Development of Plugins for Library Validation

Submitted By

Noopur Shirahatti

14MCEI10

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2016

Development of Plugins for Library Validation

Major Project

Submitted in fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

Submitted By

Noopur Shirahatti

(14MCEI10)

Guided By

Prof. Jigna Patel

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2016

Certificate

This is to certify that the major project entitled **"Development of Plugins for Library Validation"** submitted by **Noopur Shirahatti (Roll No: 14MCEI10)**, towards the fulfillment of the requirements for the award of degree of Master of Technology in Computer Science & Engineering (CSE) of 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. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Jigna Patel
Guide & Assistant Professor,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Sharada Valiveti
Associate Professor,
Coordinator M.Tech - INS
Institute of Technology,
Nirma University, Ahmedabad

Dr. Sanjay Garg
Professor and Head,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad

Dr. P. N. Tekwani
Director,
Institute of Technology,
Nirma University, Ahmedabad

Certificate

This is to certify that the major project entitled **"Development of Plugins for Library Validation"** submitted by **Noopur Shirahatti (Roll No: 14MCEI10)**, towards the fulfillment of the requirements for the award of degree of Master of Technology in Computer Science & Engineering (CSE) of 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.

Date:

Project Manager

Mrs. Jyoti Kumar

TRnD Department,

STMicroelectronics, India

Statement of Originality

I, **Noopur Shirahatti**, Roll. No. **14MCEI10**, give undertaking that the Major Project entitled "**Development of Plugins for Library Validation**" submitted by me, towards the fulfillment of the requirements for the degree of Master of Technology in **Computer Science & Engineering** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made.It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

Signature of Student
Date:16 May 2016
Place:Ahmedabad

Endorsed by
Prof. Jigna Patel

Acknowledgements

First and foremost, sincere thanks to my mentor **Mr. Rishabh Bansal**, STMicroelectronics, Noida. I enjoyed his vast knowledge and owe him lots of gratitude for having a profound impact on this report.

I would like to thank , **Mrs. Jyoti Kumar**, Manager, STMicroelectronics, Noida for her valuable guidance. Throughout the training, she has given me much valuable advice on project work. Without her, this project work would never have been completed.

My deepest thanks and profound gratitude to **Prof. Jigna Patel**, Assistant Professor, Computer Science Department, Institute of Technology, Nirma University, Ahmedabad for her valuable guidance and continual encouragement throughout this work. The appreciation and continual support she has imparted has been a great motivation to me in reaching a higher goal. Her guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

I am highly grateful to **Dr. Sharda Valiveti**, PG Coordinator ,Information and Network Security, Nirma Institute of Technology for her kind support and permission to use facilities available in the institute.

It gives me an immense pleasure to thank **Dr. Sanjay Garg**, Hon'ble Head of Computer Science and Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr. P. N. Tekwani**, Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation he has extended throughout course of this work.

- Noopur Shirahatti

14MCEI10

Abstract

Design for ICs are created by STMicroelectronics. These packages are then sold to customers and they fabricate it for further use. These packages are validated before giving it to customers. This validation when done manually takes time and efforts. The work is done to develop the scripts for validating the library, so as to reduce the time for validation.

A library for an IC chip is a collection of cells and has various layers. A cell is the basic deisgn unit. Different views are defined that tells about the physical, logical and timing information of the cell. LEF, lib, cdl, verilog etc. contains the information related to cell. The views are also customized according to the needs of different companies.

For validation different plugins are created which checks certain aspects of a library. The work was done on Modelization and Crosscheck plugin. For DRC check and LVS check the specs were changed, so the new code was developed. For some checks bugs were fixed. The customer gives the specification for a check. A spec file contains the basic flow, view on which check is to be performed, the options that user can provide for running that check, etc. Then we develop the scripts for that check. Then unit testing is done by us and we give it for Regression testing. And if there is no problem in package,then it is provided to customer.

The library can be tested for validating different views, checking the syntax of the views or checking the consistency between different views or the tags are crosschecked against certain specifications, the routing or obstruction information is also validated etc.

The scripts that are developed are integrated in a module and tested against various libraries using different tools and versions. The reports and logs are generated, so that the user can know where the error is present in library.

Abbreviations

LEF	Library Exchange Format
CDL	Circuit Description Language
SIP	System in Package
ALU	Arithmatic & Logic Unit
VLSI	Very Large Scale Integration
BDU	Basic Design Unit
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
LVS	Layout versus Schematic
GDS	Graphical Design System
HCELL	Hierarchically Corresponding Cell
CAD	Computer-aided design
V	Verilog
DEF	Design Exchange Format

Contents

List of Figures

Chapter 1

Introduction

1.1 Terminology

Library contains the design data related to an IP. It could be timing information, functionality, layout information etc. that will be fabricated on chip. Library is collection of cells. The library comprises of various views which are useful in designing a chip[1].
Cell is component performing a basic function. More a digital design concept: Boolean and basic function (AND, OR).
View is a particular representation of a cell. A view may contain information of multiple cells. Each cell may have its information in layout view, schematic view, a symbolic view, a timing view etc. Each view is used by a different tool in a given design flow.[1]

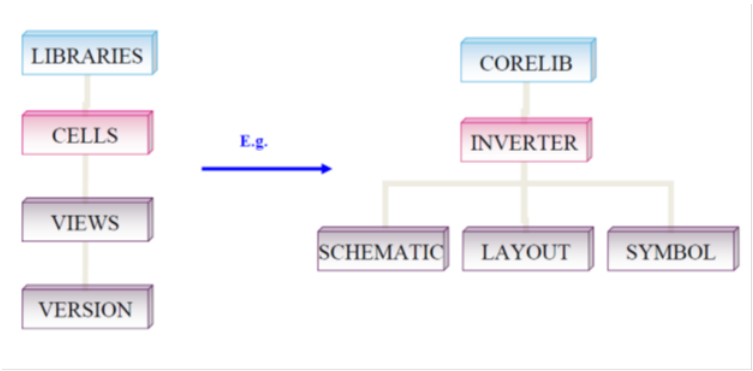

Figure 1.1: Library,Cells and views

1.2 Background

The design team generates a package that contains various views. The package (library) needs to be validated before fabricating it on chips. If validation is done manually then it takes much time and still leaves scope for error. So, the work is to develop a solution (plugins) that will validate the library in less time and in different aspects. Different views can be validated separately like LEF (Library Exchange Format), FRAM, cdl, lib etc. The syntax can be checked for various views.

Library structure

This is the structure expected for each library of IO libraries: Physical informations are in the directory <library>/SIP/PHYSICAL. Two files are used to physically describe IOs. There are a couple of files for each library :

- lef file : <library>/SIP/PHYSICAL/<library>.lef
- cml file : <library>/SIP/PHYSICAL/<library>.cml

The LEF (SIP LEF) is loaded by SoC Encounter tool and CML is loaded by CDNSIP. This translation from lef path to cml path happens automatically in the CDNSIP tool and hence its required to have lef and cml files in same directory with same names like <name >.lef and <name >.cml.

Technology LEF file is present in <library>/LEF which contains the name of the MACROs(cell) present in library and its attributes like cell area, size, coordinates of each cell, pin information present for each cell.

The various types of libraries are:

- Standard Cell Libraries[2]
- IO libraries
- Memories
- Macros

1.3 Objective

The work is to create a solution so as to validate the library and report any inconsistency or errors; so that the package can be fabricated on chip.

1.4 Scope

The scope of the project was to enhance the scripts of different plugins for ipscreen and develop scripts for new architecture.

This chapter gives a brief overview of the project and explains the terminology that will be used throughout the report the project and objective is also specified. In second chapter,the focus is on the literature survey. Tools and technology used is given in third chapter. Fourth chapter contains the implementation. Future work is specified in fifth chapter.

Chapter 2

Literature Survey

2.1 Analysis of Libraries and its views

Library

We can realize different equations through Boolean functions. Basically it is the process of breaking a complex function into smaller and understandable (basic) functions. Some of these basic functions are AND, OR, NOR, XOR. Assuming a designer is going to make a big design, say ALU, he will require the basic gates or the basic functions to constitute a big design. This is something similar to the human body which consists of basic units called cells.

Now, with this basic idea of how big VLSI designs are formed, we will be able to understand the definition of a library more clearly, as given below.

Taken in its simplest form, a library is collection of basic design gates such as AND, OR, XOR etc. or in other words Library Developers provide the basic design units to constitute bigger designs.

Types of Library

- CORE library

  It consists of Standard cells that implement a basic function like inverter, latch etc. Core Library is collection of components like gate, registers, counters, adders etc.

- Input/Output Library

  It consists of cells called IO buffers that are used to interface chip signals to chip

environment. The external voltage coming to the chip must be checked and modified accordingly, so as to ensure proper functioning of the chip.

- Memory Library

  It contains memories of different architectures like SRAM, DRAM, ROM etc. As the size of a memory can differ, a basic building block is implemented and use it to configure the generation of different memory sizes.

- Analog and Mixed cell Library

  These are implemented using CORE library. Example of this is Digital to Analog converter.

Further we will see how a library is divided into different views in order to provide different pieces of information related to the library.

Views

```
dlhl2086{EXPORT}>> ls -l
total 60
drwxr-xr-x 4 shirahan nvmecl  4096 Mar 21 14:08 behaviour
drwxr-xr-x 6 shirahan nvmecl  4096 Mar 21 14:08 CADENCE
drwxr-xr-x 3 shirahan nvmecl  4096 Mar 21 14:08 doc
drwxr-xr-x 2 shirahan nvmecl  4096 May  3 10:31 INTERFACE
drwxr-xr-x 2 shirahan nvmecl 12288 Mar 21 14:08 libs
drwxr-xr-x 3 shirahan nvmecl  4096 Mar 21 14:08 MENTOR
drwxr-xr-x 3 shirahan nvmecl  4096 Mar 21 14:08 NOVAS
drwxr-xr-x 2 shirahan nvmecl  4096 Mar 21 14:08 packaging
drwxr-xr-x 2 shirahan nvmecl  4096 Mar 31 10:28 physical
drwxr-xr-x 3 shirahan nvmecl  4096 Mar 21 14:08 SIP
-rw-r--r-- 1 shirahan nvmecl  3921 Mar 21 14:09 ST_DISCLAIMER.txt
drwxr-xr-x 4 shirahan nvmecl  4096 Mar 21 14:08 STM
drwxr-xr-x 4 shirahan nvmecl  4096 Mar 21 14:08 SYNOPSYS
dlhl2086{EXPORT}>>
```

Figure 2.1: Library's folder

In literal terms view means a kind of media to provide some specific information. A library takes help of several views to provide different pieces of information related to the libr

ary. All these views explains the different aspects, coverage and functionalists of BDU which a library contains.[1]

There are broadly two different fields of work in library development:

- Back-End

  In this, activities are concerned with:

  - The designing of BDU according to the customer specification.
  - Generation of different Back-End views and their verification.

- Front-End

  This activities encompass:

  - Calculation of different parameters for gates such timing, power etc. and providing them through Front-End views.
  - Modelling of gates in Hardware languages such as Verilog, VHDL.

Backend view is divided in 4 categories-

- **Symbol-**

  Symbol graphics define what we will see when this symbol is used in the schematic. It includes pins, symbol graphics, labels and a selection box. Pins are input and outputs of a symbol. The shape of the symbol can indicate the cells function. Labels in the symbol are used to add to the documentation of the design. Selection box in the symbol defines the area of the symbol that an instance will be selectable by. Symbol view is a pictorial representation of the cell. SLIB view is the text representation of the symbol view. SLIB is the derived view of symbol.[1]

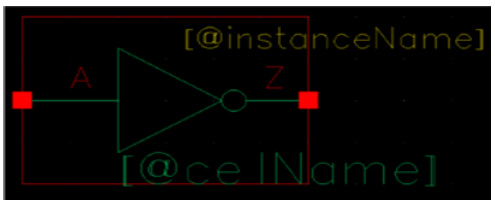

Figure 2.2: Symbol view

- **Schematic-**

  Schematic view is a simplified notation of a circuit. It shows the various parts of circuit as standard symbols that are in simplified form, and connections like the power and signal connections between the devices. It is representation of a cell at the transistor level. A schematic view includes component instances, wires and pins. Pins are the inputs and outputs of the schematic. Arrangement of the components and their interconnections on the diagram does not correspond to their physical locations in the finished device.[1]

  Circuit Description Language (CDL) is the text description of the schematic. CDL file tells about the :

```
Example:

symbol ("IVHD") {

set_minimum_boundary (0 * SCALE, -40 *
SCALE, 120 * SCALE, 40 * SCALE);

circle (88 * SCALE, 0 * SCALE, 5 *
SCALE);

line (83 * SCALE, 0 * SCALE, 40 * SCALE,
-25 * SCALE);

.....

pin ("A", 0 * SCALE, 0 * SCALE,
ANY_ROTATION);

pin ("Z", 120 * SCALE, 0 * SCALE,
ANY_ROTATION);
```

Figure 2.3: Slib example

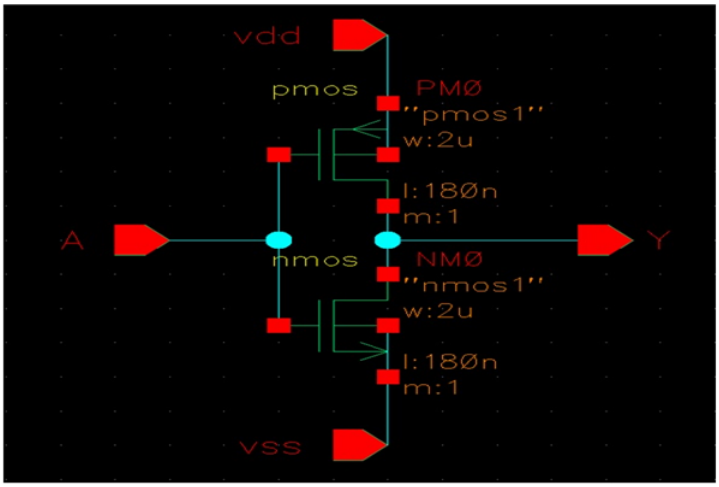

Figure 2.4: Schematic view

- Connectivity at the transistor level.
- Device parameters like device name/length/width/Area.
- The hierarchy traversal information of the pins.
- The related bias pin information of the pg pins.
- The related power pin and related ground pin information at the transistor level.

CDL is used during LVS (Layout vs. Schematic). LVS requires design cdl, which contains the top level information. The design cdl in turn contains the instances of the library cdl subckts which contains the internal connection information within the cell. Logic elements with the same name and the same number of inputs and

outputs, but with different implementations, must have different subcircuit definitions in CDL. Either a different name or the name extension should be used, else calibre gives duplicity error.

- **Layout-**

  Layout view is the actual physical representation of the electrical circuit of cell that goes on the silicon. Physical representation is made utilizing planar geometric shapes that are similar to the patterns of metal oxide or semiconductor layers that make up the parts of an IC. Different layers in a cad environment are used to draw this physical structure keeping in mind a set of rules that need to be followed.[1]

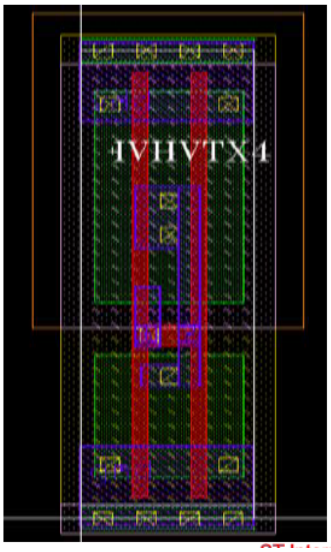

Figure 2.5: Layout view

Graphical Design System(GDS)

GDS contains the same data as the layout view and is in binary format that represents the planar geometric shapes, text labels and layers, such as text, path/wire, boundary/polygon, structure references, and planar geometric shapes in hierarchical form. GDS is not dependent on any platform because internally defined formats are used in this for its data types. Numeric attributes are assigned to the objects like layer number.[3]

Layer Map Files

A layer map file tells Cadence tool how to convert between layers in a Cadence layout and layers in a GDS file. It acts as a TRANSLATOR between the two Views. This results to the GDS format acting as list of records that are sequential.

The information in each record is contained in a header. According to GDS, the order of the record is kept. So, it is relatively easy to parse gds by the tool due to this architecture.

Hierarchically Corresponding Cell(HCELL)

HCELL contains list of cell name pairs, each pair consisting of a layout cell name and a corresponding source cell name in cdl. HCELLs are provided to improve LVS-H performance. These are the cells are placed numerous times in the hierarchy. Cells that appear in hcell lists are not flattened everytime, and therefore could impede the performance-improvement heuristics in LVS-H. In a pair of hcells, the layout cell name and corresponding source cell name may be the same, or may be different. One-to-Many relation can be specified by placing a layout cell name in several hcell pairs with different source cell names. Many-to-One relation can be specified by placing a source cell name in several hcell pairs with different layout cell names. Many-to-Many relations are not allowed.

- **Abstract-**

  Abstract view gives information about the signal and power pin layers running in the layout view along with the information of area where routing is not allowed. Abstract views have - pin information, black-box information, routing obstruction information.

  This view is useful while Placement and routing. Placement and Routing using Semi-Custom Tools does not require the full layout information but only the location of various pins to be connected and areas where routing is not allowed (obstructions). So this information is extracted from the layout to make an abstract view.[1]

  Library Exchange Format(LEF)

  The LEF file is a representation of the Abstract. Library information for a class of designs is stored in LEF. Data in library includes placement site type, layer, via, and macro cell definitions. It contains following information:

  - Cell Name
  - Cell Size
  - Definition of Pins

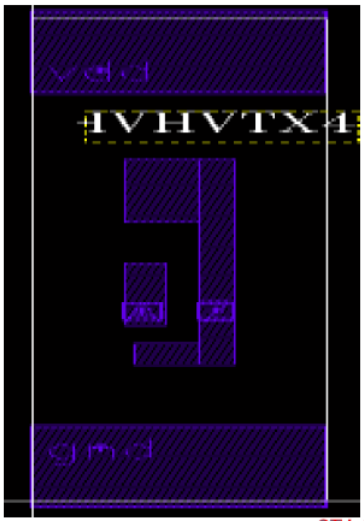

Figure 2.6: Abstract View

- Direction of Pins
- Location of Pins
- The Metal in which they are present
- Whether they are power or ground.
- OBS layer definition

A single LEF file can contain all of the library information. But this will create a large file that can be hard to manage and complex. Due to this,the information is placed separately into two files, a technology LEF file and a cell library LEF file. All of the LEF technology information for a design, such as placement and routing design rules, and process information for layers is stored in technology LEF file. A technology LEF file includes LEF TECHNO statements like: Manufacturing Grid, Layer Information (Routing/Non-routing), Via statement/Via Rules, SITE statements,etc.

The macro and standard cell information for a design is placed in cell library LEF file. A library LEF file may contain any of the following statements: MACRO statement, PIN statement, OBS statement, etc. The technology LEF file is read first at the time of reading of LEF files.[1]

Design Exchange Format (DEF)

The design-specific information of a circuit is contained in DEF file. During the layout process, it represents design at any instance. It is a representation that uses

the syntax conventions. DEF conveys Logical design data is converted to physical design data using DEF using place-and-route tools. Internal connectivity (i.e., netlist), grouping information, and physical constraints can be included in logical design data. Placement locations and orientations; routing geometry data; logical design changes for back annotation are to be included in Physical design data.[1]

Types of Information in a library

As we know, a libaray contains several views which provide different information. We can broadly categorize this information as:

1. Physical Information cell size, geometry of layer, area, direction, type (in lef file)
2. Logical Information functionality of the cell (in verilog file)
3. Timing Information rise time, fall time, delay time (in .lib file)

Library Structure The library is maintained by an index file. The structure of the index file includes various subsections that all together index the library files based on various conditions.

- Header: This section includes the library name, product name, process like 65nm, 45nm, 40nm etc, type of library (memory, standard cell, input-output etc).
- Cell: This section includes various cells.
- Conditions Section: This section includes conditions based on parameters like Process Variation(PV), Voltage(V), Temperature(T).
- Index Section: The index section includes paths to various cells based on different conditions. IPScreen parses this section to access the different views of the library.

2.2 Analysis of IPScreen

IPScreen is a framework built using TCl and Tk which is used for validation of a package(library). When it is launched, it asks for library to be loaded that is to be validated. Multiple libraries of different technology can be loaded at the same time. After that the plugin, using which validation is to be performed, is loaded. The reference libraries are

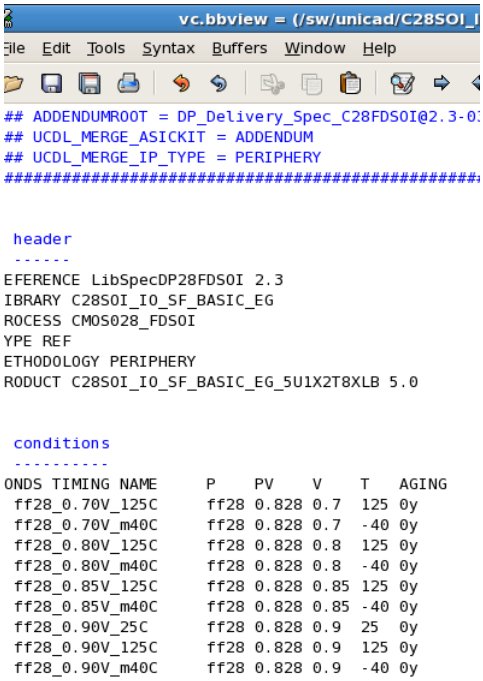

Figure 2.7: Library's vc.bbview file

loaded in next step.

The information of reference libraries is kept in a file in package. It may contain some basic cell's information that is to be used by the test library (library that is loaded).

Setup is the first task that is to be executed in this step. It creates a subdirectory for this plugin, in the directory where ipscreen is launched. In this area, a folder is created which contains information of path of various views and index file in that package and its reference libraries.

The user can give certain configuration settings for different views and it is also stored in a file, so that it can be referenced when needed. According to this configuration, the flow of check changes. For example, in DRC check if view selected is Cadence then flow will be different than that of when Physical view is selected.

When a check is executed, another folder gets created inside the plugin directory. This folder is the workarea and it contains the log file, report file and all other intermediate files that are created in that check.

Certain tools are also used by these checks for various purpose. These are present in a tool file. If some tool is missing in that file that will be required by any check in the selected plugin then that check is disabled in IPScreen. The execution time of check varies from library to library because the number of cells vary depending on the type of

library. For a Memory library the number of cells will be less than that of an IO library. The plugins and their paths are also stored in a plugin file. Different versions of plugins can also be added parallely.

When execution of check is completed, IPScreen shows whether that check ran successfully or falied.If it failed then we can see the report file that gives errors present in the library. The report is generated in text format, html format, csv format.

The check can be run on local machine or LSF. IPScreen can also be executed on local machine or on LSF.

IPScreen contains a folder in the directory where it is launched, that contains all the information about the library. For example, the views present in that library, its path, ipstyle (that is the type of library like IO library, Memory library, MACRO library etc.), path of index file and many other things.

IPScreen can be run in GUI mode or in Batch mode.

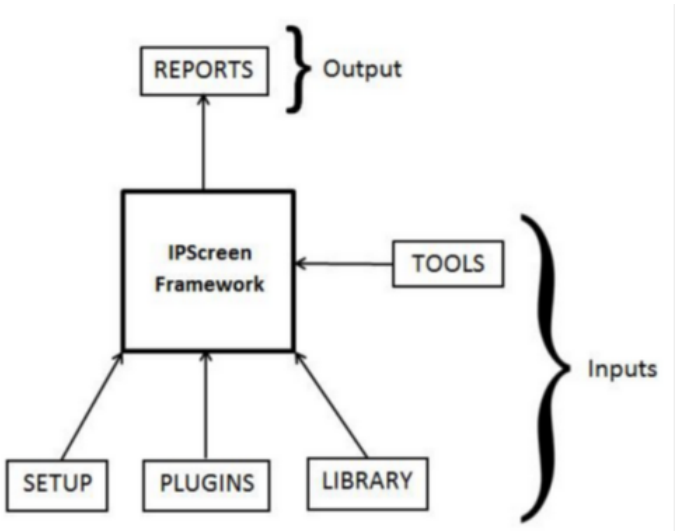

Figure 2.8: IPScreen Architecture

2.3 Validation Process Flow

The validation flow ensures the user that the IP under test is correctly validated. To make things simple to understand lets discuss on a single view.

- In the first step, presence of particular view in the package is checked. Another thing that is being checked is that indexation of the view is properly done in the file named vc.bbview or not. If this file is not populated correctly like the view is

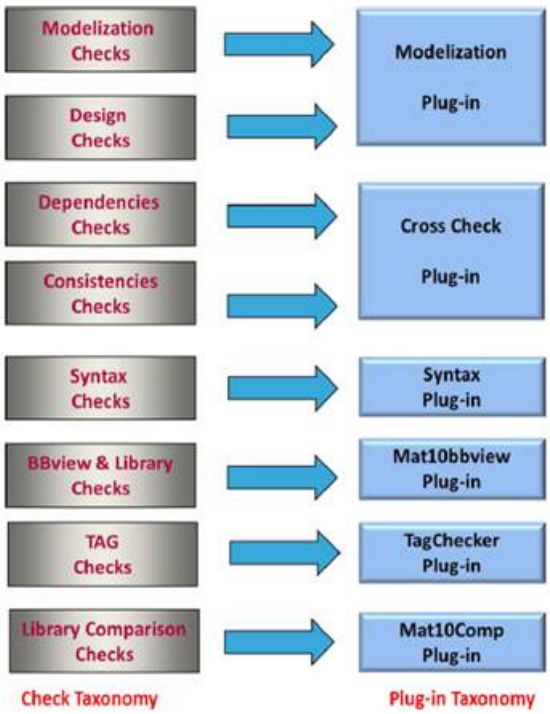

Figure 2.9: Mapping of checks and plugins

not indexed correctly then as a resultant that particular view cannot be taken by the tools. This is performed by Mat10bbview plugin.

- In the second step, syntax checking is done where the view is being read by the respective tools. In case of any incorrectness in the view the tool will not be able to process that particular view. This check is done by SyntaxCheck plugin.

- Once the view is checked syntactically, the next step is to check that the view is having the necessary attributes which are required by the tools. If they are found in the view then their values are being cross verified so that they are modelled according to the design rules mentioned in the Kits. These checks are being performed by Modelization plugin.

- In the fourth step, it is ensured that information is consistent between various views thats what the check is called as cross view consistency. Using tools of all the different vendors the view contents are verified in terms of data consistency. This is done by CrossCheck plugin.

- It is also being ensured that the tags mentioned in the layout are compliant with the convention. This check is done by a plugin named TagChecker.

- Mat10Comp plugin is responsible for checking out the difference between the previous versions and the existing version of the IPs.

2.4 Analysis of Plugins

Plugins are a group of scripts that are used to check the correctness of design and views. A certain plugin checks a library on a particular aspect. For example, CrossCheck plugin checks that all views are consistent with each other, TagChecker checks the tags for different views etc.

A file is present in plugin that contains the list of checks that can be executed, the tools on which a check is dependent on, the script that will be called first, etc.

Initially a script is called that sets all the environment variables that will be further used and it in turn calls the checks further. Such a script is called top script. Help folder is also present that a user can refer to. A subdirectory is present that contains the scripts of checks and another contains the additional scripts that are required by those scripts. The scripts are written in csh and tcl.

A file is present that stores the version of the plugin and the machines that it can work on. Then documentaion is also provided along with plugin. It contains the **user manual** that gives basic information about that plugin and its checks, **release history** that contains information about all the versions and bugs that were encountered, etc.

The specifications are provided to create a check. Specification contains the flow of the check, the configuration that a user can provide, and other necessary information.

The plugins that are used are:

1. Modelization

   This is used to check the correctness of library views model statically. It ensures accurate library functionality in the flow. In this, views on which validation is performed are LEF views, FRAM view, cdl view, gds view, lib view etc. Some of the checks are:

   - Design Rule Check
   - Layout vs. Schematic check
   - Route Guide
   - General LEF checks

- Site check
- Nitride Layer checks.

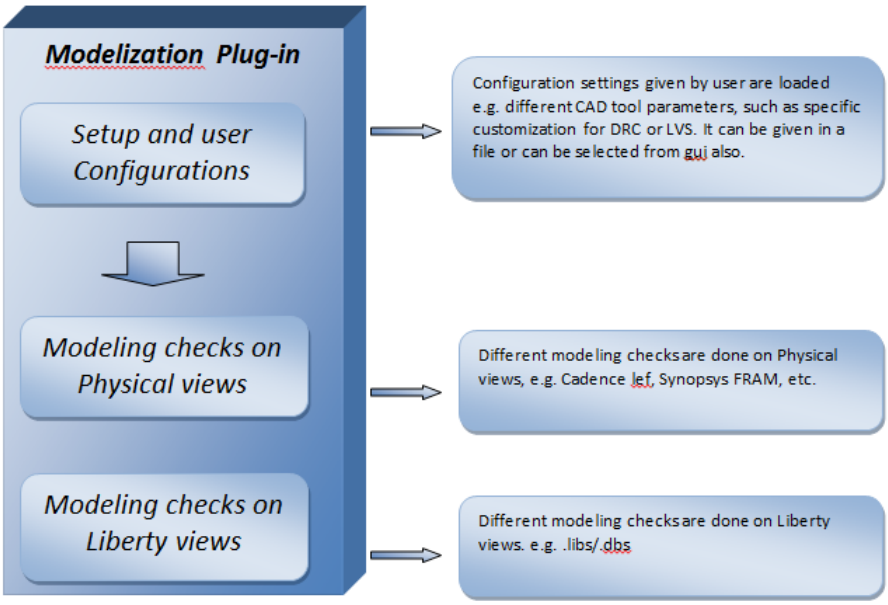

Figure 2.10: Modelization plugin

2. TagChecker

   It is dedicated to perform different checks on tags present in libraries. Two inputs must be given by the user- Spec, CAD tool required for plugin. Tag specification(Spec) can be of 4 types:

   - ADCSrevF
   - ADCSrevE
   - ADCSrevD
   - ADCSrevC

   Each contains certain set of tags. For example, Vendor, product, version, area, techno, etc.

   The views on which this is done are CDS5, CDSOA database, GDS view and AVANTI view.

3. SyntaxCheck

   This is designed to load each library view in its correspondent CAD tool to ensure syntax correctness.

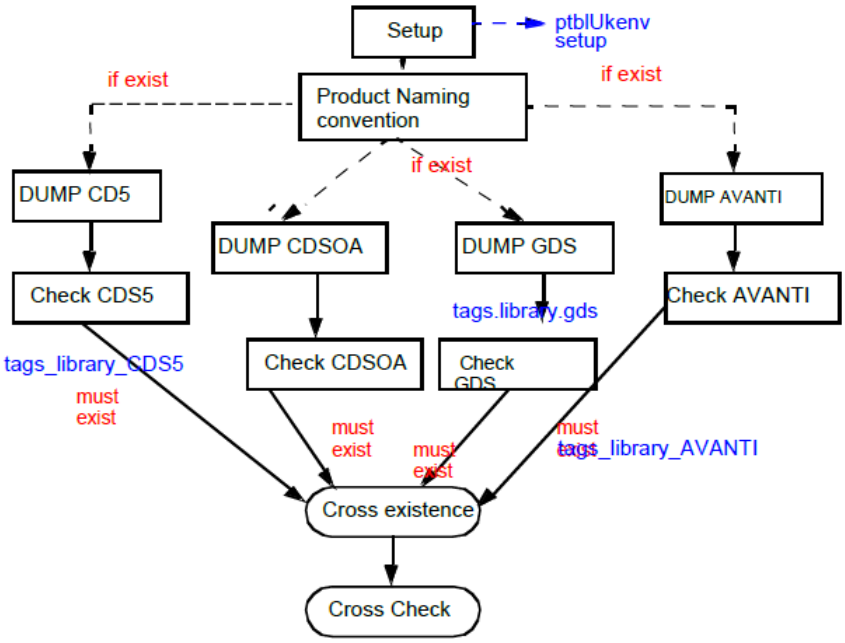

Figure 2.11: TagChecker plugin

4. CrossCheck

   This is designed to ensure views consistency. Similar view consistency - Ensure that same type of views are consistent among them (i.e. all liberty files, all lef files, all .v files etc). Cross view consistency - Ensure that all different views or Cross-Views are consistent with each other(i.e. .lef vs .lib, .lef vs .v)

   Some of the tasks are:

   - Abstract view extraction and consistency
   - Layout view extraction and consistency
   - Reference view extraction and consistency

5. Mat10Comp

   This compares two packages and lists whats different and common between them. General Comparison : It aims to compare the bbview file and libraries structure. Vendor Comparison : It contains tasks that compare two views of all the cad vendor. It also fills the plugin tables with comparison commands.

6. Mat10BBview

   This plugin checks the conformance of Mat10 libraries vs Mat10Library Specification 1.0 and 1.1 , in term of : mandatory views are present, and aligned with

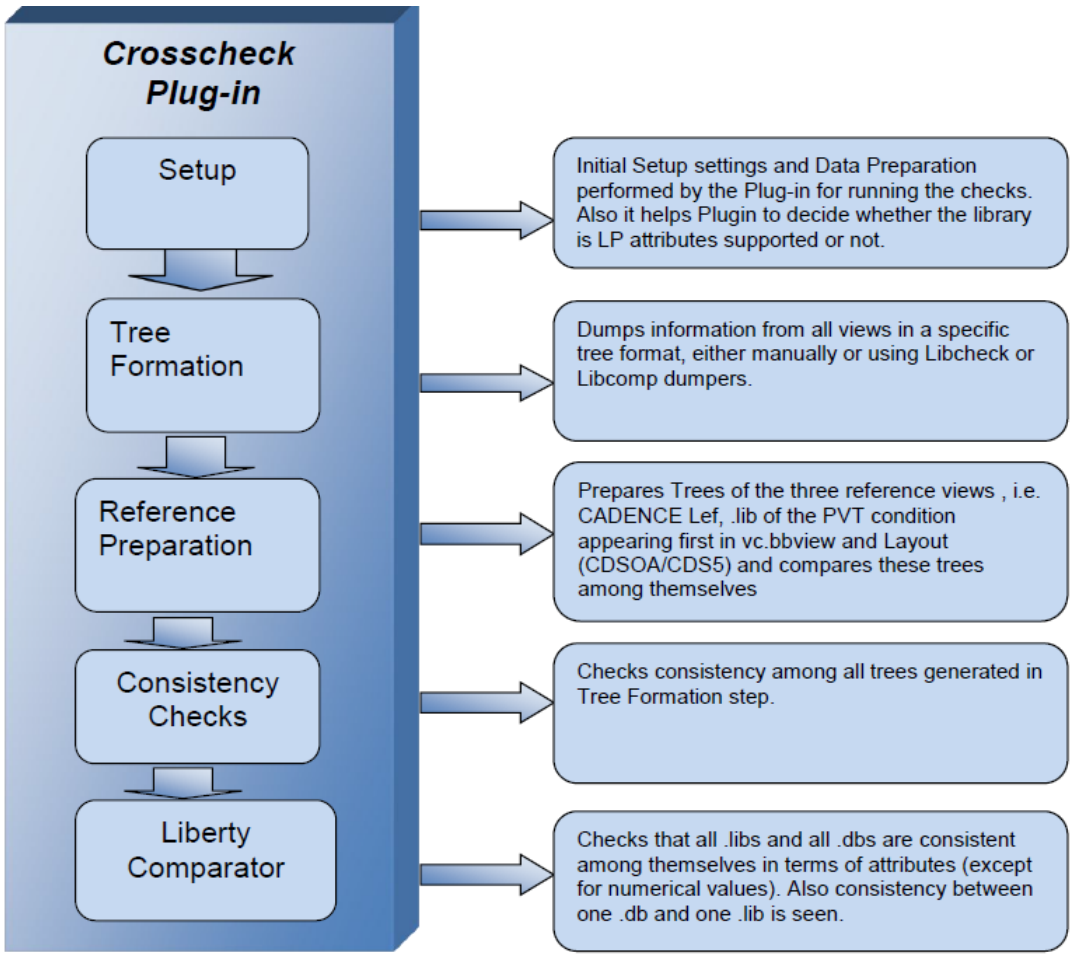

Figure 2.12: CrossCheck plugin

library structure. The checks consist of: bbview index keys, File convention name and library structure.

2.5 Analysis of Corekit and MifKit

MifKit is a kit that is used for error and warning reporting. Using this in the script, the **warnings**, **errors**, **info** etc. can be extracted from the log file. The work was to analyze this tool and see if this could be used for the new architecture.

For example, tuInfo, tuError etc. functions are predefined that are used to get the info and warning present. Then it also provides functionality for tracing and debugging, and facilitates to limit the number of messages that can be displayed. It also provides a report summary and report statistics that tell the count of the errors and warnings occured.

Corekit contains inbuilt functions for reading attributes of different views and extracting the values from that views, lef to stf converter etc. For each view, data structures are implemented using Object oriented tcl. Data structures contains classes in which

functions are implemented to get the values of the attributes present in that view.
This kit can be used in the scripts to get values required for further processing. We can instantiate an object of that views' database and then call the respective function. For example, if we want to get the names of cells present in a lef file and the pins and port information related to each cell then this can be used.

2.6 Brief about CadVal

CadVal is the new module. When the command is fired it starts executing the top script that creates the run area and inside it different directories. One such directory named CHECK is created that contains the further information related to that check. Top script calls another script that identifies the switches(options) that are provided by the user. The mandatory options that are to be provided are the library name, its bbview path and the script (csh script) that is to be called. Now, according to the user's option the script is executed.
Before this, a script that contains all the environment variables is executed. When the script is run, the file checks for the library that is provided by user and checks if the path given is correct. If it is correct the respective view is extracted from library. On this view the further operations are performed. Several intermediate files are also formed that are required by the tcl script following. For example, the list of all the layers present in the library or likewise. Then the tcl script gets executed that performs intended task. Then the logs gets stored in a separate directory and report in another one.

Advantage of CadVal over IPScreen

- It occupies less memory as compared to IPScreen because there is no need to load library and reference libraries.
- Less execution time.
- Setup file is created automatically.
- User can run specific checks without loading the whole plugin.
- Reporting is generic in new architecture.

Chapter 3

Tools and Technology

These are the technologies that are used in the project for development of plugins.

3.1 Tool Command Language (tcl)

TCL is a dynamic programming language and is the least prescriptive of the scripting languages. It contains no reserved words and there is no built-in syntax for control structures or conditionals.[4]

Everything in TCL script is implemented as a command, even control and branching operations are also implemented as commands. For example, if is a command that in its simplest form takes a conditional expression to test, and a block of code to be executed if the condition is true. A Tcl script could replace if with its own implementation. Each program written in Tcl essentially becomes a domain-specific language for whatever it is that program is designed to do.[4]

In addition to being a programming language, it is also a cross-platform C library. If you have to use glib etc. in your C program then you can do it using TCL for providing its features.[4]

For plugins, the main tasks that are performed by the checks are implemented in tcl. Different kits/tools that are used are also implemented in tcl.

3.2 csh

The C shell is a command processor typically run in a text window, allowing the user to type commands. The C shell can also read commands from a file, called a script. Like all Unix shells, it supports filename wildcarding, piping, here documents, command

substitution, variables and control structures for condition-testing and iteration. What differentiated the C shell from others, especially in the 1980s, were its interactive features and overall style. Its new features made it easier and faster to use. The overall style of the language looked more like C and was seen as more readable.[5]

For plugins, the wrapper classes that sets the path or enviromnent variables or get the information required by the tcl scripts are developed in csh.

Chapter 4

Implementation

4.1 Development of Parsers for different views using Corekit

The work was to develop parsers for traversing views like lef, def, cdl, lib and other view files and get the information required that can be used by checks.
Example, a LEF file contains the following information:

- Version of the library
- MACROs(cells) present in library
- Class of MACROs (example PAD)
- SIZE (length * breadth)
- PIN name
- PIN direction (input/output/inout)
- PIN use (power/ground/signal/clock)
- PORT present in PIN section and the LAYERs present in PORT section and layer geometry (coordinates)
- Obstruction information, etc.

```
##For multiple cdl files##

foreach file_name $argv {
    puts "\n# INFO : Reading cdl file $file_name #"
    set cdlName [lindex [split $file_name "/"] end]
    set cdlName [string range $cdlName 0 end-3]

    #Log file is created for storing information  (it's pointer is file_out)
    append cdlName "log"
    set file_out [open parsecdlOut_$cdlName w+]

    set cdldb [cdlDb new]
    $cdldb read $file_name

    set cell_obj [$cdldb getTopCells]

    set totalcells [$cdldb nbCells]
    puts $file_out "total number of cells are : $totalcells\n"
    puts $file_out "\nInformation from cdl file::\n"
    foreach cellObj $cell_obj {
          set cell_name [$cellObj getName]
          set diode_present [$cellObj hasDiodes]
          set cap_present [$cellObj hasCapacitors]
          set res_present [$cellObj hasResistors]
          set instance_present [$cellObj hasInstances]
          set bjt_present [$cellObj hasBjts]
          set port_present [$cellObj hasPorts]
          set mosfet_present [$cellObj hasMosfets]
          set jfets_present [$cellObj hasJfets]
          set module_present [$cellObj hasModules]

          #Name of all the cells present
          puts $file_out "\nCell information -> $cell_name"

          #Diode name
          if {$diode_present==true} {
                foreach diode_obj [$cellObj getDiodes] {
                   puts $file_out "\tdiode name->[$diode_obj getName], diode Mname-[$diode_obj getMname], area-[$diode_obj getArea]"
```

Figure 4.1: Part of code for cdl Parser

The work was to write scripts to get such information related to a view. Scripts were written in tcl. The parser reads each attribute of the file and checks for any problem present in the file of library.

4.2 Development of Scripts for Modelization plugin for IPScreen

Modelization contains different scripts for validating lef and fram [3] views in different aspects. Some of them are general lef check, metal obstruction, via obstruction, route guide etc. For example,the task of one such check(general lef check) is to check if obstruction in overlap layer is proper or not and port section for each pin is defined or not, or the type of layer is routing or not.

Then one check was developed for lef(site check) which was applicable for library supporting MSOT flow. In this, I had to write the script to check that the width of cells should be multiple of corresponding SITE width and height of cells should match with corresponding SITE height.

The checks for which the work was done were: Hierarchical Cell level DRC, Hierarchical Cell level LVS, General LEF checks, Site check, Nitride layer check, etc.

For DRC and LVS, specification was changed. So the work was to write code according

```
                        puts $file_out "\t\tproperty name-> [$property_obj getName]"
                    }
                }
            }
        }

        #Module name
        if {$module_present==true} {
            foreach module_obj [$cellObj getModules] {
                puts $file_out "\tmodule name-> [$jmodule_obj getName]"
            }
        }

    }

    set nets [$cellObj getNets]
    set total_ports [$cellObj nbPorts]
    puts $file_out "\n\nNumber of ports/nets -> $total_ports"
##Same information (port,bus information)can be acquired in other way..using net
  #  foreach cellObj $cell_obj {
        puts $file_out "\nNet names:"
        foreach net $nets {
            set name [$net getFullName]
            set name1 [$net getName]

            puts $file_out "\n\tname of the net : $name1"
            puts $file_out "\tnet fullname : $name, bus - [$net getIsBus]"

        }

  #  }
    close $file_out
    puts "##INFO : Output File : parsecdlOut_$cdlName##\n"
}
```

Figure 4.2: Part of code for cdl Parser

to new specs.

- **Design Rule Checks (DRC)**

  DRC checks determine if the layout satisfies a set of rules required for manufacturing. The most common of these are spacing rules between metals, minimum width rules, via rules etc. There will also be specific rules pertaining to your technology. An input to the design rule tool is a design rule file (called a run set by Synopsys hercules). The design rules ensure sufficient margins to correctly define the geometries without any connectivity issues due to proximity in the semiconductor manufacturing processes, so as to ensure that most of the parts work correctly. The minimum width rules exist for all mask layers, and spacing between the same layers are also specified. Spacing rules may change depending on the width of one or both of the layers as well. There can also be rules between two different layers, and specific via density rules etc. If the design rules are violated, the chip may not be functional.

  Check that hierarchical DRC is clean on all the top cells of library. The task uses CDS5 or CDSOA database whichever is present in library. In case both are present, CDSOA will be taken. In case of 65nm, CDS5 is taken, even if both are present.

DRC.csh

- Get path of bbview file
- If status_setup file exists
  - Get the view that user has selected
- Else
  - If CDL & GDS exists,
    - Set view=phyGDS
  - Else
    - If CDSOA or CDS5 exists,
      - Set view=cad_database
    - Else give error and exit
- Call GDSOut_new.csh with arguments lib_name, bbview_path, view
- If user has given drc_option
  - Set drc_option that user has given
- Else set drc_option = CELL_NOESD and store it in cust file
- If user has given userRunsetFile
  - Then copy that file to userRunsetConfigFile
- If usr has provided cell list and pad class then,
  - Append that file's path to cust file.
- Select between CDS5 and CDSOA
  - If both are present
    - If process is 65
      - Set flag_cds5 = 1
    - Else set flag_cdsoa = 1
  - Else select the one that is present.
- Set CDS = path of the database(CDS5/CDSOA selected)
- Get the list of cells from bbview and store it in List file.
- Set ref = path of reference libraries
- For each reference library, get path of its gds file.
- For each cell in cellList file (lib name, cell name, layout view),
  - Get values of lib name ,cell name and layout view.
  - Make a directory of name $lib_name and go to that directory.
  - If view is cad_database
    - Execute calibrerun with $cell_layout_view.gds file as input
  - Else
    - Execute calibrerun with physical GDS and gds of reference libraries
  - Go to parent directory
  - Store RULECHECK related information from drc_summary of each $cell_$layout_view and store it in a report file (of name $cell_$layout_view_rpt)
  - If error is present in report file, then give error "Cell level DRC for $cell is not clean"
  - Else Info "Cell level DRC is clean for $cell".

GDSOut_new.csh

- If TASK is LVS
  - If user has given path of file containing cell list
    - Open that file and copy cell names to List file.
  - Else copy cell names from vc.bbview to List file
- If TASK is DRC
  - If ipstyle is IO (library is of type IO)
    - If user has given path of file containing cell list
      - If pad class option is set to yes
        - Copy the names of cell to List file
        - Copy pad class provided in cellList file to padClass file
      - Else if pad class option is no
        - Copy cell names to List file from cell list file
    - Else if cell list is not provided
      - If pad class option is yes
        - Then give error
      - Else take cell names from vc.bbview
  - Else if ipstyle is not IO
    - If cell list is provided
      - Copy cell names from cellList file to List file
    - Else get cell names from bbview.
- If TASK is LVS
  - If ipstyle is IO
    - If view is phyGDS or phyGDS_phyCDL
      - Set path_database to path of cdl file of library
    - Else set path_database to path of CDSOA database
    - Call cellCompCdlCdsoa.tcl with arguments $view, $path_database
- Select between CDS5 or CDSOA database and set CDS as path of the selected database
- Evaluate uniopus tool depending on database that is selected
- Add reference libraries in cds.lib
- If ipstyle is IO and TASK is LVS
  - Copy contents of commonCellList file to finalCellList file

- Else copy contents of List file to finalCellList file
- If ipstyle is IO and view is cadLAYOUT_cadSCHEMATIC
  - For each cell in finalCellList file
  - If $CDS/$cell directory is present ans it contains "schematic" directory
    - Add that cell to a file for making its .cdl file using "si" tool
- If view is phyGDS or phyGDS_phyCDL
  - For each cell in finalCellList file
    - If $CDS/$cell directory is present
      - For each layout_view present in $CDS/$cell
        - Set layout_view_type from $layout_view
        - If $layout_view_type is not empty
          - Set cell_new to $cell_$layout_view_type
        - Add $lib $cell_new $layout_view to cellList file
- Else
  - For each cell in finalCellList file
    - Streamout each cell using strmout tool
    - Rename output file generated to $cell_$layout_view.gds

cellCompCdlCdsoa.tcl

- If view is phyGDS
  - Get the cells present in cdl file and store in cdlList file
  - Get cells common in List file and cdlList file and store it in finalCellList file
- If view is cad_database
  - Get cells in $CDS/$cell for which schematic view is present and store in cdsoaList file
  - Get cells common in List file and cdsoaList file and store it in finalCellList file.

Figure 4.3: Algo of DRC

- **Layout vs. Schematic**

  LVS is another major check in the physical verification stage. Here you are verifying that the layout you have created is functionally the same as the schematic/netlist of the design-that you have correctly transferred into geometries your intent while creating the design. So all the connections should be proper and there shouldnt any missing connections etc. The LVS tool creates a layout netlist, by extracting the geometries. This layout netlist is compared with the schematic netlist. The tool may require some steps to create either of these netlists (e.g. nettran run in synopsys). If the two netlists match, we get an LVS clean result. Else the tool reports the mismatch and the component and location of the mismatch. Along with formal verification, which verifies if your pre-layout netlist matches the post-layout netlist, LVS verifies the correctness of the layout w.r.t intended functionality.

  Some of the LVS errors are: Shorts Wires that should not be connected are overlapping, Opens Connections are not complete for certain nets, Unbound pins If the pins dont have a geometry, but all the connection to the net are made, and unbound pin is reported.

  Check that hierarchical LVS is clean on all the top cells of the library. Two options are provided - Cadence Layout vs Cadence Schematic. and Physical GDS vs Physical CDL. o Use default option is Physical GDS vs Physical CDL.

LVS.csh

- Get bbview path
- If user has provided view for LVS
  - Set view from it
  - If view is phyGDS_phyCDL
    - Get path of cdl file and gds file
  - Else get path of CDS5 and CDSOA database
- Else
  - Get path of cdl and gds file
  - If cdl and gds view exists,
    - Set view to phyGDS_phyCDL
  - Else get path of CDS5 and CDSOA database
  - If anyone of these is found
    - Set view to cadLAYOUT_cadSCHEMATIC
  - Else give error and exit
- Call GDSOut_new.csh with arguments library name, bbview path, view
- If cdl view exists
  - Create a $lib_name.cdl file that includes path of $cdl
- For each reference file
  - Get cdl and gds path and append it to respective $lib_name.cdl and $lib_name.gds files
- Get path of hcell
- Set hcell_option to -lvsUseHCells 1 -lvsHCellsFile $hcell
- For each cell in cellList
  - Get lib name, cell name and layout_view from cellList
  - Go to sub directory of lib name
  - If view is phyGDS_phyCDL
    - Execute calibrerun with source path as $lib_name.cdl and opther options
  - Else
  - If ipstyle is IO
    - Execute calibrerun with source path $cell.cdl
  - Else
    - Execute calibrerun with source path $lib_name.cdl
  - Got to parent directory
  - If report file generated by calibre contains "error"
    - Then give error : LVS is not clean for $cell

Figure 4.4: Algo of LVS

- **General Lef Checks**

  Some General checks are performed on Cadence lef : There are no 45 degree lines on overlap layer, Ports to be defined on routing layer, etc.

Lef_check.csh

- Get path of cadence lef file present in library and store its path in path_lef file
- Get path of bbview of library
- Get path of lef file present in Technology Kit (cadence techno kit or encounter techno kit) and store it in technolef
- Get list of LAYER present in technolef
- For each layer in technolef, copy the information related to it from technolef to a new file layer_$layer
- Call lef_checks.tcl

Lef_checks.tcl

- Read lef file of library line by line
- If MACRO is encountered in line
  - Get cell name from it
  - If LAYER overlap is encountered
    - Get x and y coordinates of POLYGON of that overlap layer
    - Store x coordinates of all overlap layers in one list and y coordinates in another
    - Get difference between the x coordinates
    - Get difference between y coordinates
    - If difference_x is not 0 and difference_y is not 0
      - Give error : Overlap layer of cell has 45 degree lines in it
  - If PIN is encountered
    - Get pin name
    - If PORT is encountered
      - If LAYER is encountered
        - Get layer name
        - Append layer name to a list
      - For ach layer in that list
        - Open layer_$layer file
          - Get layer type
          - If layer type is not ROUTING
            - Give error
    - Else if PORT section is not present
      - Give error : PORT section missing for pin

Figure 4.5: Algo of General Lef Check

- **Site Check**

  Checks on Sites in Cadence CDSOA database. The check is applicable only in std cells & IOs. This check is relevant only for libraries supporting MSOT flow. It checks following: The SITE of all library cells should have the definition present in SiteDefKit, Width of all the cells should be multiple of corresponding SITE width, Height of all the cells should match with the corresponding SITE height.

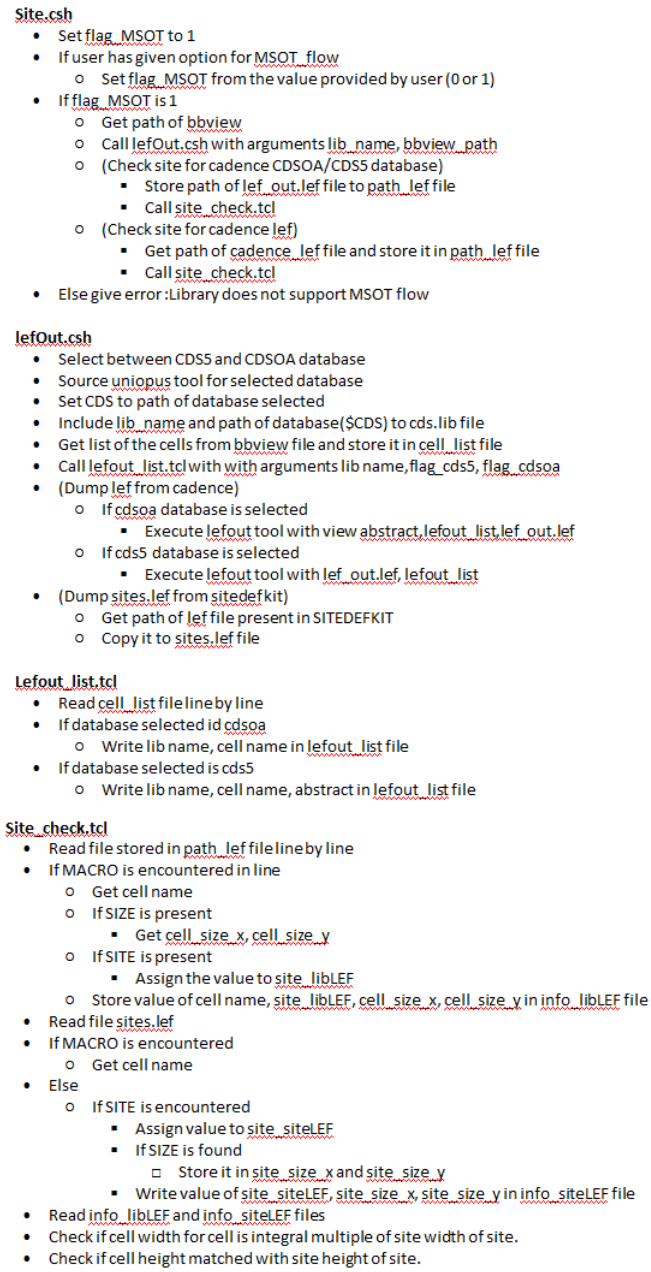

Figure 4.6: Algo of Site Check

- **Nitride layer Check**

  Checks that Nitride layer is present on wire-bond cells having class PAD in SIP LEF

and SIP CML. Option is provided in configuration file : Select WB (i.e. Wire Bond) to enable the check to run. By default it assumes Flip-Chip, so check does not run.

Nitride_layer.csh

- Set flag_wb_fc to FC
- If user has provided option wb_fc
  - Set flag_wb_fc from it
- If flag_wb_fc is WB
  - Get path of sip_lef and sip_cml and store it in path_sip_lef and path_sip_cml respectively.
  - Call nitride_layer.tcl
- Else give info : Library is Flipchip. So check is not performed.

nitride_layer.tcl

- Open file present in path_sip_lef
- If MACRO is encountered
  - Get cell name
  - If CLASS PAD is encountered
    - If PIN is found
      - Get pin name
      - If LAYER Nitride is encountered
        - Set flag_layer_nitride_lef to 1
    - If flag_layer_nitride_lef is 0
      - Give error : Layer nitride is missing for pin of cell for sip_lef
      - Set flag_layer_nitride_lef to 1
- Open file present in path_sip_cml
- If MACRO is encountered
  - Get cell name
  - If MACROCLASSNAME PAD is found
    - If PIN is found
      - Get pin name
      - If Nitride is encountered
        - Set flag_layer_nitride_cml to 1
    - If flag_layer_nitride_cml is 0
      - Give error : Layer nitride is missing for pin of cell for sip_cml

Figure 4.7: Algo of Nitride layer check

4.3 Development of Scripts for CrossCheck plugin for IPScreen

Scripts for Abstract and layout view were developed.

- **Abstract view extraction**

  Attributes of different lef views are dumped in .tree files.

Abstract.csh

- Call Abstract.tcl
- If snps_icc_IPS_pr_cat.lef file is present
  - Call snps_icc.tcl
- Extract errors and warnings present in tool log and check's log
- If errors are present
  - Write fail in check rpt fie
- Else If warnings are present
  - Write warn in check rpt file
- Else write done in it.

Abstract.tcl

- Get tools necessary for this task from tasks file
- Get views from tasks file that are to be crosschecked
- Get path of views and append it to list inputfiles
- Make a list outputfiles that contains list of view.tree files
- (Before tools were used to dump files. But now manual dumping is done to create .tree file)
- For each input in $inputfiles, out in $outputfiles, $view
  - Open $input file in read mode and $out file in write mode
  - If MACRO is encountered in line
    - Get cell name and write it in $out file
    - If CLASS is present
      - Write CLASS =$class in $out file
    - If SIZE is present
      - Get cell length and cell breadth
      - Calculate area= length *breadth
      - Write AREA=$area in $out file
    - If PIN is encountered in line
      - Get pin name and write it in $out
    - If ANTENNAGATEAREA id present
      - Write it in $out
    - If ANTENADIFFAREA is present
      - Write it in $out
    - If DIRECTION is present
      - Write it in $out
    - If TYPE is present
      - Write it in $out
- Call Antenna_check.csh

Antenna_check.csh

- Get path of avanti_pr view
- If astro tool is present
  - Execute astro tool with astro_command and avanti_pr view as input
  - Output is a lef file(snps_icc_IPS_pr_cat.lef)
- Else if milkyway tool is present
  - Execute milkyway tool with milkyway_command and avanti_pr view as input
  - Output is a lef file(snps_icc_IPS_pr_cat.lef)

snps_icc.tcl

- Convert the lef generated above to standard lef
  - Remove certain attributes like VERSION, CASESENSITIVE, etc. from it
- Open new lef(snps_icc_IPS_pr.lef) in read mode
- Open file snps_icc_IPSpr.tree ($out) in write mode
- If MACRO is encountered in line
  - Get cell name and write it in $out file
  - If CLASS is present
    - Write CLASS =$class in $out file
  - If SIZE is present
    - Get cell length and cell breadth
    - Calculate area= length *breadth
    - Write AREA=$area in $out file
  - If PIN is encountered in line
    - Get pin name and write it in $out
  - If ANTENNAGATEAREA id present
    - Write it in $out
  - If ANTENADIFFAREA is present
    - Write it in $out
  - If DIRECTION is present
    - Write it in $out
  - If TYPE is present
    - Write it in $out

Figure 4.8: Algo of Abstract view extraction

- **Abstract view consistency**

  Tree files dumped in extraction phase are compared with each other.

Crossabstract.csh

- Get test library and reference libraries names
- Get path of bbview, CDS5 database and CDSOA database
- Call siplefcadleflib.tcl
- Call lefcadlef.tcl
- If snps_icc_IPS_pr_cat.lef is formed in abstract view extraction check
  - Call prcadlef.tcl
- Call cadlefdetailedlef.tcl

siplefcadlef.tcl

- If sip_lef.tree is present
  - If RefAbstract.tree is present
    - Call procedure with arguments RefAbstract.tree, sip_lef.tree, RefLib.tree
  - Else if DetailedLef.tree is present
    - Call procedure with arguments DetailedLef.tree, sip_lef.tree, RefLib.tree
- Procedure-
  - (This procedure compares sip_lef.tree with RefLib.tree and sip_lef.tree with RefAbstract.tree/DetailedLef.tree)
  - (sip_lef.tree is formed in abstract view extraction check)
  - (RefLib.tree, RefAbstract.tree/DetailedLef.tree are generated in reference view extraction check)
  - Read RefAbstract.tree
    - Get these attributes : cell name, class, area, pin name, direction, type
    - Store them in data structures (example, cellevel($n,$m,0) with n=0, m=0 will represent cellname; n= 0,m=1,0 represents pin name etc. depending on the view structure)
  - Read sip_lef.tree
    - Get these attributes : cell name, class, area, pin name, direction, type
    - Store them in data structures
  - Read RefLib.tree
    - Get cell name, group, pin name, direction, is_pad information
    - Store them in data sturcture
  - Compare attributes of sip_lef.tree and RefLib.tree
  - Compare attributes of RefAbstract.tree and RefLib.tree
  - Give error if some attribute is missing/different
  - (Earlier dumping and comparision was done using tool libcomplef)

lefcadlef.tcl

- Comparision between attributes of RefAbstract.tree (cadence lef) and techno lef file
- Same as above

prcadlef.tcl

- Comparision between attributes of RefAbstract.tree (cadence lef) and FRAM (snps_icc_IPS_pr) file

cadlefdetailedlef.tcl

- Compariosion between attributes of RefAbstract.tree (cadence lef) and DetailedLef.tree(snps_icc2_IPS_lef)

Figure 4.9: Algo of Abstract view consistency

- **Reference view extraction**

  Reference lib files and lef files are dumped.

Refpreptree.csh

- Get names of test library and reference libraries
- Get path of bbview, CDS5 database, CDSOA database of test library
- Select between CDS5 and CDSOA database and source the tool uniopus for selected database
- If CDSOA is selected
  - Call Layoutrefoa.tcl
- Else if CDS5 is selected
  - Call Layoutrefcds5.tcl
- Call Abstractref.tcl
- Call Signoffref.tcl
- Get path of timing.lib
- Remove comments from timing.lib and copy it to nldm.lib file
- Call Libertyref.tcl

Layoutrefoa.tcl

- Get views from tasks file that are to be crosschecked
- Get path of views and append it to list inputfiles
- Make a list outputfiles that contains list of view.tree files
- Use libcomp tool to convert layout view(cds5/cdsoa database) from list $inputfiles to lef file
- Rename the lef file generated above to RefLayout.lef
- Read RefLayout.lef
- Write in $out file (RefLayout.tree file)
- If MACRO is encountered in line
  - Get cell name and write it in $out file
  - If CLASS is present
    - Write CLASS =$class in $out file
  - If SIZE is present
    - Get cell length and cell breadth
    - Calculate area= length *breadth
    - Write AREA=$area in $out file
  - If PIN is encountered in line
    - Get pin name and write it in $out
  - If ANTENNAGATEAREA id present
    - Write it in $out
  - If ANTENADIFFAREA is present
    - Write it in $out
  - If DIRECTION is present
    - Write it in $out
  - If TYPE is present
    - Write it in $out

Layoutrefcds5.tcl

- Similar to Layoutrefos.tcl just view is different.

Abstractref.tcl

- Get path of cds_soc_IPS_abstract.lef file
- Read cds_soc_IPS_abstract.lef file
- Write in $out file (RefAbstract.tree)
- If MACRO is encountered in line
  - Get cell name and write it in $out file
  - If CLASS is present
    - Write CLASS =$class in $out file
  - If SIZE is present
    - Get cell length and cell breadth
    - Calculate area= length *breadth
    - Write AREA=$area in $out file
  - If PIN is encountered in line
    - Get pin name and write it in $out
  - If ANTENNAGATEAREA id present
    - Write it in $out
  - If ANTENADIFFAREA is present
    - Write it in $out
  - If DIRECTION is present
    - Write it in $out
  - If TYPE is present
    - Write it in $out

Signoffref.tcl

- Get path of snps_rcxt_IPS_lef file
- Read snps_rcxt_IPS_lef file
- Write in $out file (snps_rcxt_IPS_lef.tree)
- If MACRO is encountered in line
  - Get cell name and write it in $out file
  - If CLASS is present
    - Write CLASS =$class in $out file

- If SIZE is present
  - Get cell length and cell breadth
  - Calculate area= length *breadth
  - Write AREA=$area in $out file
- If PIN is encountered in line
  - Get pin name and write it in $out
- If ANTENNAGATEAREA id present
  - Write it in $out
- If ANTENADIFFAREA is present
  - Write it in $out
- If DIRECTION is present
  - Write it in $out
- If TYPE is present
  - Write it in $out

Libertyref.tcl

- Read nldm.lib
- Write in RefLib.tree file
- Get value of normal process, voltage and temperature and write in RefLib.tree
- If cell is encountered
  - Get cell name and write in output file
  - If area is present in nldm.lib
    - Write it to output file
  - If drc_blockgroup, drc_block_type, drc_ground_pins, drc_power_pins, drc_iotype, pinout_trackpins, pinout_layout_classes are present
    - Write the values to output file
  - If pg_pin is present
    - Get pg_pin name and write it to output file
    - If is_pad attribute is present
      - Write IS_PAD=true in output file
    - If direction is present
      - Write it in output file
    - If type is present
      - Write it in output file
  - If pin is encountered
    - Get pin name and write it to output file
    - If drc_pinsigtype, drc_pinkind, IS_PAD are present
      - Write their values in output file
    - If direction is present, write it to output file

Figure 4.10: Algo of Reference view extraction

- **Layout view extraction**

Layout view's tree is dumped in this check.

Layout.csh

- Get path of bbview, CDS5 database, CDSOA database of test library
- Select between CDS5 and CDSOA database and source uniopus tool
- If CDSOA is selected
  - Call Layoutoa.tcl
- Else if CDS5 is selected
  - Call Layoutcds5.tcl

Layoutoa.tcl

- Get views from tasks file
- For each view in $views
  - Get path of the view and append it to list inputfiles
  - Make a list outputfiles that contains list of view.tree files
  - Execute libcomp tool to dump inputfile to $out.lef file
  - Read $out.lef file
  - Open $out.tree(cds_dfII_IPS_oa.tree) in write mode
  - If MACRO is encountered in line
    - Get cell name and write it in $out file
    - If SIZE is present
      - Get cell length and cell breadth
      - Calculate area= length *breadth
      - Write AREA=$area in $out file
    - If PIN is encountered in line
      - Get pin name and write it in $out
    - If DIRECTION is present
      - Write it in $out
    - If TYPE is present
      - Write it in $out

Layoutcds5.tcl

- Same as Layoutoa.tcl but for cds5 database (output file is cds_dfII_IPS_cdba.tree)

Figure 4.11: Algo of Layout view extraction

The work that is going on parallely with development of new module, is to remove

the bugs from IPScreen and develop the scripts for new views of different plugins. So that temporary the work can be carried by the clients. And at the same time the new module is also being developed which will take a couple of years for completion.

4.4 Screenshots

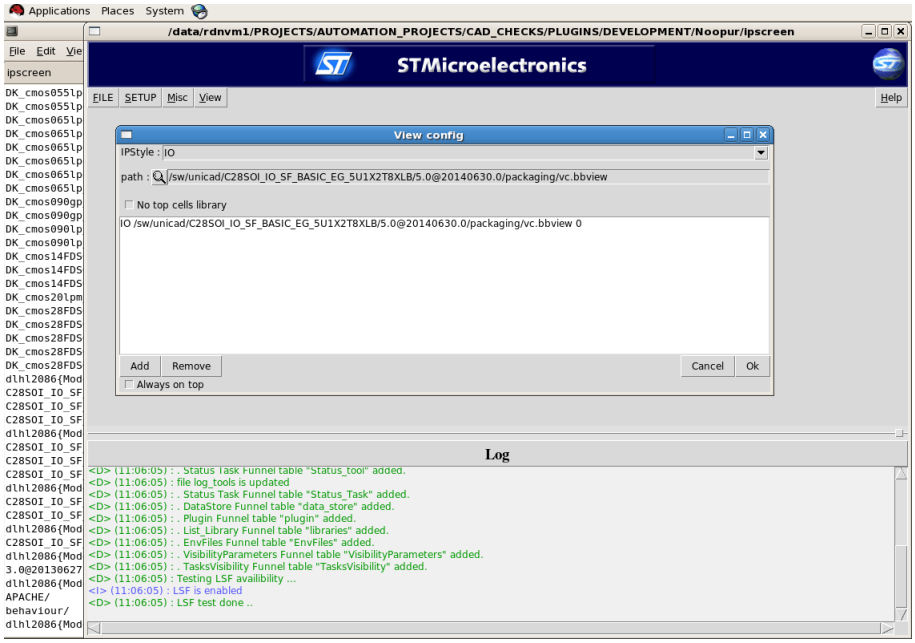

Figure 4.12: Library loaded in IPScreen

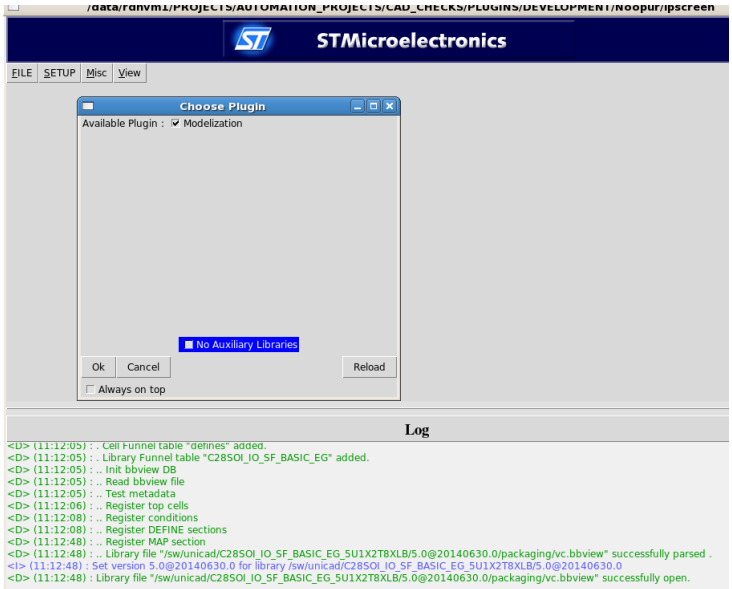

Figure 4.13: Plugin is selected

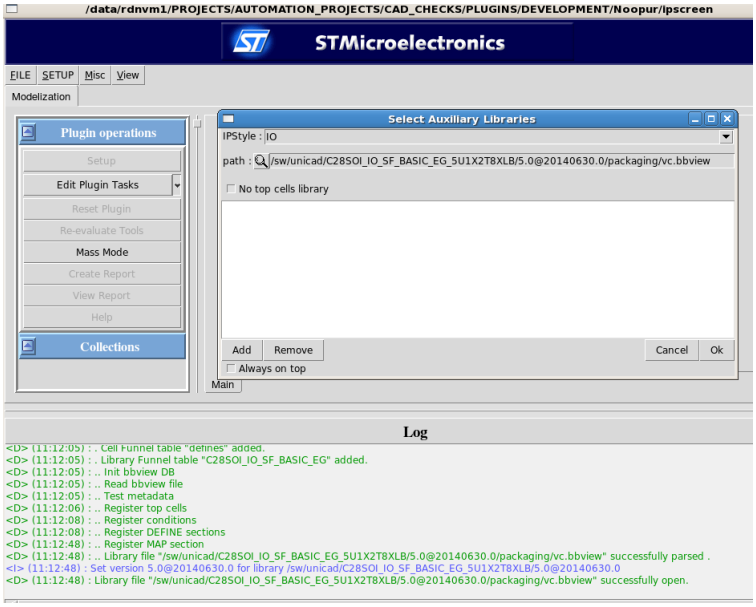

Figure 4.14: Reference library loaded

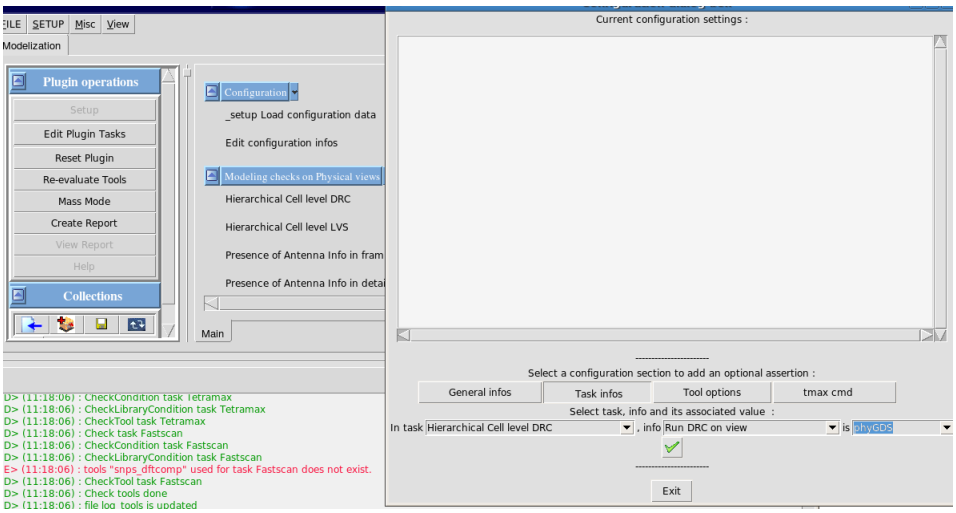

Figure 4.15: User provides option for Checks

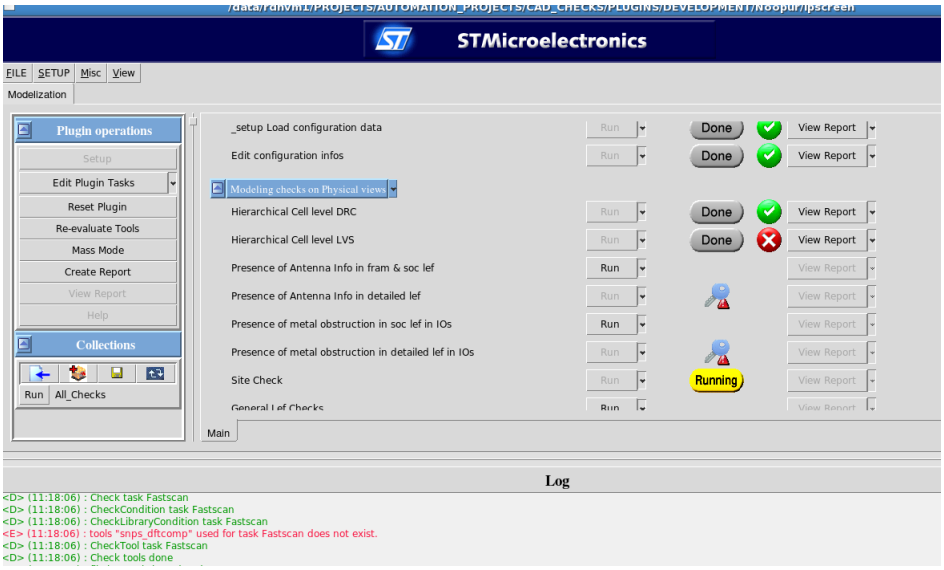

Figure 4.16: Checks of Modelization plugin

```
lhl2086{ipscreen}>> g Modelization/
b/                DRC_Hier/           edit_configuration/ load_configuration/ LVS_Hier/          Site_Check/
lhl2086{ipscreen}>> g Modelization/DRC_Hier/
ux_libs                close                  DRC_check.log          IPS_report/           objectMapTable*       physical_view_GDS     uk-init.il
28SOI_IO_SF_BASIC_EG/  cust                   DRC.rpt                LayerMapTable*        paths_file            test_libs             userRunsetConfigFile
ds.lib                 display.drf*           ipscreen_run.csh*      List                  physical_view_CDL     uk-get-prods.log
lhl2086{ipscreen}>> g Modelization/DRC_Hier/DRC_check.log
```

Figure 4.17: Run area of DRC check

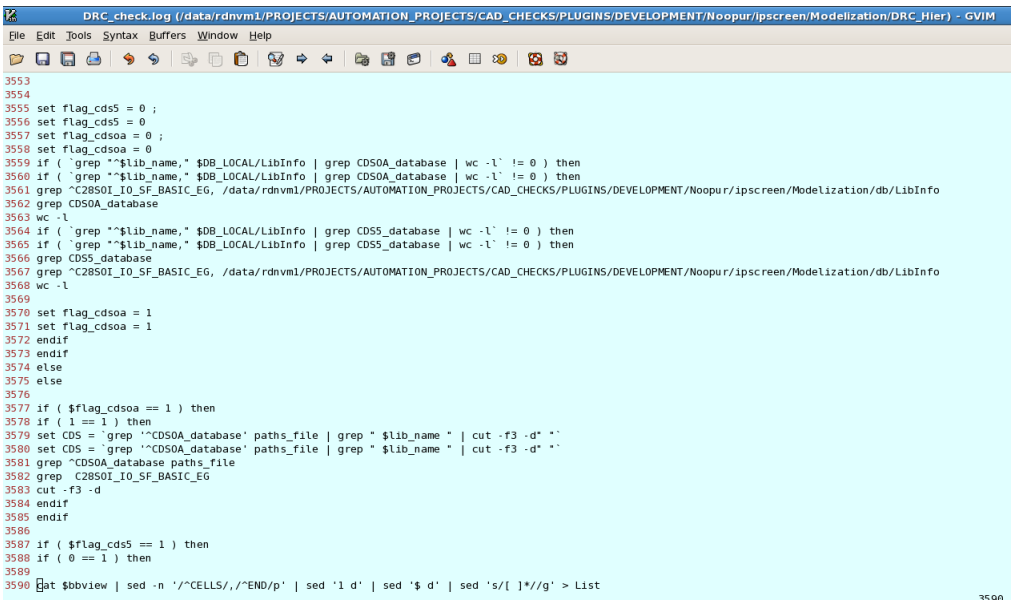

Figure 4.18: Log file of DRC check

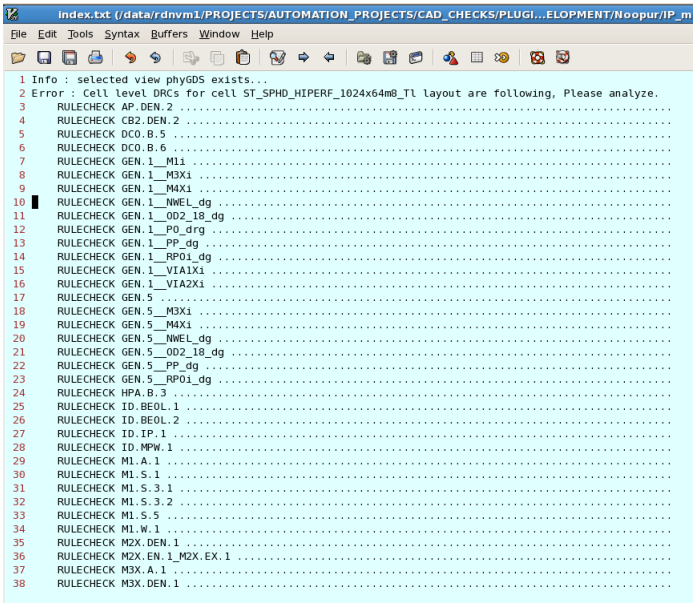

Figure 4.19: Report file of DRC check

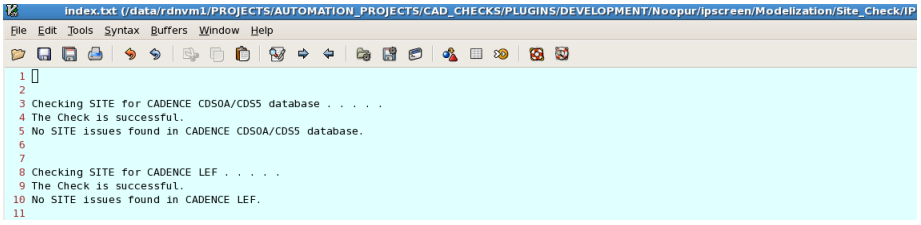

Figure 4.20: Report file of Site check

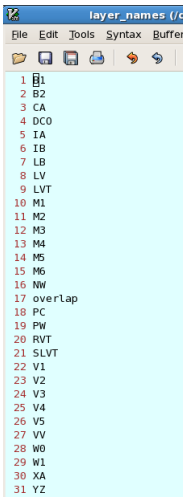

Figure 4.21: File containing layer names in General LEF check

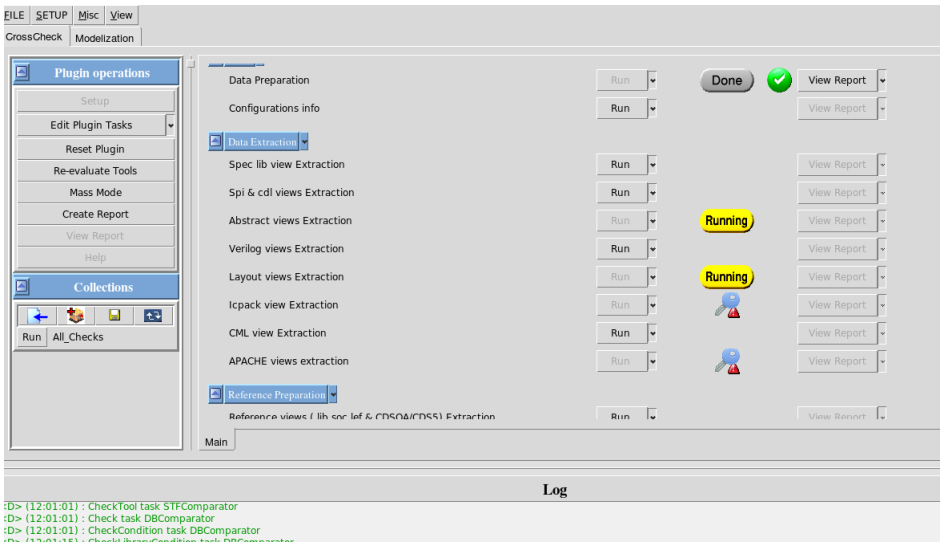

Figure 4.22: Checks in CrossCheck plugin

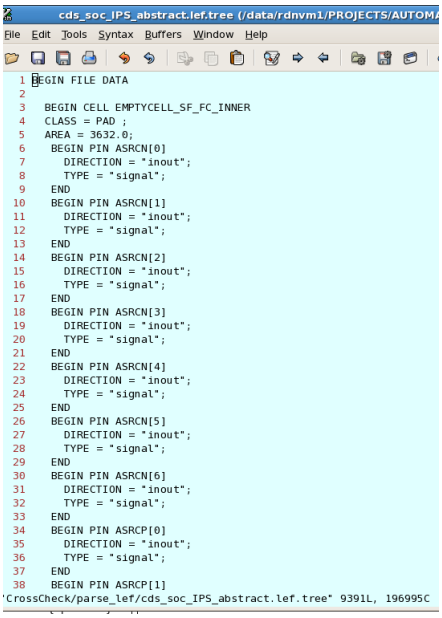

Figure 4.23: Lef tree

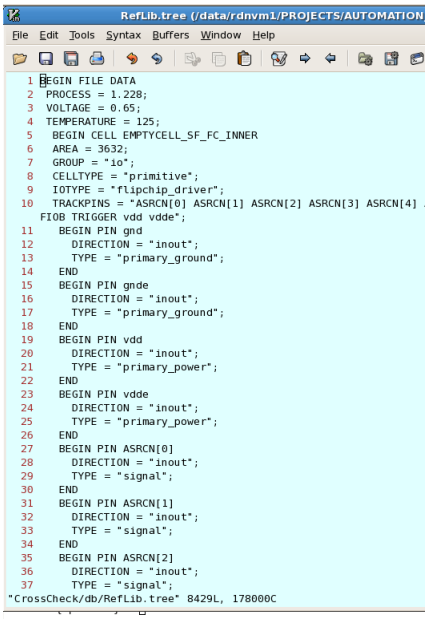

Figure 4.24: Lib tree

Chapter 5

Conclusion

Validation is one of the major step that certifies the quality of the product being provided to the user. Conventional validation approach has high cycle time and also requires high man resources so there came a need to go for Hybrid validation approach.
This validation environment has made it possible to reduce the cycle time to a large extent. Talking about any specific IP which generally takes 7-8 man days for validation following the conventional approach now can be validated only in a single man day. So this alternate approach has made a huge gain in run time for validating.

Bibliography

[1] M. J. S. Smith, "Application Specific Integrated Circuits,"

[2] W. Agatstein, K. McFaul, and P. Themins, "Validating an ASIC standard cell library," pp. P12–6, 1990.

[3] SYNOPSYS, "Astro User Manual, STMicroelectronics,"

[4] http://wiki.tcl.tk/299.

[5] https://en.wikipedia.org/wiki/C_shell.