AS-2 flow, certain Environment variables need to be set prior to loading modules or stages. For this a configuration file is generated in beginning of AS-2 flow. The importance of configuration file are as follows:-

- \* AS-2 flow reads configuration and figures out the tools to be used.
- \* It also helps to load library required for the flow either from the local repository or from the central where the actual collateral's are present. The collateral's can be in form of IP needed when AS-2 runs on sub-system or an SoC.
- \* All these configuration updation were earlier hand maintained and prone to error. But AS-2 have automated this process, and thus this automation needs quality and we need a unit test.

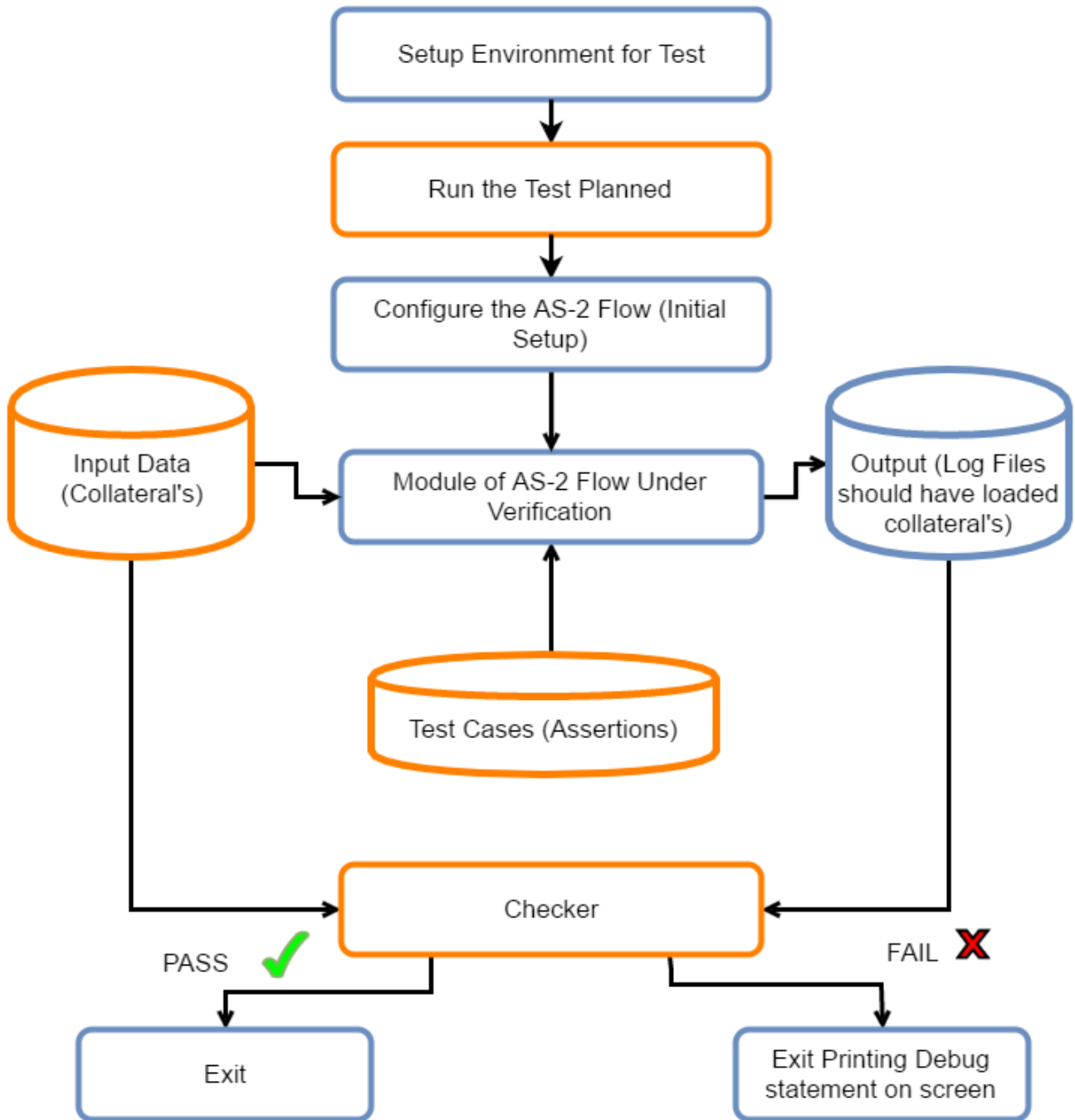


Figure 4.15: Function of Unit Test

**Implementation:** The test is needed to override the data in the configure file with the data input provided by an input file or test itself. The unit test inputs configuration file and check where the collateral's (IP) are present and loads collateral's for an SoC on which AS-2 flow is running.

**Critical Findings:** Here are some of the critical finding regarding the unit test:

- \* For reading the input configuration file the environment setup for test had to be modified to read my configuration file which was not getting overridden. Thus inputs need to be compared to output indirectly by passing inside the test itself.
- \* The collateral's generated should be DUT specific which was not the case and was same for all the DUT's.
- \* If the user has given setting in command line then it should be given priority rather than the default settings.

## 4.4 Summary

This chapter thus describes the development of the AS-2 prototype and results of the development observed so far. The prototype has been made much more efficient to qualify regression on it and validate any IP or SoC using AS-2 flow.

# Conclusion

Time-to-market is one of the key factors for any company to release its product and get hold on customers. The designing part should be completed in a short span of time so that verification and fabrication can be completed without missing technology window. Verification of the design takes about 70% of the total turn around time. The main aim of this project is to provide efficient methodology for front end process through which designers and verification engineers can complete the front end tasks using a single flow with some time saving. AS-2 which is a front end design IP validation system. It is converged, 2-step, and based on next generation build flows. It is a common solution used for both big and small IP. It is a next generation builds flow, capable of handling complex design capabilities and difficult algorithm.

With the improvement in AS-2 flow, it has become 75% faster with proper regression that cover almost all the corner cases of next generation IP. The advancement done in AS-2 flow are time saving and helps to cut down the time required for front end verification process. It provided a great platform both for smaller or complex designs making AS-2 used as a cutting edge build flow.

# References

- [1] Randal Schwartz, Tom Christiansen Larry Wall, “Hashes,” in *Learning Perl*, 2nd ed., O’REILLY,1997, ch. 5, sec. 5.4.
- [2] A.K. Oudjida, D. Benamrouche ”*Front-end IP development: Basic know-how*”, IEEE Conference Publications,Rabat, 2007.
- [3] David Till, “*Object-Oriented Programming in Perl*,” *Teach Yourself Perl 5 in 21 days*, 2nd ed., Sams Publishing, 1996, ch. 19.
- [4] Grant McLean “*An Introduction to Test Driven Development Using Perl*”, Catalyst IT Limited, Sept. 2008.
- [5] Prof. I. Sengupta (2008 Aug 6). Lecture Series on Internet Technologies  
Available : <http://nptel.iitm.ac.in>
- [6] ”Intelpedia,” Intel, December 2015.  
Available : <https://intelpedia.intel.com>
- [7] Josef Weidendorfer, ”*The KCachegrind Handbook*”, 1st ed.
- [8] ”Test::Builder” - Backend for building test libraries  
Available : <http://search.cpan.org/~exodist/Test-Simple-1.001014/lib/Test/Builder.pm>
- [9] Muhammad Aamir ”*Implementation and testing of optimal design of RTU hardware for Wireless SCADA*”, IEEE Conference Publications,Aalborg , 2014.