IP COLLATERAL MANAGEMENT API DEVELOPMENT AND VALIDATION SYSTEM

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering

in

Compubter Science and Engineering

Submitted by

TARIKA MATHUR

(Roll No. 801632053)

Under the supervision of:

Mr. Lalit Monoj B C Satapathy,

Engineering Manager, Intel Corporation

Dr. Karun Verma,

Assistant Professor, Thapar University

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY

PATIALA-147004

JUNE 2018

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, *"IP COLLATERAL MANAGEMENT API DEVELOPMENT AND VALIDATION SYSTEM"*, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology, Patiala, is an authentic record of my own work carried out under the supervision of *Mr. Lalit Mohan Satapathy and Dr. Karun Verma* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature:

(Tarika Mathur)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Lalit Manoj B C Satapathy)

(Dr. Karun Verma)
Assistant Professor
CSED.

ACKNOWLEDGEMENTS

Accomplishment of any project requires the hard work and efforts of many people. I would like to extend my gratitude to Dr. Maninder Singh, Head, Computer Science and Engineering Department and Dr. S S Bhatia, Dean, for giving the environment that encouraged me to work towards my goal. I am also grateful to our Vice Presidents and our Vice Chancellor for their encouragement and support.

I would like to convey my deep sense of gratitude to my project guide, Dr. Karun Verma, who is a constant source of motivation and firm support in carrying out this project. The support and supervision that he gave has helped me to progress in the project. His co-operation is highly appreciated and I highly oblige to him for his valuable comments and moral support during this research period.

I would also like to thank my manager, Mr. Lalit Mohan Satapathy, Engineering Man-ager, Intel Corporation, Bangalore for their esteemed guidance, valuable suggestions and time throughout my internship. Their guidance and vast knowledge directed me to accomplish critical tasks smoothly. Also, my special gratitude to my family for their constant support and love.

I thank all the lecturers and professors of Department of Computer Science and En-gineering, Thapar Institute of Technology for helping us out in the respective subjects whenever needed and giving us more input regarding the same.

ABSTRACT

The development of a VLSI product can be explained as an idea developing as a set of steps, each of which improves the level of detail, moving towards a functionality exploiting the features of a semiconductor. Owing to the continuous improvements in the market, System on Chip design has emerged as an never ending choice for large scale complex system requirements. It involves high quality IP development. This process broadly goes through a front end and another back end set of steps along with a bunch of validation steps.

The steps are linked to each other by way of inter dependencies of collaterals. So the output of one stage is consumed by another. Therefore there is a need for an IP development and validation tool which would amalgamate the tool operations and offer easier handling of all the collaterals.

In order to fit in all of these criteria there is a need for validation system. As part of this thesis, such an API Development and Validation System is taken through various aspects of improvement. The validation system regression aspect is switched from a legacy flow to advanced flow.

Multiple teams are working globally on IP development flow in parallel using multiple tools in each stage. So we need a central interface to manage the IP development flow. Through my project proposal a solution will be developed that will help to drastically overcome the shortcomings of the previous work methodology through the development of the new IP tool which will let the integrators focus on engineering decisions and not on tool or configuration problems. At the same time it will greatly reduce the integration time of any IP in the SOC environment and will empowers the IP to know that their delivered collateral will be healthy and complete in the customers environment.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

From the beginning of its era in the 1960s, ICs have evolved from few transistors to over a billion transistors. This rapid growth in IC fabrication technology in these four decades has been made possible by automation of various steps involved in design and fabrication of VLSI chips.

The process of converting the specifications of an electrical circuit into a layout is called the physical design. It is an extremely tedious and an error prone process because of the tight tolerance requirements and the minuteness of the individual components.

In IP/SoC (System on Chip) design there are a significant number of functional blocks or Intellectual Property. Development of each IP goes through different stages from requirement collection to fabrication. There are certain rules to check and verify each step. Multiple tools are used at each stage to verify the design and function of each IP. As changes are made in chips, so has their designing methods have changed.

The outlining of the chips by composing all the RTL from the earliest starting point, coordinating RTL obstructs into a best level plan, and doing level amalgamation took after by situation, never again works for chips which are complex[6].

Design reuse, the utilisation of predesigned and pre-verified centres is currently the foundation of SoC plan, for it is the main technique that enables immense chips to be composed at a worthy cost, regarding group size and plan, and at a satisfactory quality. The challenge for architects isn't whether to receive reuse, however how to utilise it effectively. As a consequence not only has the emphasis on quality of collaterals in each stage is changed, it also has had an effect on the overall way in which the development is done. Automation is the key element for the ease of design and also for time perspective. Such an automation system needs to evolve to cater to higher set of requirements.

1.1 Research Orientation

1.1.1 Research Motivation

The method for planning and working an electronic framework from the begin has been supplanted and System on-chip (SoC) has developed as an endless arrangement, where the major utilitarian parts of an entire final result are incorporated into a single chip. Officially outlined electronic parts to be reused for these useful segments constitute intellectual property (IP) centres. To be reused, an IP ought to have finish detail and legitimate documentation.

An SoC usually contains reusable IPs, embedded processors (a general purpose pro-cessor and multiple special purpose processors based on requirements) or controllers, memory elements (SRAM, ROM, etc.), bus architecture (for interfacing IPs and other components on SoC), mixed signal blocks, programmable blocks (FPGA), voltage level shifter, clock circuits, test architecture, and so forth. An SoC may easily be enhanced by integrating more IP components with it.

Therefore through this research on IP quality along with API development and valida-tion, we will ensure that IP functionality in system-on-chip (SoC) will be smooth and will help to reduce the integration time by using advanced algorithms for its implemen-tation will help help the system to be fully automated.

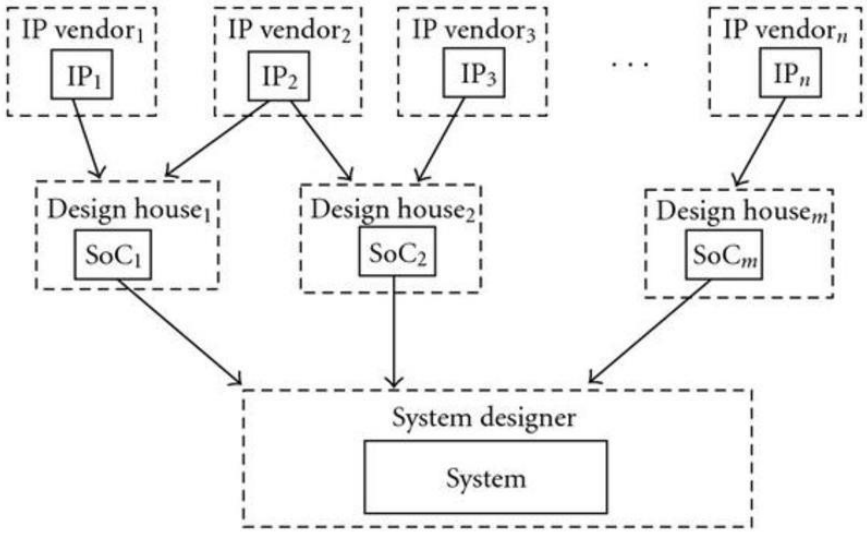


Figure 1.1: IP reuse in SoC environment

1.1.2 Key Research Area

(a) The IP Packaging today delivers content to integrator through local handoff. A mix of reusable and non-reusable or shared or non shared content is delivered. The IP documentation with critical integration requirements should be delivered but is often incomplete. A large effort in defining and maintaining correct file locations is required.

(b) The IP Integration today spends weeks locating and modifying current files needed by various flows and involves talking with the IP provider about various soft documented requirements. Therefore it encounters many issues not seen by testing that was done in the IP environment. A mix of reusable and non-reusable content at various stages of the project is also discovered. The integration happens in brute force manner, one flow at a time with iteration loops at SoC level. The delivered collateral is often programmatic or dynamic and very fragile.

(c) So, this kind of solution will help to drastically overcome the shortcomings of the previous work methodology through the development of the new IP tool which will let the integrators focus on engineering decisions and not on tool or config-uration problems. At the same time it will greatly reduce the integration time of any IP in the SOC environment and will empowers the IP to know that their delivered collateral will be healthy and complete in the customer environment.

1.2 Related Work

1.2.1 Moore's Law

In connection with integrated circuits (IC), Moore's Law depicts the pattern for the measure of transistors which can be put on a chip modestly. Contingent upon the source, the amount doubles each 18 to 24 months [3].
Figure 1 outlines the pattern of coordinating transistors on a single chip in the course of recent decades.

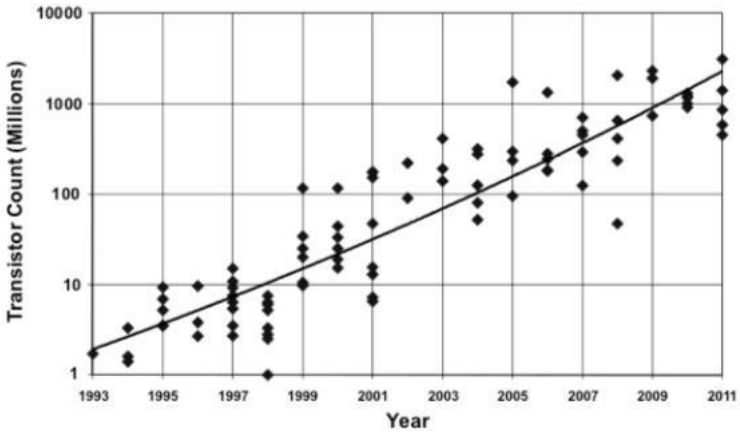


Figure 1.2: Moore's Law

Figure 2 plots productivity against complexity over a span of some years.

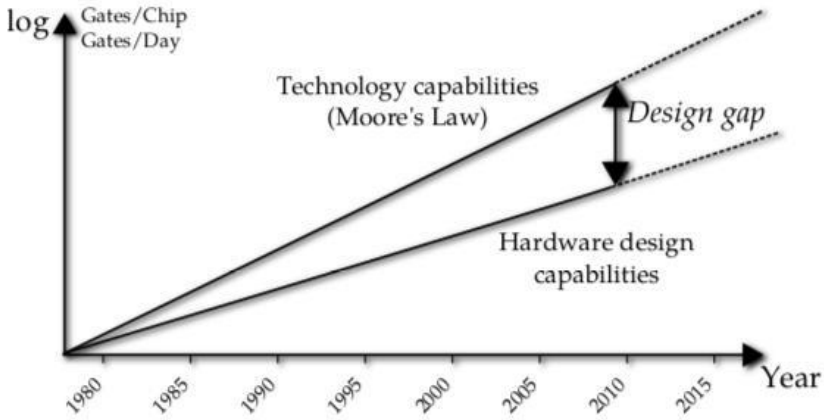


Figure 1.3: Design Gap

1.2.2 Methods for SOC Design Validation

This corresponds to a method examining design integrity of an SoC IC having multiple functional cores, and more particularly a method for SOC design validation in which design validation can be evaluated. An intended function of each core with timings, interface and an overall system operation of the SoC IC is there [4].

(a) Functional Verification includes compliance testing which is the testing to con-firm compliance to design specification, corner testing which is testing for com-plex scenarios and corner cases such as in the minimum and maximum conditions in voltage, temperature and process and random testing which is non-targeted testing that could detect very obscure bugs.

(b) Interface Verification requires a list of all transactions at each interface, thus, it is an impossible task because all possible test cases cannot be generated. Thus, limited verification is done. The second task is to confirm that the core interacts correctly for all data values and for all data sequence.

(c) Hardware Prototyping in which all design teams attempt to make the first silicon fully functional but more than half of the designs fail when put into the system for the first time. This is due to lack of system level verification or SoC level design validation.Therefore more real applications should be simulated. The available techniques are FPGA/LPGA and emulation.

(d) Timing Verification is even harder task than functional verification. Static timing analysis is the most widely available method today. It s performed on representa-tive net-list of cores that have been synthesized with various technology libraries. Also, the worst case timing scenarios are not exercised in simulation, since they are too complex and numerous to be identified properly by the design engineer.

1.2.3 Verification Technology Options

The possibility of verification is to ensure that the design meets the useful necessities as characterised in the functional specification. Verification of SoC devices takes from 40 to 70 percent of the aggregate advancement exertion for the design. A portion of the issues that emerge are how much verification is enough, what procedures and innovation alternatives to use for verification, and how to anticipate for and limit verification time. These alternatives can be broadly categorised into four classifications [5]:

(a) Simulation Technologies incorporate occasion-based and cycle-based simula-tors, transaction based verification, code scope, HW/SW co-verification,8 SoC Verification agents, for example, emulation, fast model frameworks, hardware modellers, and hardware accelerators.

(b) Static Technologies incorporate lint checking and static planning verification. This innovation does not require test bench or test vectors for completing out the verification.

(c) Formal Technologies incorporates detecting bugs early and the thorough idea of formal verification have been the main driving impetuses toward utilizing for-mal verification techniques. It doesn't require test benches or vectors for veri-fication. They hypothetically guarantee a very quick verification time and 100 percent scope.

Function	Simulation Timing Verification Yes	Static Timing Verification No	Formal Timing Verification No
Abstraction level	Behavioral, RTL, Gate	RTL, Gate	Gate
Functional Equivalence	Yes	Yes	No
Timing	Yes	No	Yes
Gate capacity	Low	High	Medium
Run time	<10 cycles	Medium	High
Cost	Low	Medium	Low

Figure 1.4: Comparision of Verification Technologies

1.3 Research Specification & Objectives

1.3.1 Research Target Specifications

(a) All the repository creation content should be encapsulated into activity enabling scripts which may also be run standalone, and which would double as on-demand repository-update scripts (which may be nested).

(b) Stage one is for each of these scripts would create default starter content for the generated files with only command-line input providing any variations. Contents which are expected to potentially vary should have default settings initialised in a hash.

(c) The starter contents of files should correspond to what current flow enabling cheat sheets document. Ideally flow owners would contribute to writing this starter content, but if not, then at least once we have gotten it ready, flow owners should review and confirm that the content has the correct configuration. They should note any content/file names which might change on different tool versions.

(d) Stage two of these scripts would initialised the default values from stage one, then read in existing files and override or append to the hash contents as needed, and then output the new files with updated content, or throw an error if something is found in the original file which cannot be merged into the hash.

(e) Depending on the script, the updated content will usually be either a merge of original file content and default file content, or just a processed/cleaned up form of the original.

1.3.2 Research Objective

The main purpose of this project is to develop an inclusive solution that will highly improve the complexities for IP Quality checking, IP integration and IP collateral management compared to how it is done today. Each IP undergoes different stages of development we call it as milestones. If IP lead found an error IP owners need to reverse all the steps, this will lead endless design loop and increase turnaround time and finally it will affect the scope and schedule of the project.

Therefore the development of this tool will:

(a) Let the integrators focus on engineering decisions and not tool / configuration problems.

(b) It will greatly reduce the integration time of any IP in the SoC environment and overall goal is to reduce the integration time from 8 weeks to 2 weeks during 0.5 and 1 week during 0.8/1.0.

(c) It will also empowers the IP to know that their delivered collateral will be healthy and complete in the customer environment.

(d) It will checks the bill of materials provided by the IP to ensure compatibility with a variety of flows (with required flows as appropriate to milestone).

(e) It will provide a consistent view for each IP before it gets integrated and will also reduce the integration iterations loops for each IP because the IP being integrated will be clean by itself in the customer environment.

(f) It will enable integration automation in a compositional manner according to the requirements and also aids tool flow management deltas between provider/consumer.

(g) It will provide better tractability of the IP/sub-IP, IP quality and integration time and will push many of the discoveries back to the IP provider as formal rules.

(h) It can work for any IP (internal/external) including VIPs and will arrive to a uni-form solution to qualify reusable IP collateral for all front end activities.

1.4 API Layer Proposal

Coding begin on the metadata API long before the metadata schema is finalised. Tasks which are not gated by the final metadata schema actually include most of the API functions:

(a) Creation of Test Driven Development unit tests.

(b) All development work on the API will have an equivalent test created before coding the implementation, and the test will be updated, or new tests added as appropriate, as each new feature is added or updated. Each function will do one simple task, so they would be easy to keep very short. Tests are run and confirmed if they are pass.

(c) Creation of basic data structure and possibly package structure used by the API, used to store the data in memory which the metadata represents. Metadata schema structure does not need to control API data structure.

(d) The structures wil be very independent as long as they can both represent the same information. Decision whether one wants to create an instance of a Perl package for each metadata or if want to have all content visible in one huge data structure.

(e) Creation a static lookup table of the type of data to the metadata specifications or possibly an array of possible metadata specification locations if we want to allow for potential backward compatibility to older specs on the readers. Then having all appropriate functions which we code, both in TDD tests and metadata API, using this lookup table. This way any metadata spec changes can be incorporated with minimal effort.

1.5 Architecture Flow

(a) This tool architecture consists of many Perl files for each layer of implementation. At first the data is given to us which is in a form of DTD which is not well formed. Using XMLPAD the data contained in the DTD is converted into its equivalent XML and then that XML is validated against the DTD to check if there are no syntax errors and the XML is well formed. After the data is cleaned it is now used at the first level of the architecture.

(b) At first the data is read from the XML in a Schema read Perl file where the Perl packages play a very important role. There are Perl predefined packages that help in reading the XML file and retrieving the data i.e. LibXML. After the reading is done then a Write Perl module is coded to check whether writing the new elements and attributes is happening in the new XML file.

(c) At the second layer there is a static mapping Perl Module that maps the schema passed as XML to the data structure. It contains a static mapping table which maps the name of multiple IPs with the schema.

(d) At the top level there is a metadata Perl module which contains multiple Perl modules of each data structure. It also contains reader and writer specific and generic APIs.

(e) The creation of TDD unit tests is done next. All development work on the API have an equivalent test created before coding the implementation, and the tests are updated, or new tests are added as appropriate, as each new feature is added or updated. Keeping all functions to under 10 lines without creating super long lines of spaghetti code.

(f) Creation of basic data structure and possibly package structure used by the API is done which is used to store the data in memory which the metadata represents.

(g) The default values for all data structure elements is expected in an ideal repository case with different combination functions used to construct (or deconstruct) more complex content.

1.6 Thesis Organisation

(a) Chapter 1 is the introduction which explains about that there is a need to bring a change in the design of IP/SoC because as changes in chips are there, so their designing methods. It also explains about why the research is conducted and what is the main key research area in this thesis. An overview of the literature survey is also presented by mentioning various methods that were used for SoC design validation along with different verification technology options. At the end of this chapter, a detailed summary of the major objective of this project is explained along with the proposal of this project.

(b) Chapter 2 defines the methodologies that were implemented in this project. A brief description of the IP Quality Checklist and Audit Tool is presented. It talks about various implementation methods carried out in this project like the data that is used is in the form of XML data. Using XMLPad, we first converted the data into its equivalent DTD and then validate the DTD. A very fast and simple approach for testing is used i.e. Test Driven Development along with implementing OOPs concept in Perl using Perl Moose.

(c) Chapter 3 is all about tool purpose summary which briefs about what are the main focus areas in this project. It tells about the metadata and API layer sum-mary which in turn briefs about the major guidelines to be followed and what are the key factors that will be taken care of for its creation. New repository setup description is also provided which along with instructions also tells about its advantages.

(d) Chapter 4 provided detailed information about the tool specifications, its inter-face definition and its objectives. Metadata and API layer specifications mainly briefs about metadata file content guidelines and what are its major specs. New repository setup specification explains the procedure for a new repo setup by cre-ating a new fully functional repo from scratch and what users are expecting from default.

(e) Chapter 5 is about use cases which the tool follows i.e the effects when the tool is run by the IP provider, when the tool is run by the ship and when the toll is run by the IP consumer. It also describes about various use cases of metadata which mainly involves the IP provider tasks and finally talks about the API layer use cases which focuses on user point of view where the user adds are collateral to their design, remove them from their design or the tool itself adds a new flow. A brief overview of new repo setup use cases are also given which is from the IP owner perspective.

(f) Chapter 6 is for the results and its analysis. It displays various screenshots of the major functionality of the tool, its overall flow, pass results and fail results.

(g) Chapter 7 explains about the conclusion drawn from this project in detail and what can be the future scope of this project to make the system more flexible and helps to improve the tool run time and efficiency by making it automated.

CHAPTER 2

METHODOLOGIES

(a) Quality of SoC is dictated by the incorporated IP blocks. To guarantee quality we have to screen the IP execution and to investigate the aftereffects of IP execution. Every IP configuration stream contains arrangement of instruments, device inputs and expected yields will shift in light of prerequisites, there plan robotisation will assume a critical part to guarantee IP execution smoothly.

(b) IP group require a framework to guarantee quality checkpoints for IP deliverables. Each phase of IP advancement should check a portion of the compulsory checks through this framework, evaluators will review the outcomes and will guarantee quality before convey IP to SoC.

(c) Each apparatus is running in view of specific principles, these run will differ start-ing with one IP then onto the next. Any variety of these principles will brings about infringement. In light of extent of these infringement in the task prereq-uisite situation, IP proprietors can postpone some these infringement. At the IP conveyance, just consider net infringement. IP group require an interface to check nature of IP in light of run results.

(d) Quality checks are assembled for various sub bunches on various phases of IP advancement stream. Checks will differ starting with one development then onto the next, same starting with one venturing then onto the next. So approach group requires an interface to oversee checks, need to include new checks and change existing checks.

(e) Users are running IPs of various tasks through dash interface. IP group require an interface to dissect instrument run time, memory utilization, and plate use. From these outcomes , IP group will ready to recognize the instruments which expending more assets and executes longer time , these will prompt enhance the

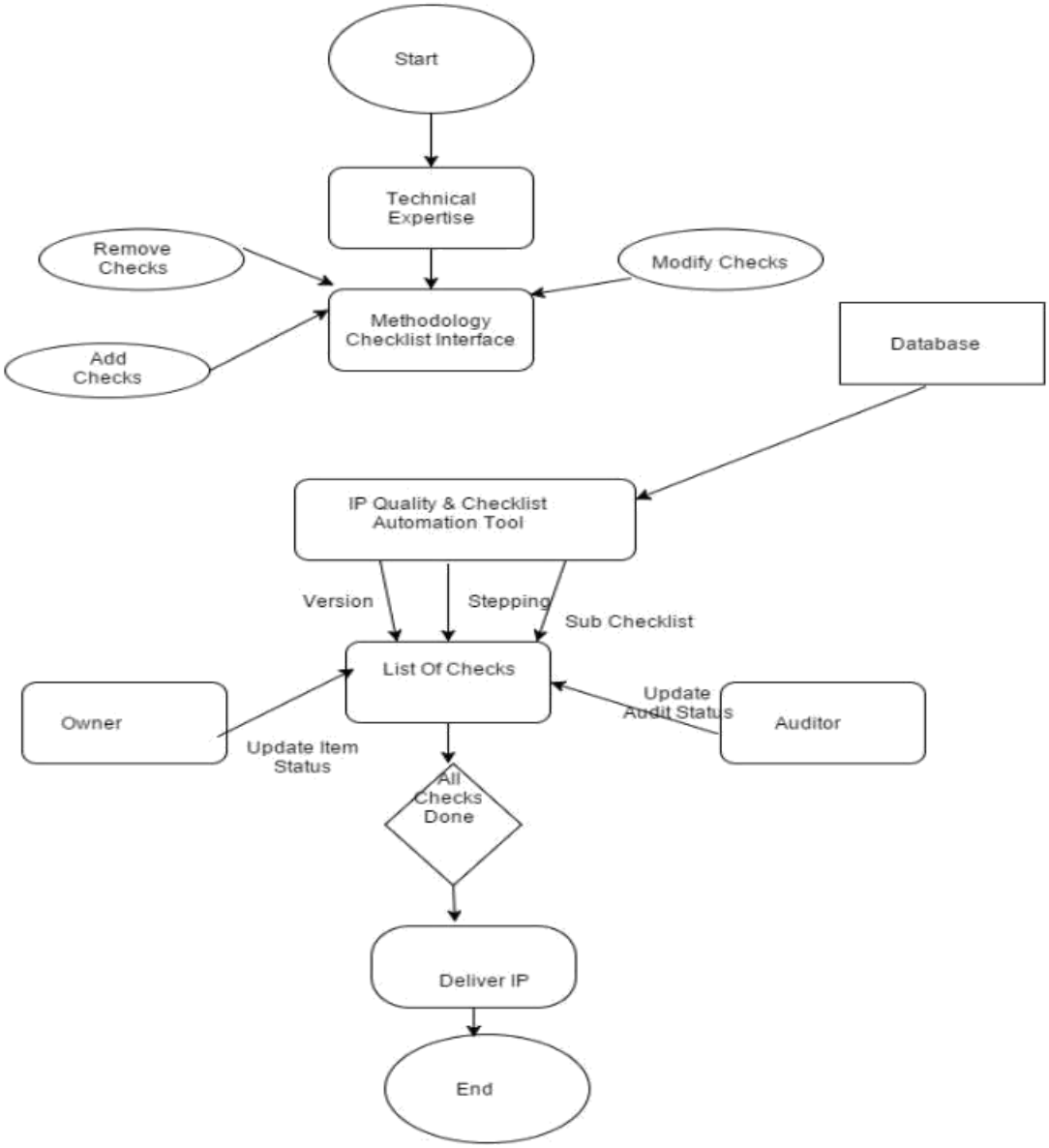


Figure 2.1: IP Quality Checklist And Audit Tool Flow

apparatus execution, in this manner decrease pivot time and enhance IP quality and diminish cost of IP development.

IP Quality Checklist And Audit Tool This tool will ensure quality check points for IP deliverables. Each IP should undergoes auditing for all mandatory checks. Features designed for the system as follows.

(a) Checkpoints : there is an option in DART interface to view checklist for each block. Access of checklist is restricted by settings in block editor. We have to enable flags to see the checklist for each block.When user clicks view checklist

page, will open a GUI which provides options to select sub checklist, stepping and version.

We are auto filling sub checklists, stepping and version if there is pre filled data. User can select multiple sub checklists but only one stepping and version. Block Types and milestone is configured in block editor. We need to set some questions in order to set block type. milestone, block type, sub checklist, stepping, version are mandatory to view checks.

(b) Auditing : is responsible to audit all item statuses filled by owner. Auditor can fill audit status, by selecting one of the options, AR, or Approve from the list. AR means owner need to re-check the item. Approve means Item is working fine in IP. If Auditor changes audit status to AR item status will automatically goes to not done. Owner need to re-check the item. There is an option Approve All for each acronym. If owner status is filled Auditor can approve all items under particular acronym by a single action. Auditor approval summary will give a complete picture of checklist quality for a particular IP.

(c) Freezing : IP lead can freeze the checkpoints, this is important because for each IP there will be a schedule time to deliver, user should check the quality before the delivery of IP to SoC. Once IP lead freeze the checks, all action will be disabled for an IP.

(d) User Support : There is an option to notify auditor, to chat with auditor or can send email to auditor. There is an option to contact methodology team for support.

2.1 Design

A powerful design based on block methodology needs a broad collection of reusable blocks, or macros. The developers of these macros must, thusly, utilise an outline strat-egy that reliably creates macros that are reusable [1]. This design reuse technique depends on the accompanying principles:

(a) The macro should be extremely easy to incorporate into the general chip design.

(b) The macro must be so robust to the point that the developer needs to perform basically no utilitarian check of macro internals.

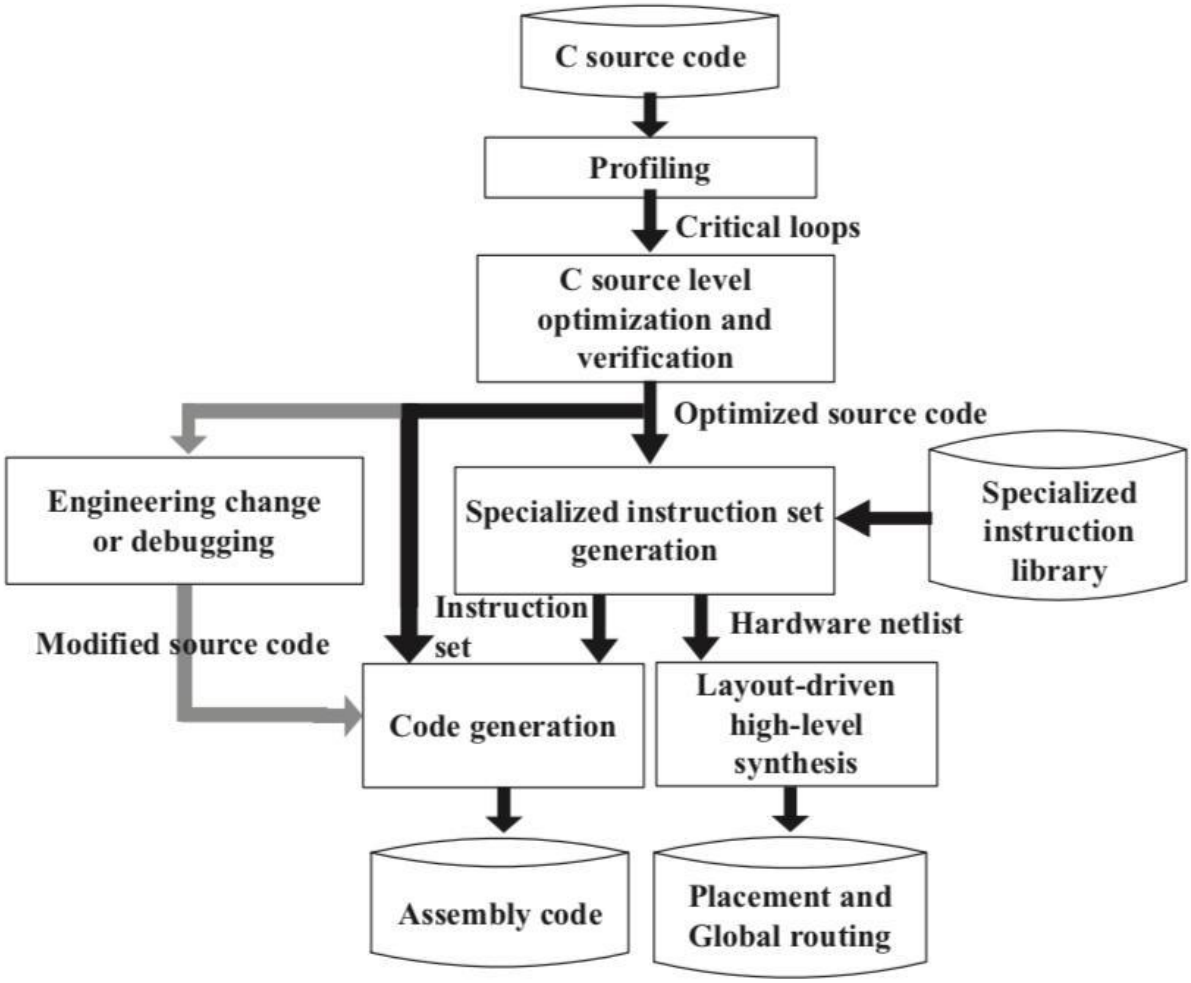


Figure 2.2: Design Flow

Reuse design presents particular difficulties to the design team. In any case, to be used again, a plan should first be used again: a robust and correct design.

(i) Good documentation with great code with exhaustive commenting.

(ii) Well-composed confirmation situations and suites strong scripts.

(iii) A alternate way may seem to abbreviate the plan cycle, yet in actuality it just moves exertion from the outline stage to the combination period of the project. At first, following great outline practices may appear like a superfluous weight. Be that as it may, these methods speed the outline, confirmation, and investigate procedures of any venture by lessening emphasis through the coding and check loop.

Additonal to the requirements above for a powerful design, there are some extra prerequisites for a macro of hardware type to be completely reusable. The macro should be [2] :

(i) Designed in such a way that it can tackle a general issue and be utilised as a part of numerous technologies.

(ii) Designed to reproduce with different types of simulators with benchmarks based interfaces.

(iii) Independently verified for the chip in which it will be used to an high state of confidence.

These prerequisites increases the amount of time and exertion required to the advancement of a macro, but they give the critical advantage of making a reusable macro.

2.2 Implementation

2.2.1 Test Driven Development (TDD)

Test-driven development (TDD) is a development technique where we first write a test that fails before we write new functional code. TDD is being quickly adopted by agile software developers for development of application source code. The steps of TDD are :

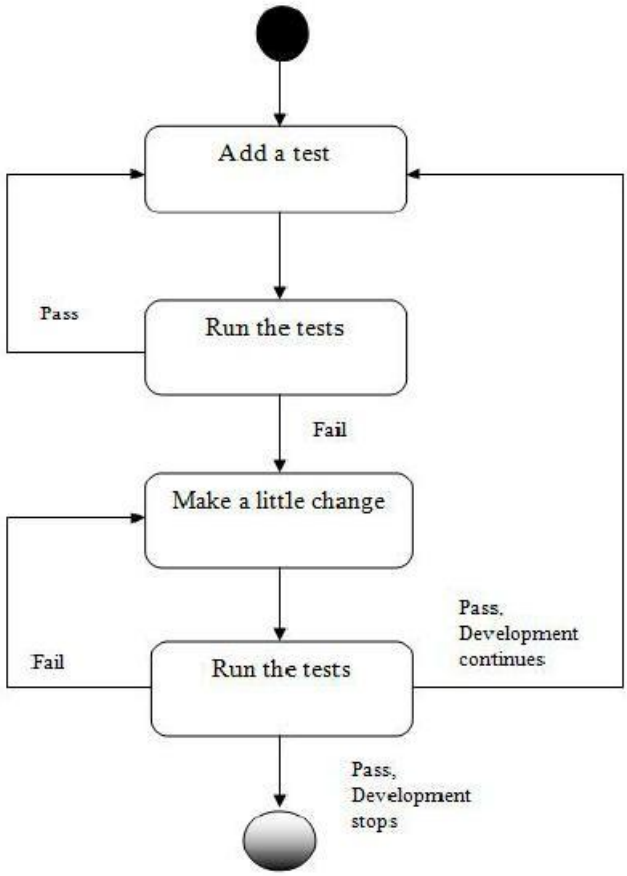


Figure 2.3: TDD Flow

(a) Add a test is the initial step where each new element starts with composing a test. Compose a test that characterises a capacity or enhancements of a capacity, which ought to be extremely concise. This is a separating highlight of test-driven advancement as opposed to composing unit tests after the code is composed. It influences the engineer to centre around the prerequisites previously composing

the code.

(b) Run all tests and check whether the new test fails approves that the test tackle is working effectively and demonstrates that the new test does not go without requiring new code on the grounds that the required conduct as of now exists, and it decides out the likelihood that the new test is defective and will dependably pass.

(c) Write the code that makes the test pass. The new code composed at this stage isn't impeccable and may, for instance, breeze through the test in an inelegant way. That is satisfactory in light of the fact that it will be enhanced and sharpened in Step 5.

(d) Run tests if all experiments currently pass, the developer can be sure that the new code meets the test prerequisites, and does not break or debase any current highlights. On the off chance that they don't, the new code must be balanced until they do.

(e) Refactor code includes that duplication must be evacuated. Question, class, mod-ule, variable and technique names ought to plainly speak to their present reason and use, as additional usefulness is included. By constantly re-running the exper-iments all through each refactoring stage, the engineer can be sure that procedure isn't changing any current functionality.

(f) Repeat includes beginning with another new test, the cycle is then rehashed to push forward the usefulness. The span of the means ought to dependably be little, with as few as 1 to 10 alters between each trial. In the event that new code does not quickly fulfil another test, or different tests flop startlingly, the software engineer ought to fix or return in inclination to intemperate debugging.

As per the functionality implemented in this project, TDD is used mainly for testing each perl module as shown in the below figures.

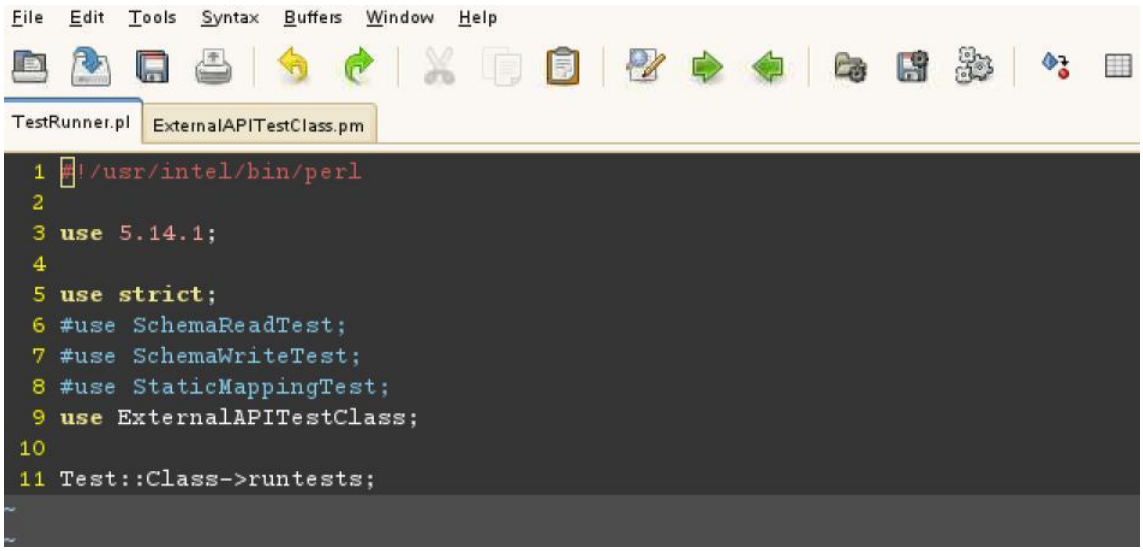


Figure 2.4: TDD example

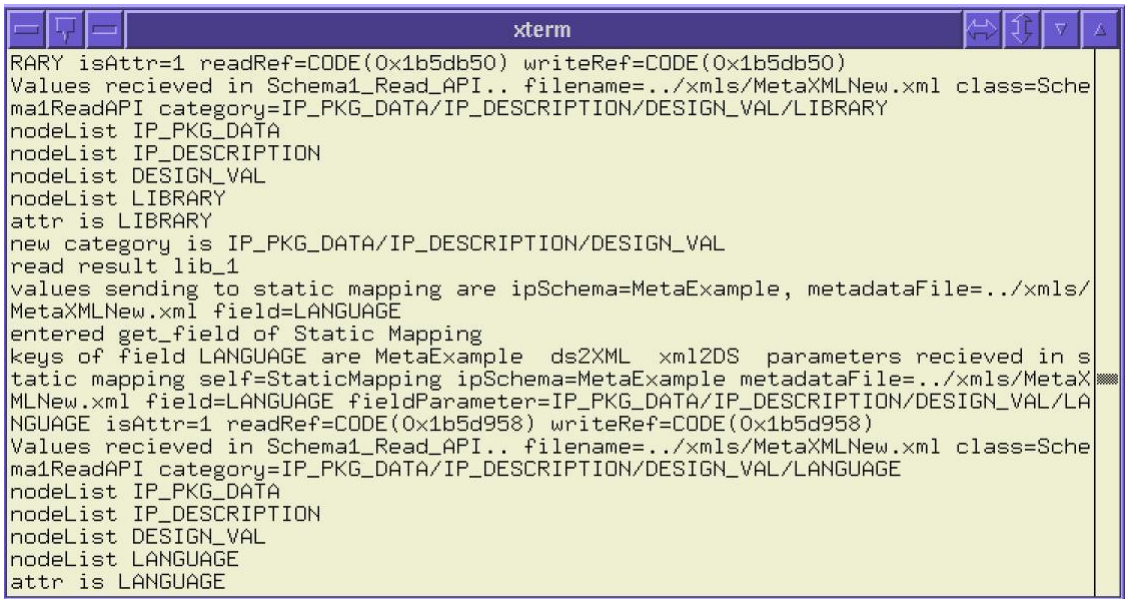


Figure 2.5: TDD output example

2.2.2 XML Parsing of data

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. In this project, XML schema is required at the first level of the implementation where a sample XML is given and using Perl language or inbuilt tool, it need to be converted into an XML and then fetched to the metadata layer for reading and writing.

XML processors are named approving or non-approving relying upon regardless of whether they check XML reports for legitimacy. A processor that finds a legitimacy mistake must have the capacity to report it, yet may proceed with ordinary processing. XMLPad apparatus is utilised as a part of this task for parsing of XML information and checking whether it is substantial or not.

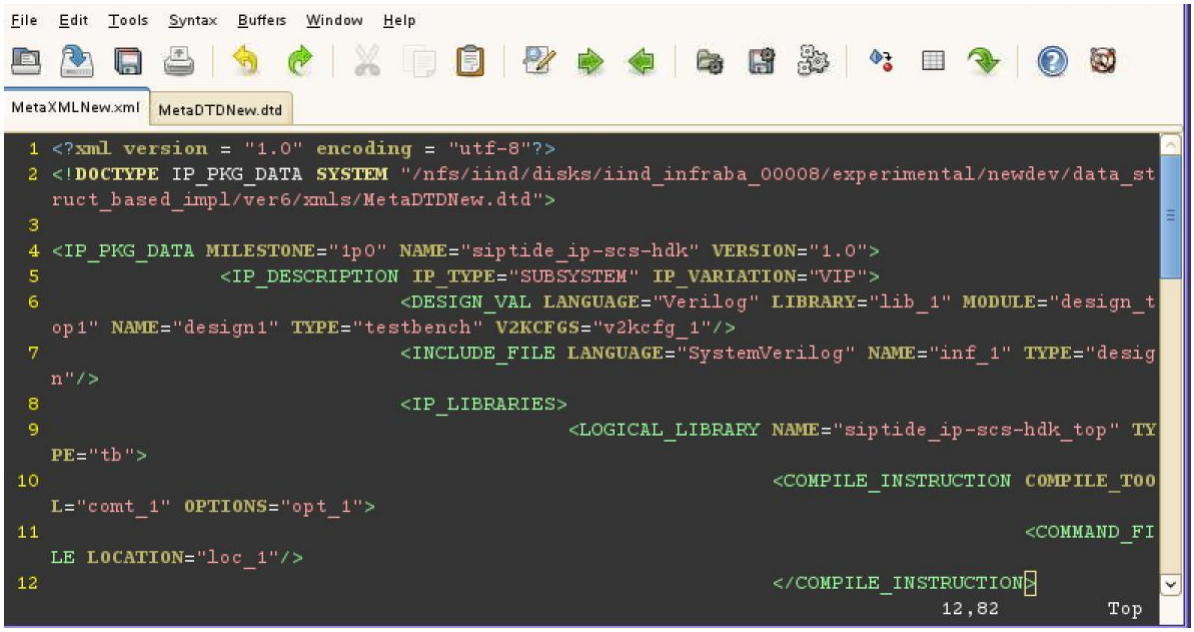


Figure 2.6: XML Example

2.2.3 DTD Conversion

A Document Type Definition (DTD) defines the legitimate building squares of a XML report. It characterises the archive structure with a rundown of legitimate components and qualities. A DTD can be announced inline inside a XML report, or as an outer reference.

The affirmations in the inward subset frame some portion of the DOCTYPE in the archive itself. The statements in the outer subset are situated in a different content file. The principle errands XML DTD performed in this venture is to confirm that XML information is valid.

In this task, the information is given as a XML information. Utilising XMLPad, we change over the XML information into its identical DTD and after that utilisation check whether the information is legitimate or not.

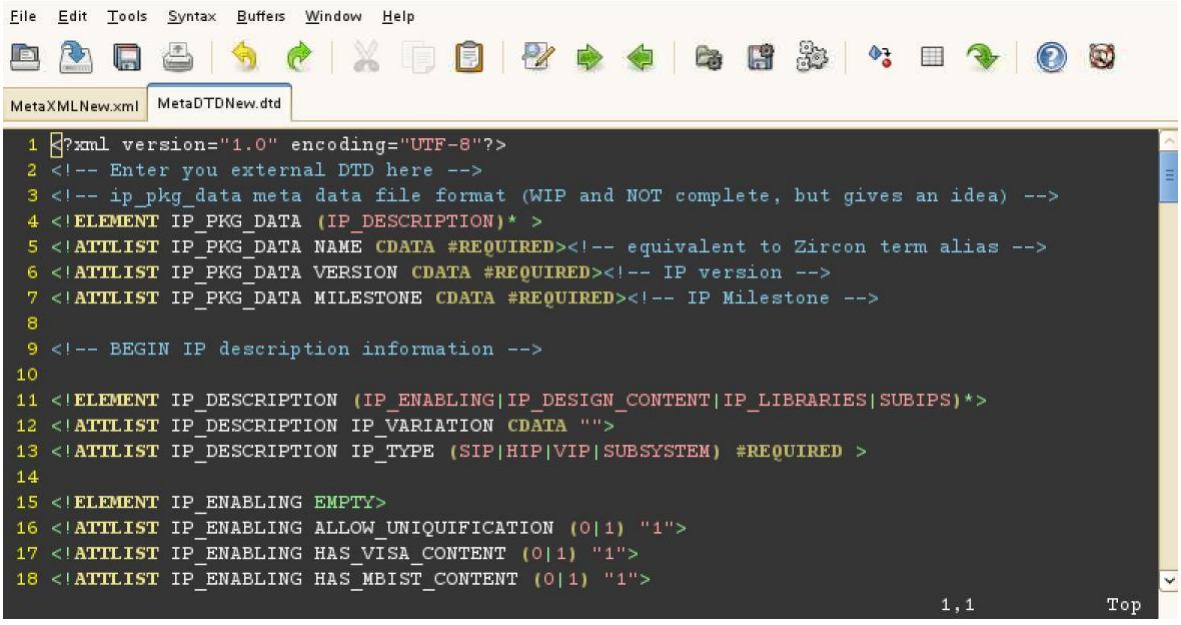


Figure 2.7: DTD Example

2.2.4 Perl Moose

Perl default object framework is insignificant however adaptable. Its syntax is some-what inconvenient, and it exposes how an object framework works.

Perl Moose is a total object framework for Perl. Its a total distribution accessible from the CPAN which is not a part of the centre language but worth installing and using regardless. Moose offers both a less complex approach to utilise a object framework.

(a) Classes A Moose object is a solid instance of a class, which is a format portraying information and behaviour particular to the object. A class belongs to a package which provides its name:

```
package Cat
{ use Moose;
}
```

(b) Methods is a function related with a class. Similarly such that a function belongs to a namespace, a method will belong to a class. package Cat {

```
use Moose;
sub meow { my
$self; say 'Meow!';}
}
```

Every object can have its own different data. Methods that read or compose the information of their invocants are instance methods, they rely upon the nearness of a proper invocant to work correctly.

(c) Attributes Every Perl object is one of a kind. Objects can contain private infor-mation associated with each unique object often called properties, instance data, or object state. We characterise an attribute by announcing it as part of the class: package Cat {

```
use Moose;
has 'name', is => 'ro', isa => 'Str'; }
```

Moose sends out the has function for you to use to announce an attribute.

CHAPTER 3

TOOL PURPOSE SUMMARY

3.1 Summary

(a) Let the integrators focus on engineering decisions and not tool / configuration problems

(b) Greatly reduce the integration time of any IP in the SOC environment.

(c) Overall goal is to reduce the integration time from 8 weeks to 2 weeks during 0.5 and 1 week during 0.8/1.0.

(d) Empowers the IP to know that their delivered collateral will be healthy and com-plete in the customers environment.

(e) Checks the bill-of-materials provided by the IP to ensure compatibility with a variety of flows (with required flows as appropriate to milestone).

(f) Provides a consistent view for each IP before it gets integrated.

(g) Reduces the integration iterations loops for each IP because the IP being inte-grated will be clean by itself in the customer environment. The only bugs should be due to integration decisions (connectivity, behaviour of surrounding collateral, or tool overrides from baseline).

(h) Better tractability of the IP/sub-IP, IP quality and integration time.

(i) It can work for any IP including VIPs.

(j) Arrive to a uniform solution to qualify reusable IP collateral for all front end RTL activities.

(k) Potential method for creating precompiled IP binaries.

3.2 Metadata & API Layer Summary

(a) Provide a tool-readable specification of bill-of-materials with all of the infor-mation that the IP provider needs to provide.

(i) Formalise the recording of IP packaging data that is required for integration, enabling integration to be compositional and automated.

(ii) Content must be data-centric whenever possible and focus should be on the data provided, and not on the specific tool which will consume that data.

(iii) Content should strive to avoid Intel-specific files and formats whenever pos-sible. An idealistic file would contain data and pointers to industry-standard file formats only.

(iv) Eliminate the new-drop re-integration of old IP collateral which is no longer active.

(b) Provide an API for interacting with this tool-readable specification.

(i) Support both read & write operations.

(ii) Support uniquification of data and namespaces.

(iii) Support recursive loading & processing of metadata files.

(iv) Enable tracking and comparing all versions of an IP across all nested layers of IP.

(v) Enable integrators to focus on IPs which are directly integrated.

(c) Enable scripts which read existing IP collateral to populate metadata content and/or check it for consistency.

(d) Enable templates and/or scripts which use this API plus tool-specific instructions to generate collateral for specific tools.

3.3 New Repository Setup Summary

(a) Create a repo with simple design & validation content, which has all the required flows.

(i) Simple starter design & validation content is needed to enable all of the various tool flows to both run & pass.

(ii) Designers & validators can focus on coding & debugging RTL, and not need to become tool experts to get a tool working for the first time in their repo.

(iii) Known-good starting point reduces the scope of the problem when some-thing breaks.

(iv) Once the tool is completing a run, existing training and BKMs (delivered by tool owner) will assist the designer/validator with debugging the problem.

(b) Reduce the maintenance cost associated with upgrading flows and adding new tools/flows.

(i) Provide an update repository script or methodology that call pull in desired tool changes.

(ii) Designers do not need to focus on tool maintenance. They will still need to address fallout with newer tool versions and flows as they update their repository through the script mentioned above.

CHAPTER 4

API SPECIFICATIONS

4.1 Specifications

(a) Test all of an IP's reusable collateral from an IP and SOC agnostic environment. Use the IP-provided metadata to run all the required flows on the IPs content to ensure no dependency upon IP-specific environment assumptions and also ensure no dependency upon IP-specific tool/flow assumptions.

(b) Tool Interface definition

Required: Name, Path and Metadata of IP to be tested.

(c) Test the IP's collateral from a clean environment.

Can't allow contamination by user's currently sourced environment. It might also need to report if settings from user's login scripts contain IP-specific environment settings to prevent accidental contamination and also can't allow access to group-restricted content which would not be customer visible.

(d) Run all applicable flows using default/automated settings, with metadata as the only means of making any alterations.

(e) Flows must run cleanly to the extent possible from a throw away repo.

All sim build compile flows are capable of running cleanly given correct IP con-tent. All static checks are capable of running cleanly given correct IP content. Uses new repository setup to create the temporary repository where testing will occur.

(f) This repo is intended to be deleted once testing is complete. The new repository setup is called with the ip to be tested by the tool as the sole explicit subIP if the metadata is complete and IP collateral is coded correctly then this will be sufficient even for testing IPs with deeply nested subIPs.

4.2 Metadata & API Layer Specifications

(a) IP-provider based view records information about the IP as seen by the IP provider. Each IP should have their own metadata file. The only information in an IP's metadata file about the subIPs they used should be:

  (i) The unique name of the subIP, the path to the subIP, and the version of the subIP.

  (ii) This enables IPs to easily change subIPs without needing to alter their own metadata file significantly.

(b) Ability to specify all types of collateral that is delivered by IP i.e current for-mat is a good start in terms of collateral specified. The metadata should only contain deliverable items from the IP and nothing else. If anything else is needed to be able to measure the metadata quality, it has to be given on the side (from customer or pseudo-customer).

(c) Human readable

  (i) The user of functions/macros should not be allowed unless the definition is shown in the metadata file. The specification has to be non-programmatic. This means that we are not allowing if/then/else/while for-loops, allow ref-erences to other existing entries. We want to allow references to other meta-data as the design is hierarchical and be able to add IRR information in the metadata.

  (ii) Support multiple configurations of the same IP which needs to come with minimal duplication of data.

(d) Machine-readable

  (i) Multiple languages can be supported but exact language is less important than content

  (ii) Current metadata language is Perl.

  (iii) The future language is XML. This is to be compatible with the Infra Meta-data. Ideal metadata language for external IPs is IP-XACT (an industry

standard form of xml-based metadata) with no Intel-specific add-ons.

(iv) Non-reusable information may be used by checker scripts or by IP develop-ment, but should act only as documentation for an IP customer.

(v) Reusable information should be able to be plugged in as-is during a vanilla customer integration.

(e) Must be able to be read recursively. IPs should need to record only direct subIPs, and not need to know details of nested subIPs and also should have ability to override (when necessary) the version and path of nested subIPs to prevent collisions.

(f) Must be modular and extensible. IPs should not have to have their metadata file match the metadata reader version.

(g) New metadata readers must support older metadata files (backward compati-ble metadata handling). Old metadata readers must be able to report fields which are not recognised without erroring (to enable forward compatible metadata han-dling).

4.3 New Repository Setup Specifications

(a) Create a new, fully functional repository from scratch

(i) Start from an empty repository.

(ii) Must enable all the flows to be functional without requiring any manual modification which includes creating any required tool input & configura-tion files.

(iii) Simplest starting RTL content with simple single module design containing a single flop which has clock, reset, & data (bus) pins.

(b) Ideally all tool files & settings would come from the related tools so in that way they stay in lock-step with the most recent changes and recommendations for that tool. This will require tool owners to maintain templates and/or scripts to enable populating these files, but it should make it significantly easier for users to upgrade tool versions and maintain good settings.

(c) Some users want all the flows to run by default. This is something that is easily user configured after repository creation, but the conflicting requests are concerning the default behaviour of a newly created repository. The current im-plementation is to have all flows run by default so that any issues on side flows would be visible quickly, rather than only when someone thought to run that side flow.

CHAPTER 5

API TOOL USE CASES

5.1 Use Cases

(a) The tool is run by an IP provider regardless of IP-type, so includes at least hip, sip and subsystem providers At any point during development to check bill of materials readiness prior to packaging and after packaging to check that nothing was accidentally left out of a package or altered/moved during packaging in a way that contradicted the bill of materials.

(b) The tool is run by ship to check quality of the bill of materials content in the context of the project/domain ship ran from.Ideally should be run done after any ship-introduced pruning of directory structure occurs, as ship-time pruning is not visible to any tool runs done by an IP provider.

(c) The tool is run by any IP consumer prior to (re)integration of an IP, to check that packaged/dropped collateral is self-consistent and is not improper. When changing tool or global IP versions, to check compatibility of individual IPs in-cluding all nested subIPs and potential run of quality check post integration of IPs, possibly for multiple IPs in parallel.

5.2 Metadata Use Cases

(a) IP provider creates initial metadata file Users create metadata Perl file to gen-erate initial metadata content for their IP based on existing IP content. Edits metadata file to fix any gaps in metadata content which could not be automati-cally determined and commits metadata to their development repo as persistent design collateral. Script to update sections of metadata would be very helpful.

(b) IP provider maintains metadata content over time

(i) User might run a "metadata updater" script to merge content from IP files & settings into existing metadata content.

(ii) User might run a "collateral creation" script to generate content from meta-data file.

(iii) User might run if a new tool/flow is required. After initial creation of the new tool/flow settings files, user might opt to leave these files as generated from metadata or user might opt to make these files "source" files to be used when doing future metadata updates.

(iv) User might run both of the above when integrating (or re-integrating) any subIP.

(v) User would manually provide subIP name, path, and metadata file to be integrated.

(vi) User might provide override settings for subIP metadata content if needed (but does not touch provided subIP metadata file).

(vii) User might run collateral creation before running a tool/flow (i.e., run file creation whenever they plan to run that tool/flow, if they do not want to retain those generated files in their development repo and risk accidental modification).

5.3 Metadata API Layer Use Cases

(a) User adds new collateral to their design and updates metadata to reflect the addition as needed. This could be a manual addition if there is no source ex-isting in the metadata for the new collateral or could be an automated addition (i.e. running a metadata updater script) if it was made within an existing source component.

(b) User removes collateral from their design and updates metadata to reflect the removal as needed. This could be a manual removal if there is no source existing in the metadata for the dead collateral or could be an automated removal (i.e. run-ning a metadata updater script) assuming the change was made within an existing source component.

(c) The tool owner adds a new tool/flow or updates an existing one if necessary the tool owner adds/updates metadata schema and API, including adding default settings appropriate to the simplest IPs for all schema additions. However, when-ever possible additions to metadata should be crafted as data-centric, even if the data is only currently needed by one tool.

5.4 New Repository Setup Use Cases

File creation scripts used from within new repository setup to populate a brand new repository are the same scripts which may be used by an existing repo to regenerate files for a given flow.

(a) An IP owner runs new repository setup to create a new repo for their IP which might happen because IP is actually a new IP which has not yet been developed or because IP wants to clean house or to be used to create a wrapper IP to be paired with an external IP so that it won't be touched this way, but can have Intel-specific collateral created on its behalf within this new wrapper IP.

(b) Tool owners can run new repository setup as an aid for testing their tool. To help create small test cases to debug issues seen in their tool, to test population script changes prior to releasing those scripts to users and to check baseline tools compatibility with their tool.

(c) Tools can create a repository for automatic integration of subIP. It uses new repository setup to create a simple wrapper IP around the IP to be tested. A uniquification tool might use new repository setup to run uniquification and then test that post-uniquification all provided reusable collateral can be applied.

CHAPTER 6

RESULTS & ANALYSIS

(a) Reading the XML data using XML LibXML perl package in the function

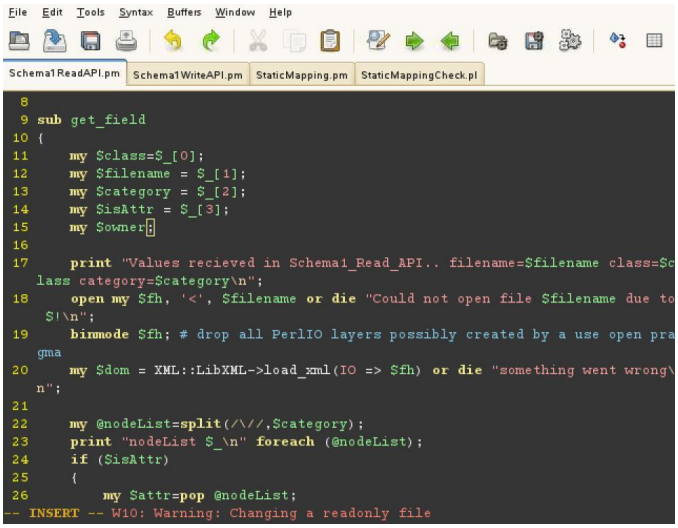


Figure 6.1: Reading XML File example

(b) Writing of the XML file into a new output file which can be a text file or .xml file

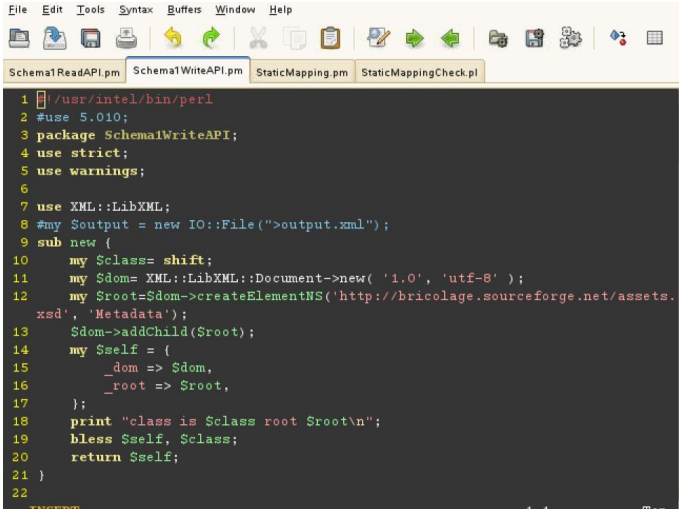
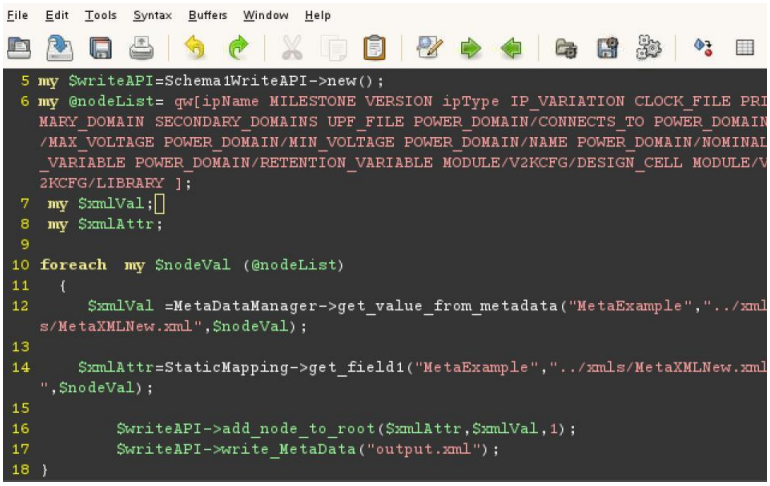


Figure 6.2: Writing XML file example

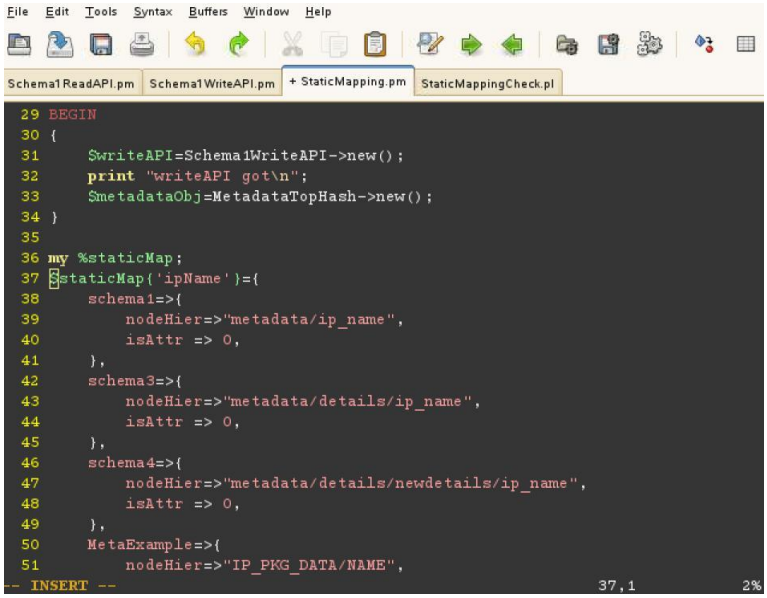
(c) Perl module to check the functionality of write file.



```
File  Edit  Tools  Syntax  Buffers  Window  Help
 5 my $writeAPI=Schema1WriteAPI->new();
 6 my @nodeList= qw[ipName MILESTONE VERSION ipType IP_VARIATION CLOCK_FILE PRI
   MARY_DOMAIN SECONDARY_DOMAINS UPF_FILE POWER_DOMAIN/CONNECTS_TO POWER_DOMAIN
   /MAX_VOLTAGE POWER_DOMAIN/MIN_VOLTAGE POWER_DOMAIN/NAME POWER_DOMAIN/NOMINAL
   _VARIABLE POWER_DOMAIN/RETENTION_VARIABLE MODULE/V2KCFG/DESIGN_CELL MODULE/V
   2KCFG/LIBRARY ];
 7  my $xmlVal;
 8  my $xmlAttr;
 9
10 foreach  my $nodeVal (@nodeList)
11   {
12      $xmlVal =MetaDataManager->get_value_from_metadata("MetaExample","../xml
   s/MetaXMLNew.xml",$nodeVal);
13
14     $xmlAttr=StaticMapping->get_field1("MetaExample","../xmls/MetaXMLNew.xml
   ",$nodeVal);
15
16         $writeAPI->add_node_to_root($xmlAttr,$xmlVal,1);
17         $writeAPI->write_MetaData("output.xml");
18 }
```

Figure 6.3: Checking Writer Module example

(d) Mapping of XML data to data structures is done through this static mapping mod-ule.



```
File  Edit  Tools  Syntax  Buffers  Window  Help
Schema1ReadAPI.pm | Schema1WriteAPI.pm | + StaticMapping.pm | StaticMappingCheck.pl
29 BEGIN
30 {
31     $writeAPI=Schema1WriteAPI->new();
32     print "writeAPI got\n";
33     $metadataObj=MetadataTopHash->new();
34 }
35
36 my %staticMap;
37 $staticMap{'ipName'}={
38     schema1=>{
39         nodeHier=>"metadata/ip_name",
40         isAttr => 0,
41     },
42     schema3=>{
43         nodeHier=>"metadata/details/ip_name",
44         isAttr => 0,
45     },
46     schema4=>{
47         nodeHier=>"metadata/details/newdetails/ip_name",
48         isAttr => 0,
49     },
50     MetaExample=>{
51         nodeHier=>"IP_PKG_DATA/NAME",
-- INSERT --                                   37,1          2%
```

Figure 6.4: Example of Static Mapping of XML data to Data Structure

(e) A check module to test the functionality of static mapping.

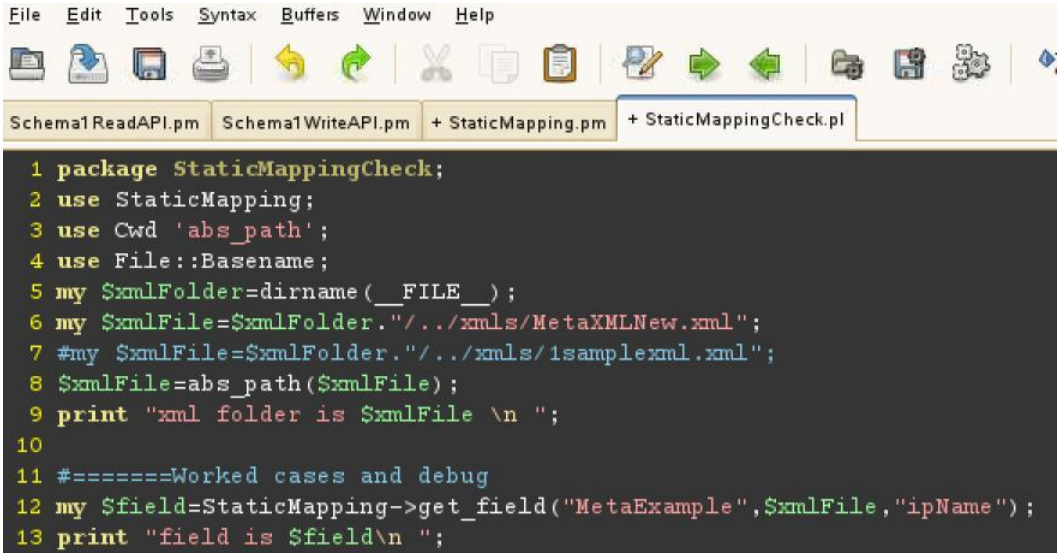


Figure 6.5: Static Mapping check example

(f) Metadata manager is the main module of the whole tool that access the read write and mapping modules.

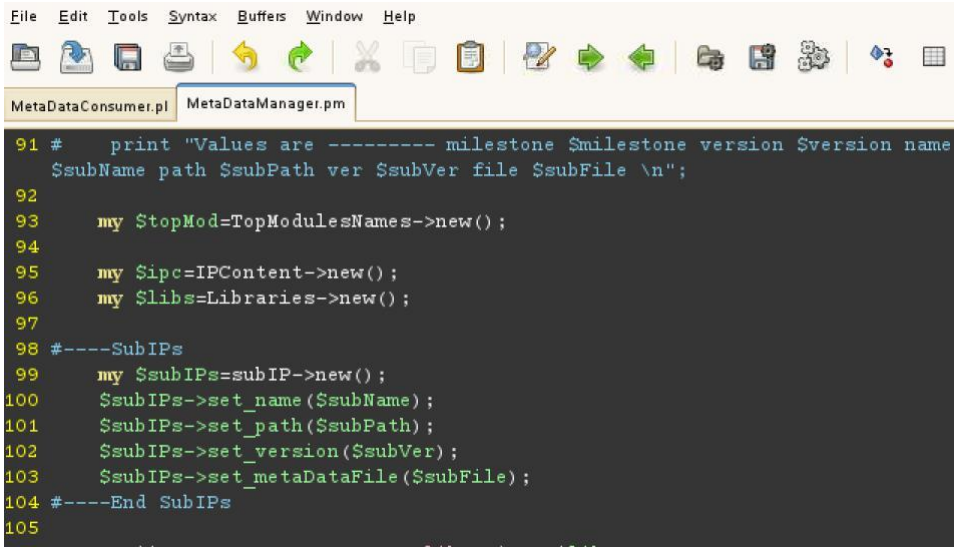


Figure 6.6: Metadata Manager Module Example

(g) The consumer will run this file to test the functionality of each data structure.

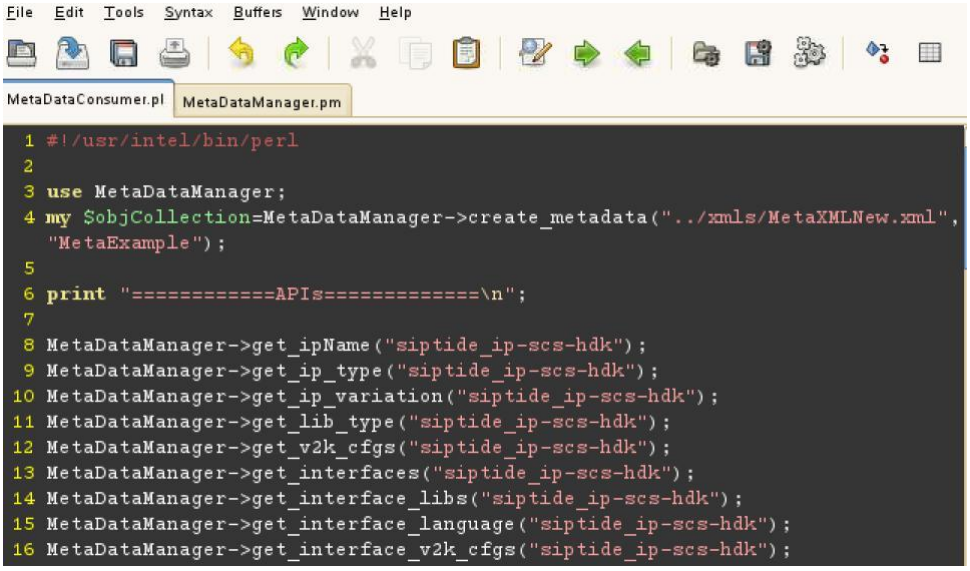


Figure 6.7: Metadata Consumer Example

(h) A small snapshot of the result of overall flow of the tool.

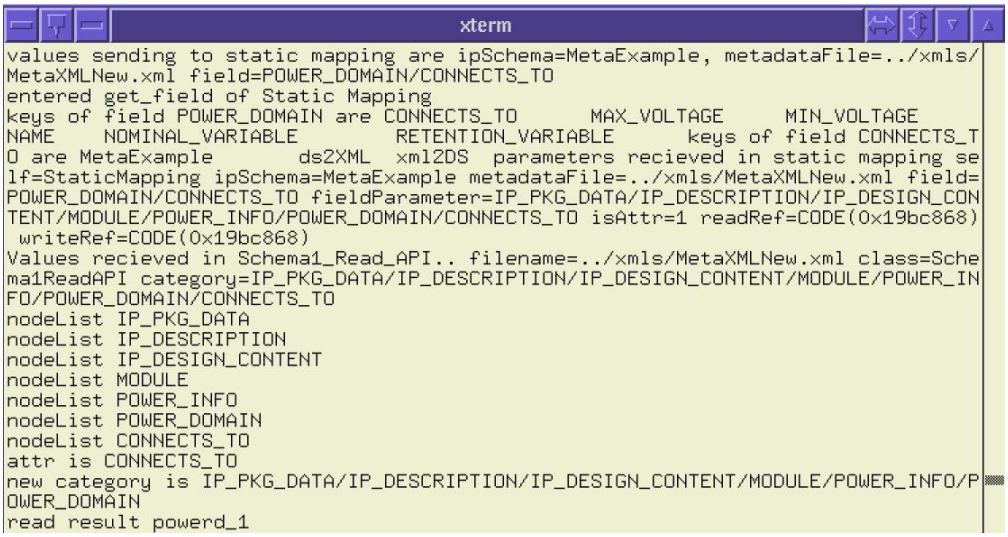


Figure 6.8: Example of overall flow result from reading to mapping

CHAPTER 7

CONCLUSION & FUTURE SCOPE

7.1 Work Conclusion

(a) In this paper, I have proposed another engineering and its outline strategy which empower fashioners to confirm, adjust, and troubleshoot their chip after creation utilizing this API approval device. In addition, design adaptability and debugga-bility will greatly increment if FPGAs are joined into the proposed engineering as extra useful units.

(b) The Objectives of the undertaking are effectively accomplished as appeared as results. Presently numerous IP groups have begun utilizing this tool as central area to arrange the information go as XML pattern and decide the IP quality and combination along with IP collateral management.

(c) It follows TDD based usage for testing which tests every area independently and also supports dynamically changing schema, without requiring changes at the client of the Metadata API.

7.2 Future Scope of Work

(a) In IP Tool, IP teams suggested some enhancements to make auditing system more flexible to the users. So many checks can automate by analysing the results of tool run which is done by TDD methodology.

(b) This will improve accuracy and reduce the effort of IP teams. Our system should be intelligent to parse the tool run logs for each checks and need to pre-fill the item status and comments.

(c) Several similar IPs can be grouped together, thereby no need to fill the checks for each IP. IP teams need an option to import the result data from previous milestone to current milestone.

(d) So tool can retain the audited checks to the current milestone. IP teams need an option to add checks specific to each IP.

(e) It should be flexible to load from new metadata formats in future. One of the major challenges in central area is tool run time.

(f) If we reduce the run time, we can save man power and improve quality. Main agenda is to design a framework to improve tool run time and efficiency.

(g) We are likewise building up a direction age apparatus which can treat FPGAs. What's more we are likewise building up a C-based identicalness checking ap-paratus and a format driven abnormal state amalgamation instrument in view of O-tree, which are utilised as a part of our outline flow.

REFERENCES

[1] Bricaud, P. . ., Reuse methodology manual: for system-on-a-chip designs. Springer Science & Business Media, 2012.

[2] Fujita, M., S. Komatsu, S. Saito, K. Seto, T. Sakunkonchak, and Y. Kojima, Field modifiable architecture with FPGAs and its design/verification/debugging methodologies. IEEE, 2003.

[3] Menhorn, B. and F. Slomka, Confirming the design gap. In D. Nagamalai, A. Ku-mar, and A. Annamalai (eds.), Advances in Computational Science, Engineering and Information Technology. Springer International Publishing, Heidelberg, 2013. ISBN 978-3-319-00951-3.

[4] Rajsuman, R. and H. Yamoto (2004). Method and apparatus for soc design vali-dation. Google Patents.

[5] Rashinkar, P., P. Paterson, and L. . . Singh, System-on-a-chip verification: methodology and techniques. Springer Science & Business Media, 2007.

[6] Saha, D. and S. Sur-Kolay (2011). Soc: A real platform for ip reuse, ip in-fringement, and ip protection. VLSI Design, 2011, 1–10. ISSN 1563-5171. URL http://dx.doi.org/10.1155/2011/731957.