IP VIEW VALIDATION AUTOMATION

Submitted By

Arpit Garg

13MCEI03

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

MAY 2015

IP VIEW VALIDATION AUTOMATION

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

(Information and Network Security)

Submitted By

Arpit Garg

(13MCEI03.)

Guided By

Mr Vivek Garg

and

Prof. Vijay Ukani

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

MAY 2015

Certificate

This is to certify that the major project entitled **"IP VIEW VALIDATION AUTOMATION"** submitted by **Arpit Garg(Roll No: 13MCEI03)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Information & Network Security (CSE) of Institute of Technology, Nirma University, Ahmedabad, is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this 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 Vijay Ukani
Internal Guide,
Assoc. Professor,
Nirma University, Ahmedabad

Prof. Sharda Valiveti
Program Co-ordinator-INS,
Assoc. Professor,
Nirma University, Ahmedabad

Dr Sanjay Garg
Head of CSE Department
Institute of Technogy
Nirma University
Ahmedabad

Dr K Kotecha
Director,
Institute of Technology,
Nirma University
Ahmedabad

Statement of Originality

I, **Arpit Garg**, Roll. No. **13MCEI03**, give undertaking that the Major Project entitled "**IP View Validation Automation**" submitted by me, towards the partial 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.

_____

Arpit Garg

Date:

Place:

Endorsed by

Prof Vijay Ukani

Acknowledgements

First and foremost, I would sincerely like to thank **Mr Vivek Garg**, Project Guide, STMicroElectronics, Greater Noida. I would like to thank, Project Manager **Mr Ashu Talwar** STMicroElectronics, Greater Noida for his constant motivation and support throughout the year. I would thank my thesis supervisor **Prof Vijay Ukani**, Internal Guide Nirma University for their guidance and support.

I would also like to express my gratitude and sincere thanks to **Dr. Sanjay Garg** Head of Computer Science and Engineering Department and **Prof. Sharda Valiveti** Coordinator of M.Tech Information and Network Security program for allowing me to undertake the thesis work and for her guidelines during the review process.

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

I would also thank the Institution, all faculty members of Computer Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

See that you acknowledge each one who have helped you in the project directly or indirectly.

- Arpit Garg

13MCEI03

Abstract

The project is related to automation of validation of libraries which are the Intellectual Property. These libraries represent design data of cells, transistor level design and timing information that will be integrated and fabricated on a chip. The automation requires plugins to validate different views of a library under test. The plugins are in TCL, C Shell. Launching the plugins on IPs and validation of the behavior of plugins is a time consuming process. The aim of this project is to automate the task of input preparation required for launching QA and simplify log analysis. PluginQAKit has been developed in order to reduce human effort and save developer's time to launch the plugins and validate the results of plugins

Abbreviations

IP	Intellectual Property(Library).
QA	Quality Assurance
TCL	Tool Command Language
EDA	Electronic Design Automation
LEF	Library Exchange Format
GDS	Graphical Data System
BASH	Bourne Again SHell

Contents

List of Figures

List of Tables

Chapter 1

Introduction

System on Chip(SoC) design trend is to combine more functionalities usually named IPs (Intellectual Property), that are developed in the design team or already available in market. One of the demanding jobs for IP providers is related to IP models or representation needed in the design flow targeted for the System on Chip(SoC) development. The main debate for IP developers are:

- Coherency between different Views.
- Accuracy between IP models and real IP.

This chapter consists of brief preface to library IPs, types of libraries.

1.1 Introduction of Library IP

LIBRARY [1] is defined as a set of all design data available for an IP. The design data consists of functionality, transistor level design of the IP, timing information that will be fabricated on chip. We can also say that library is a collection of cells, comprising of various views which are useful for designing a chip. Cell is a component performing a basic function and a view is a particular representation of a cell. Due to increased complexity of circuit and shorter time to market SoC designers cannot really concentrate on design of basic building blocks. This database can be reused in designing various System on Chip(SoC).

1.2 Types of Library

The different types of Library are as follows:

1.2.1 Core Library

It consists of a group of cells called Standard Cells. They implement basic logic functions like Inverter, latches and flip flops etc. Digital designs are build from basic components like gate, register, counters, adder, subtractors, RAM, ROM. Core Library is collection of such building blocks. Physical/logical/timing models are created for these cells.

1.2.2 Input/Output Library

It consists of a group of cells called I/O buffers. I/O buffers are designed to interface off chip signals to inside chip environment and vice-versa. I/Os are placed on the periphery of the chip. Any signal which comes from off-chip environment (external voltages are at a typical voltage of 2.5V, 3.3V or 5V) into the chip, must be checked by I/O for any discrepancy in its behavior other than defined by the core for that particular signal. If I/O finds any signal defying the expected behaviour from it, it modifies the signal so as to ensure proper functioning of chip.

1.2.3 Memory Library

These library contain memory of different architecture. Examples are SRAM, DRAM, ROM. As there can be number of memory sizes, we implement basic building block of memory and configure the generation of different memory sizes.

1.2.4 Analog and Mixed Cell Library

These library are implemented in a full custom or semi custom manner using the CORE library. Example of Analog and Mixed Signal Library are Digital to Analog Convertor, U.S.B, Phase Locked Loop(PLL).

Chapter 2

Literature Survey

2.1 Library Views

As discussed previously, Library is a collection of cells and each cell has views which are the representation of the cell. [2] All the cells have Layout view, Abstract view, Schematic view, Symbolic view, Timing view etc. A cell is delivered as a set of view and each view is used by different tool in a given electronic design flow.The basic Library views are as:

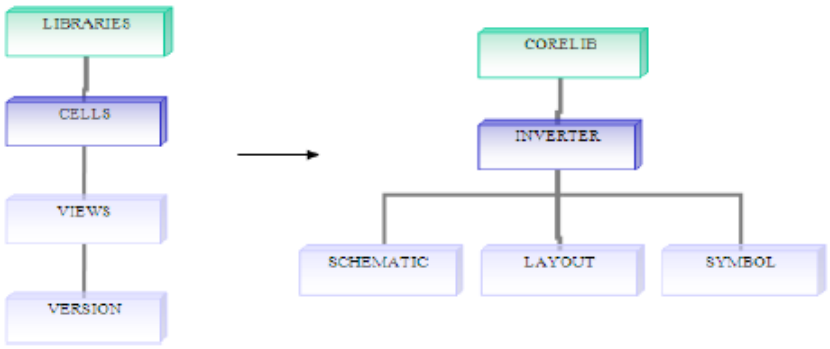

Figure 2.1: Library View Block Diagram

2.1.1 Symbolic View

It is the pictorial representation of cell. It includes pins, symbol, labels, selection box. Pins represent the input and output of the cells. The shape of cells indicate its function. Labels are mapped to some documentation of design, selection box select the complete area for the cells. Symbolic Views allows the user to abstract a complex Schematic View and replace it by a Symbolic view that can be used as in further designs. A symbol view

for an NOT gate is shown below:

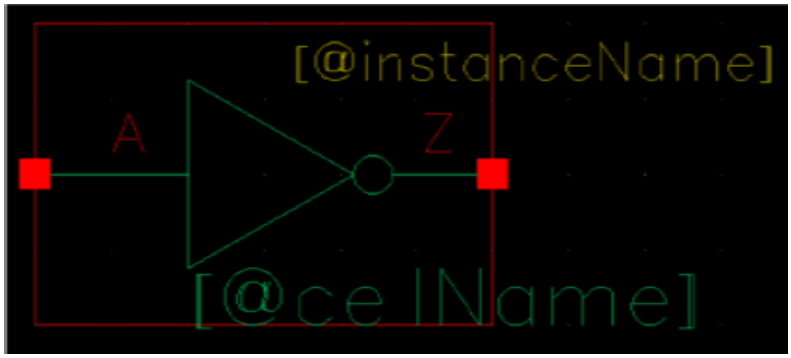

Figure 2.2: Symbolic View of a Inverter

2.1.2 SLIB

This view is derived from symbol view and is the textual representation of Symbol View.

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)

2.1.3 Schematic View

Schematic Views are map to designing, building circuit. Different components of circuit are shown as standard symbols along with power sources and signals.

2.1.4 Circuit Description Language(CDL)

It is the textual representation of Schematic View at Transistor level. Creating a sub circuit allows you to reuse the circuit multiple times in a design and in future designs.

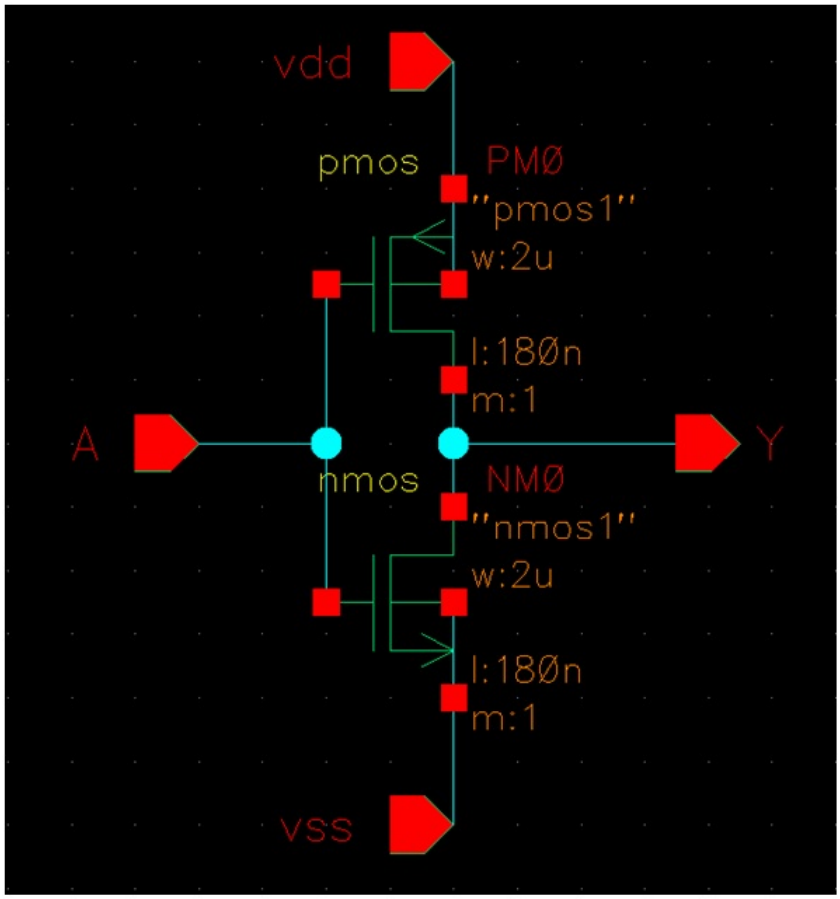

Figure 2.3: Schematic View of a Inverter as in Virtuso

Sub-circuits are similar to subroutines in software programming. It is representation of circuit at transistor level.It is automatically generated from schematic view. It is used in Layout vs Schematic Check and Electronic Circuit Simulation.

The sub-circuit for the circuit shown is as below:

SUBCKT Example_1 5 12 8

I 10 12 DC 10A

R_a 5 12 5.0

R_b5 13 4.0

R_c12 13 2.0

R_d5 18 8.0

R_e13 18 3.0

R_f10 13 1.0

R_g10 18 6.0

.ENDS

Here, in the example R_a is the resistance between nodes 5 and 12 having value 5.0, R_b is

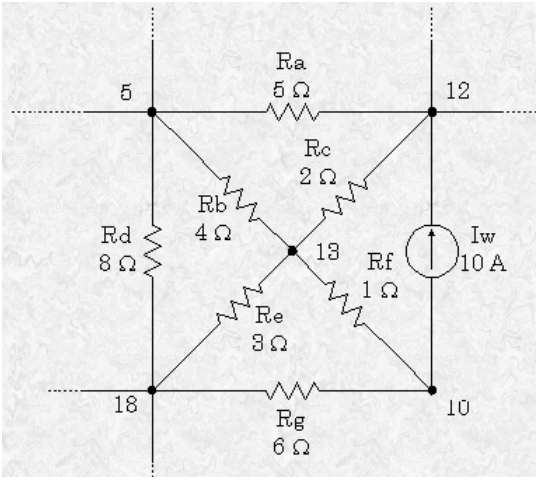

Figure 2.4: Sub circuit of a sample circuit

the resistance between nodes 5 and 13 which id 4 ohm. Current of 10A flows from node 10 towards node 12.

2.1.5 Layout Views

It is the representation of the circuit that goes to the foundry for fabrication. It is the representation of IC that in terms of geometric shapes which correspond to patterns of metal,oxide, semiconductor. Layout view must pass a sequence of checks during Verification process. The most important checks are Design Rule Check (DRC) and Layout vs Schematic (LVS). The parameters for such check are given by chip manufactures. A design rule set specifies certain geometric and connectivity restrictions to ensure that most of the part work correctly. Layout is also known as mask design, IC mask layout.

2.1.6 Lef View (Library Exchange format)

An ASCII data format file, to describe physical layout of an Integrated circuit. It includes design rules and abstract information about cells. LEF has only the necessary information required by the CAD tool. It provides only an abstract view and consumes less storage overhead.

A LEF file contains the following sections:

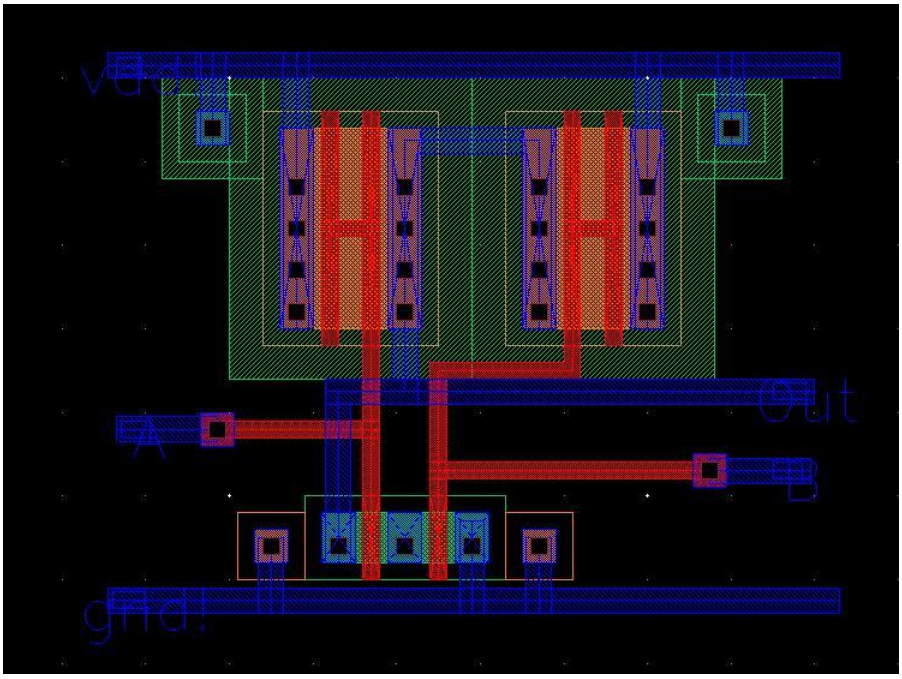

Figure 2.5: Layout View

- Technology section: layer, Placement and Design Rules, Via definitions, metal capacitance.

- Macros Cell definitions: cell descriptions, cell dimensions, layout of pins and blockages, capacitances.

  These two sections can be stored in two different file if the size of file becomes large. While reading the lef view the technology file must be read first before reading Macro Cell definitions.

2.1.7 Synopsis Technology File (.lib)

This file includes functional, timing and all the pin definitions. The functional part contains the logical operation that the cell performs. This file includes the delay model, documentation details, library units that is the unit of voltage, current, time unit, resistance, capacitance unit, technology family like CMOS, operating condition (normal pressure, temperature, voltage), default values for attributes like fanout, pin capacitance etc. The ascii form of library usually has an .lib extension. Before a library can be used it must be compiled to get lib.db, this is done using Design Compiler tool.

2.2 Structure of Library

The library is maintained by a index file .The structure of 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* example 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.

2.3 IC Design Flow

The steps involved in IC design are as follows:

1. **Design Specification:** It is the first step in IC design, here the design functionality is stated. All the requirements are clearly stated in terms of performance, speed, power, functionality. All the architectural part is stated clearly.
2. **Design Implementation using HDL:** Hardware description Language allow to implement a design without going into much architecture, simulate and verify the output.For example Rather than building a MUX in hardware, Verilog code allows us to verify the functionality.
3. **Synthesis:** A Register Transfer Language(RTL) is transformed into design implementation in terms of logic gates, using program called as synthesis tool. Examples of synthesis tool are Synopsys's Design Compiler and Candence's Ambit. At the end of this stage we have logic circuit in terms of gates and memories. The output of Synthesis is netlist. Netlist indicates all the devices and interconnection between them.
4. **Simulation:** This netlist is simulated to verify the functionality of gate level implementation of design.

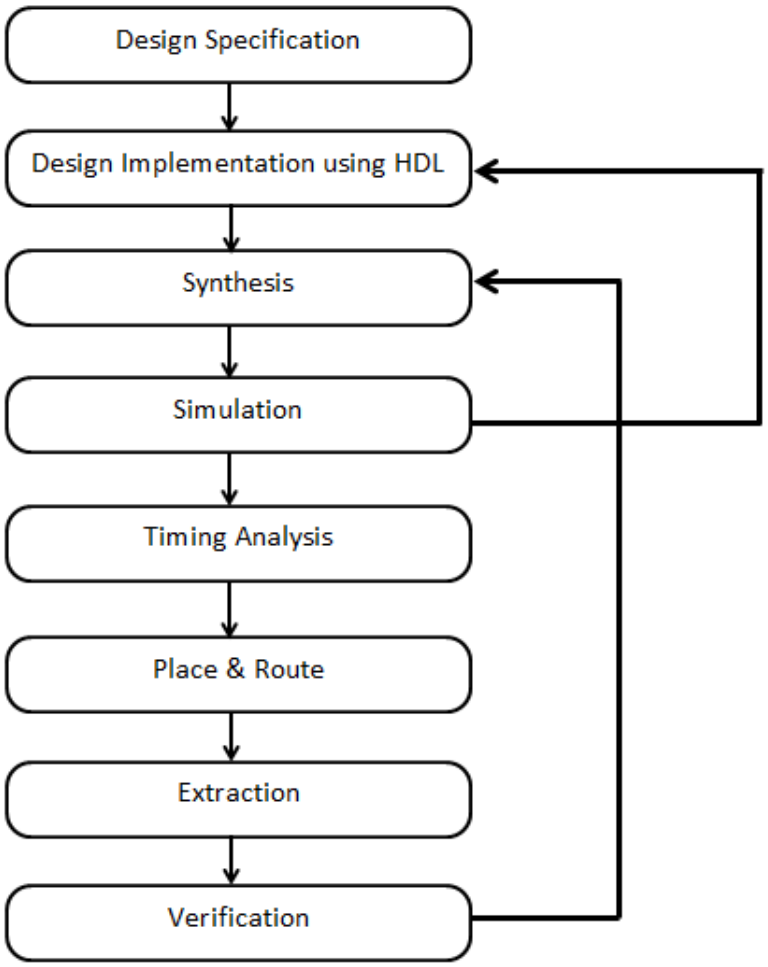

Figure 2.6: IC Design Flow

5. **Timing Analysis:** RTL and gate level simulation does not take into account time delay in signal propagation from device to device. It is the method of calculating the timing of digital circuit without simulating the circuit.Delays are errors that arise due to clock skew, gate delays etc. This step ensures that the circuit meets the required timing criterion.

6. **Place and Route:** This step of IC design decides where to place logic design, components in a limited amount of space. Routing decides where to place wires in order to connect the placed cells. This step implements all the connections handling along with the space rules and constraints.

7. **Extraction:** During this step parasitic calculations are done in devices and interconnections to find parasitic effects like inductance, resistance, inductance. As the

design of ICs gets compact, more are the chances of parasitic components. This interfaces in the functioning and performance in terms of timing speed and power consumption.

8. **Verification:** It would either be tape out stage or the design is again send through the same flow for the purpose of optimization or modification.

Chapter 3

Technology and Tools

Scripting Languages like Perl, Shell Script, Bash, Korn Shell, Python are used for automating small tasks. These languages are interpreted and not compiled. They are dynamically typed, that is type checking is done at run time and not at compile time. They are used for rapid application development and are good at string processing. They allow batch jobs that would be entered manually entered on the command line to be executed automatically one after another using scripts. Also, writing the scripts is more fast than equivalent code in other programming language. The main advantage of scripting languages is that the commands and syntax are exactly same when used at terminal. Also, interactive debugging, easy file selection, quick start are other advantages. The plugins are developed in C Shell and TCL.

3.1 C Shell

C Shell is a command interpreter with a syntax similar to the C programming language. It is used both as an interactive login shell and a shell script command processor. It provides I/O redirection, Joining of multiple commands using '&&' , ';'. **Piping** provides output of one command to be given as input to another command, the advantage is that both commands run in parallel. C Shell provides **Command Substitution** which allows output of one command to be taken as argument to another. **Background Execution** *command &* means to run command in background and prompt immediately for a new command line(command followed by ampersand). C Shell provides **Control statements** like if, while, foreach, switch etc. It also provides **SubShell**, in which a child copy of

current shell is created inheriting the current state, without affecting the parent. Shell script provide -x option for debugging by displaying commands as they are executed, -v option to display all lines as they are read.

3.2 Bash (Bourne-Again SHell)

BASH is the Bourne -again SHell. BASH is a sh compatible shell having many useful features from Korn Shell(ksh) and C Shell(csh). It has functional improvements over the sh for both programming and interactive use. The improvements offered by bash include

- Shell Functions and Aliases . An alias allows a string to be substituted for a word when it is used as the first word of a simple command.
- Bash provides one dimensional array variables. There is no maximum limit on the size of array. Neither there is a requirement that members can be indexed or assigned contiguously.
- Integer arithmetic in any base from two to sixty four.
- Unlimited size command history.
- Directory stack, which is a list of recently visited directories. The pushd built in adds directories to the stack on change of current directory, and popd removes specified directories from the stack and changes the current directory to the directory removed.

3.3 Tool Command Language(TCL) [3]

TCL is commonly used for rapid prototyping,scripted applications, GUIs and testing. The main features of TCL are:

- The TCL uses tclsh command line interpreter. Other ways of starting TCL are using Wish or Windowing Shell.
- All data types can be manipulated as strings.
- TCL interfaces natively with C language.

- TCL interpreter performs run time compilation of script into byte code. Running compiled code allows TCL script to run faster than Perl.
- TCL supports most of modern programming constructs like subroutines, standard programming flow constructs, rich set of variable type like lists, associative array,float, integer, string etc.
- TCL provides powerful string manipulation commands for searching and replacing, extracting portion s of string, converting strings to list.
- Extensibility: TCL extension add a few new commands to extend interpreter into new application domain.
- Provides GUI interface Tk.
- All commands of TCL generate error message on incorrect usage.

3.4 VNC

VNC allows you to remotely get to and control your PCs from another system. It provides Cross Platform remote control. All the connections are encrypted using 128-bit AES. You deploy VNC on system that you want to control and VNC viewer on system that you want to control from. The Enterprise edition provides features like file transfer, dedicated support channel, multi language support etc, remote deployment strategies, remote configure techniques etc.

3.5 GMake [4]

GMake (GNU Make) finds which pieces of a large program need to be recompiled, and issues commands to recompile them. Make uses the information on how to build your program from a file called MakeFile. The features of Make are :

- Make allows any user to build and install package without the knowledge of how it is done.
- Make automatically determines which file is to be made upto date on the basis of timestamp of each file.

- Make is not limited to any particular language. For each non-source file in the program, the makefile specifies the shell commands to compute on it. These shell commands can run a compiler to produce an object file, the linker to produce an executable, or TeX to format document.

3.5.1 Concept of Make rules and Target

Make file consist of targets and rules. Rules are commands that tell what to execute in order to build the targets from the source file. The command line starts with tab. It specifies the list of dependencies(prerequisite) of the target file. The list includes all files that are input to commands in the rule. The simple rule looks like :

target : dependencies

commands(recipe)

...

A rule tells two things, firstly the targets that are out of date, and how to update them when necessary.

Chapter 4

IPScreen and PLUGINS

IPScreen [5] is a framework built using TCL and Tk on which library are loaded for validation. Multiple libraries of different technology can be loaded at the same time. Plugins are loaded after the library are loaded on IPScreen. Each tuple of library, plugin corresponds to a new tab in IPScreen window. The IPScreen can be run in GUI mode or else in Batch mode, in GUI mode the user has to click on each task that they want to run. To run in batch mode all the commands are given in a file as input.
Depending upon the availability of license for tools, the execution time of the check may vary. Also the time of execution varies on the number of cells in the library and size of each cell.

4.1 Inputs to IPScreen

- **Command File**: All the check(s) that are to be performed are given in this file along with libraries.
  The contents of command file are: command to load library, command to load auxiliary library, command to run specific task.

- **Tools, Plugin record**: Depending upon the techno of library(65nm, 32 nm etc) the tools version may change, so correct tools are to be ensured for each validation of library. These tools are Electronic Design Automation Tools (EDA tools). All the plugins that are to be used, are specified in a separate file along with their path.

- **Setup:** The setup script sets the environment in order to execute tasks on load

sharing facility and other environment variables required.

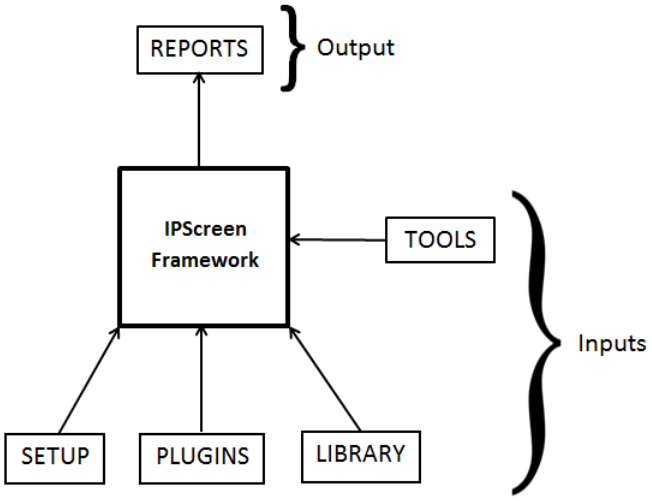

Figure 4.1: IPScreen and Plugins Block Diagram

The above block diagram shows the input required by the IPScreen Framework and the final reports as output. Command File, tools record, plugin record, setup are the inputs required.

4.2 Information Dumped by IPScreen on parsing an IP

After an IP is loaded and parsed by IPScreen framework, the information dumped by IPScreen is used by plugin for its specfic check. The developer of plugin may use this information as per his choice. The structure of the dumped information is plugin independent.

- Status of Tools : It tells {tool name, status of tool}, where status of tool is missing/ present.
- Library Information : For all the libraries loaded following information are dumped, {Name, Version,Type, path, path till index file, product name, iptype.}
- Status of all Collection having {collection name, Status} , where Collection status is failed, warning, Done correctly.

- Status of Task: It has plugin name, task name, tool status, task status.
- List of all cells present in a specific library.
- Path of each view for each cell, condition, plugin, type of library (main or supporting library).
- Pairs of all plugin and library that are to be used.

On running, aborting, rerunning a task the Status of Task table is updated. On loading, reloading a tool Status of tools table is updated. Initially when Library is loaded Library table is created. On running and re running a collection, Collection status table is updated. All the tables are initially dumped by IPScreen, and are later modified during task run.

4.3 Final Output of Plugin Run on IPScreen

The primary level report shows "No Error / Warnings found" if execution is correct and input library is correctly validated, a green tick is shown on IPScreen GUI along with "Done". If during the execution of script, if the scripts aborts due to abnormal program execution, then the task fails and primary report is not prepared, so FAILED is shown on the IpScreen GUI along with Red Cross. If there are errors present in library then the report shows "FAILED" message, and IPscreen GUI shows "Done" along with Red Cross. If there are no errors but warnings like ".v file not present in library" then "Warn" message is shown in report, along with Done on IpScreen GUI, and brown colour Tick indicating warning. The final xls reports are hierarchical and tell status of each check.

4.4 Plugins

Plugins are the collection of checks that validate the IP. They are classified on the basis of taxonomy that is, to ensure consistency between views, there is one plugin. To ensure alignment with library structure there is different plugin. To validate the different tags present in the library there is another plugin. These plugins are built in TCL and C Shell.

4.4.1 Old Approach vs Plugins

These plugins bundle collection of checks. Prior to automation, there were two ways in which the validation was done :

- Some checks required the user to manually check all the relationship between various different views.
- By giving input to EDA tool through command line or GUI, the results had to be collected, and analyzed for each cell. This would take weeks and the results of validation may still be prone to human errors.

The different plugins are as below:

- Plugin to ensure Views consistency
- Tags Evaluator.
- Plugin to Ensure alignment with Library Structure.
- IP Comparator.

4.4.2 Plugin to ensure Views consistency [6]

This Plugin is designed to ensure view consistency. The aspects of this plugin are **Similar View Consistency**: Ensure that the same type of views are consistent among them self. **DifferentView Consistency**: Ensure that all the different type of views are consistent among each other example:(.lef vs lib, .lef vs .v).

The tasks performed in this plugin are as follows:

1. **Data Extraction**: During this step the layout view are dumped into .lef format and from the .lef a common tree format is built, the tree format includes information like cell name, cell area, pin name, pin direction. Also, the abstract view is dumped in a common tree format using ST internal tools, and information like cell name, cell area, Pin name, Antenna Information, Pin direction, Pin type are dumped into a textual format. Similarly, Verilog Views, Apache views are extracted and dumped into a common tree format.

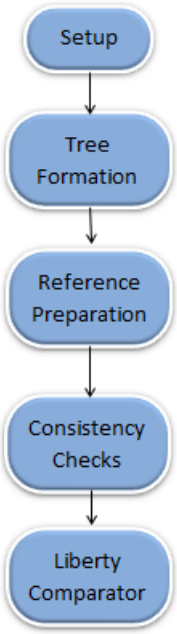

Figure 4.2: Plugin Flow: Ensure Views Consistency

2. **Reference Preparation**: During this step, the reference trees are prepared and are then compared among themselves to ensure consistency among themselves. Here, .lef, .lib and layout are compared among themselves. Out of all the reference views one view is choosen as reference further consistency check.

3. **Consistency Check**: The consistency between all the trees dumped in Tree Formation steps checked with respect to reference views.

4. **Liberty Comparator**: This check check the consistency between .lib and .db respectively.Also it checks the consistency between .lib and reference choosen.

4.4.3 Tags Evaluator

This plugin [7] is dedicated to different tags that are present in the library. The library must be aligned with the tag specification, which is given as input during Setup task. The tag specification input comprises of Vendor, Product, Version, CellType, Cell ID, Tag Date, Library. The plugin can be run in both batch mode and GUI mode. The detailed working of Tag Checker is as below:

1. **Setup**: Firstly all the required environment is setup, and the input tag Specification is given.

2. **Dumping of G.D.S. views**: During this step all the layout view are dumped into text files using dumping tools.

3. **Checking of dumped tags**: In this step, the dumped tags will be compared to the specification input given. Existence of duplicate cells in done, checking that all the top cells present in index file are tagged in the dump. Check Vendor, if different vendor name is present in the same library, and vendor name does not match with a valid vendor name then, the plugin reports error. Check Techno is done to ensure that the techno value of all the cells should be equal to that in Tag Specification.

4. **Cross Existence**: This check will be launched if atleast two different dumps of layout view exist. Suppose two dumps A and B exist, this check will be executed in two ways that is from A to B and B to A. It compares the cells tagged in view1 with the cells tagged in view2, and reports warning if cell is missing in view2. Then it compares the cells tagged in view2 with cells tagged in view1 and reports warning if tagged cell is missing in view1. If all the dump exist, then this check is done on all the layout dumps pairwise. Product name, product Version are also checked.

5. **Cross Compare**: This check is done, if atleast two dumps exist. Here, check is performed for all the fields of tags between two dumps, it checks all the value of fields of tag of one cell in view1 to same cell in view2. If all dumps exist then this check is done pair wise on all the dumps.

4.4.4 Plugin to ensure alignment with Library Structure

This plugin compares the library with the library structure. The library structure is predefined. Some of the checks are as follows :

- Checking of indexation : This check verifies that all the all the paths present in the index file are physically present the library. If for some view there is no such physical presence file or directory in the IP, then error are reported. Also, it checks for symbolic links, hidden files, locked files.

- Index File Verification: This check verifies the sections of index file are syntactically correct or not. i.e. header section, cell section, mapping section.

- Checking of mandatory views: This task verifies that all the mandatory views as per the library specification are present in the IP. It reports error if some mandatory views are absent, or some extra views are present.

4.4.5 IP Comparator

This plugin compares different versions of the same IP. It is used to see how new IP version is different from the old version. It tells all the files, views that are present or absent in the newer version of IP.

Chapter 5

Implementation Details and Results

5.1 Manual Approach for Validation.

In the manual approach for IP Validation, the user had to firstly launch IPScreen window, manually locate the path of the IP to be loaded, specify its path and wait till the library gets completely parsed by IPScreen. Secondly, the user then had to check if any supporting libraries are needed(auxiliary library) by the main library, if yes then these all supporting libraries are to be manually loaded so that all the IP data is parsed and dumped by IPScreen. Depending on the size of the supporting IP, IPScreen takes time to parse the IP and generate the dump.
The user had to find all the latest version of the plugins needed and specify their paths in the IPScreen window. The user had to manually find all the correct version of tools based on the technology of IP and write them in the tool record file. On launching the validation, then all the tools are evaluated based on the technology of input IP. If some missing tools are found, the cause of missing tool needs to be analyzed and fixed. Reasons for missing tool error can be incompatible version of tool may be loaded due to human error, it may be absent in the tool record, wrong tool name or its version.

5.2 Intermediate PluginQAKIt

Firstly by using an ST Internal tool, tool record and plugin record are prepared from technology specific data. This technology specific data is decided and developed by the Management, and stores all the valid tools their versions, specifies the tools based on the

type of IP (Memory/IO/CORE) etc.

I created a module that automatically parses some input given on command line and prepares input command file as in figure 5.1. The input given to module was name of library to validate, all its supporting IPs, path of the plugins to use, working directory, option to append specific command in the generated command file. An example help option is also provided that shows syntax of command.

After the input command file is prepared, and tool records, plugin records are prepared the user automatically launches the job on the Load Sharing Facility. The validation cannot be done on local machines as the tools used require large compute resources and Licensed Tools. Licenses are uniformly given to jobs submitted by users by the Load Sharing manager in a uniform manner. Also, the compute resources required are available on LSF. The figure shows block diagram of Intermediate PluginQAKit.

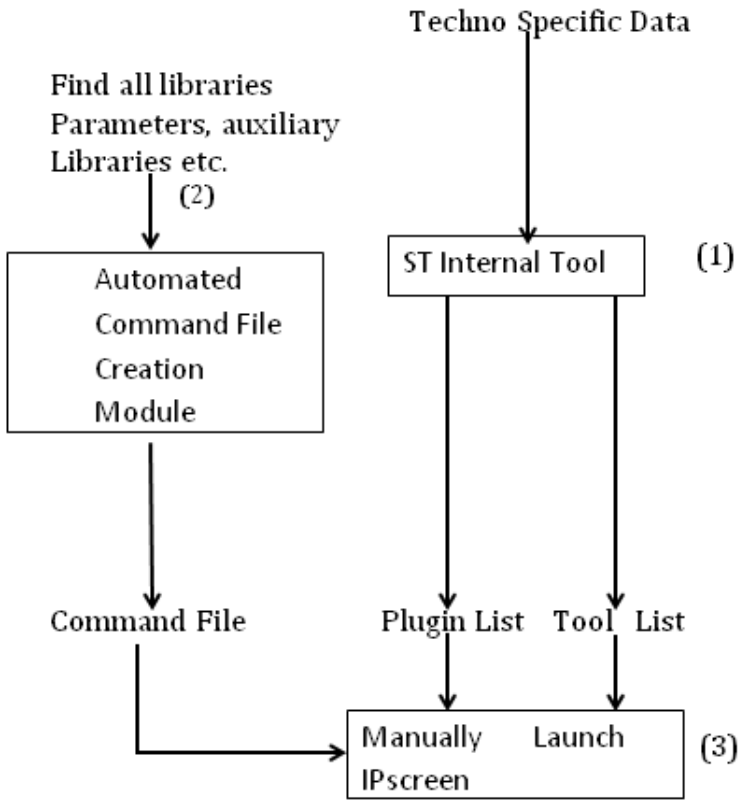

Figure 5.1: Block diagram of Intermediate PluginQAKit

The main advantage of using Intermediate PluginQAKit was that overhead of creating tool list,plugin list and command file was removed.

5.3 Algorithm for automated Command File Creation

The input given for automated command file creation is of the form.
Qa_kit -libraries "l1" -supp_lib "l2 l3" -plugins "plugin_name plugin_param plugin_libs task_field collection_name" -work_directory A/B/c -output command_file -run yes.
where,

l1 is the name of the library to be validated.

l2 and l3 are the supporting libraries for l1.

plugin option specifies five fields namely plugin_name, **plugin_param** ie : any parameter to pluginname (it is of the form of File "File path"). It is optional and is plugin specific. **plugin_libs** specifies the libraries to be validated by **plugin_name**, **task_field** tells all the indvidual tasks to execute, **collection_name** are collection present in the plugin_name.

A/B/c specifies the path to run area to be created at run time.

run yes means to launch ipscreen after command file creation.

Below are the steps to create ipscreen command :

1. Source all the required TCL packages and ST internal tools.
2. If **work_directory** option is not specified then, set work_directory to current directory, else create the specified work_directory.
3. If name of output command file is not specified set the name "ipscreen.cmd" as the default command_file else use the specified name in **output** field.
4. For each parameter of **-plugin option** parse **plugin_name**, **plugin_param**, **plugin_libs**, **task_field** and **collection_name**.
5. Link all the libraries specified in **-libraries option** using an ST internal command. A symbolic link is created by this command to the specified library, the actual library is present in one of the central location.
6. For each specified library as argument to **lib option**, check if index file exists in the locally linked library in the **work_directory**, if no set **run_ipscreen** as failure.

(a) If index file exists, set the iptype, parsing the METHOD field in index file.

(b) Parse the index file to get the library_name.

(c) Append in newly created command_file "LoadIP lib library_name -ipstyle iptype path_to_index_file".

7. For each plugin_name, check if plugin_library equals *, if yes set all the libraries to be loaded for this plugin name, else set the specified library **(plugin_libs)** to libraries to be loaded for the plugin_name.

8. For each **plugin_name**, Write command SetupPlugin -lib Libname plugin_name in the command_file.

9. For each plugin_name,

   (a) If Collection field is *, Check if collection table exists inside plugin, for all collection_name present in the the collection table append "RunCollection -plugin plugin_name -lib Libname collection_name" in the command_file.

   (b) If Collection field specified is not *, and is **collection_name**, check if the collection_name is present in the plugin, if yes write command "RunCollection -plugin plugin_name -lib Libname collection_name", if collection_name is absent in plugin raise an error.

10. For each plugin_name in -plugin, check if **task_field** is present in parsed plugin_name field.

    (a) If **task_field** is *, then extract all the task_name from the plugin, and write command "RunTask -plugin plugin_name -lib Libname task_name".

    (b) If task_field is not *, check if the specified task exists in the plugin, if yes, write command "RunTask -plugin plugin_name -lib Libname task_name".

11. If the **run_ipscreen** flag is set to yes and **gui option** is present and is yes, set the current work_directory to work area created, and execute "ipscreen -f command_file" , if -gui option is no or absent, then execute "ipscreen -nodisplay -f command_file". If **run_ipscreen** is set to failure display "cannot start ipscreen due to fatal error at startup".

5.4 Plugin QA Kit

In order to further ease the task of regressive QA sessions before release of any plugin, we have maintained a common technology specific repository which has correct library name and names of its supporting libraries as in Figure 5.2. This repository is referred at run time by automated QAKit, and all the required libraries are accessed by reading their names from here. As in figure 5.2 there are different directory for each techno like 65 nm, 32nm, 28FDSOI etc. Each of them have corresponding library data for them. Creation of tool record file has been bypassed in final QAKit, as it is picked from the technology specific repository. Plugin record are generated at execution time based on the path given by user. The tools records holds all the correct version of tool and version as per the the technology of IP.

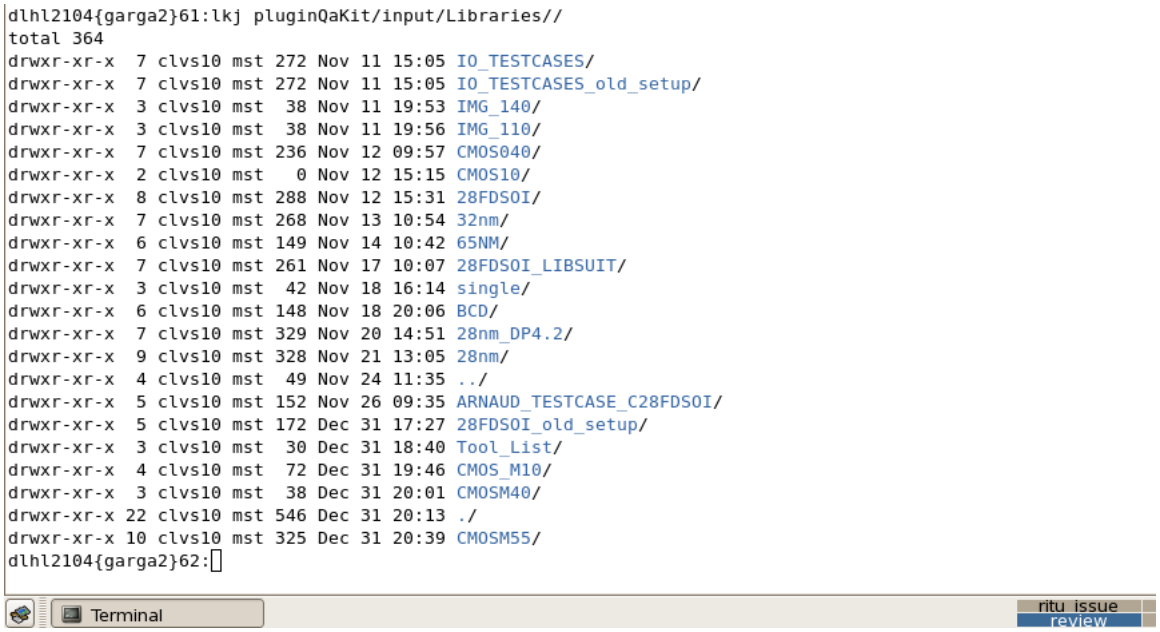

Figure 5.2: Illustration of technology specific library repository created for Automated QA Sessions.

In order to run extensive QA session, different libraries of IO, memory, core etc have been considered for designing the Library repository. All the IPs of a particular technology have a common tool record, so a single correct tool record is stored for each technology. The pluginQAKit has been developed in Sh.

As in the figure on next page, techno1 data store has required library name and its supporting library names, similarly techno2 may have n number of library sharing the

common tool record. On run time the correct library name and all its supporting IPs are automatically picked from this data store. At run time the specified library are automatically downloaded in our area.

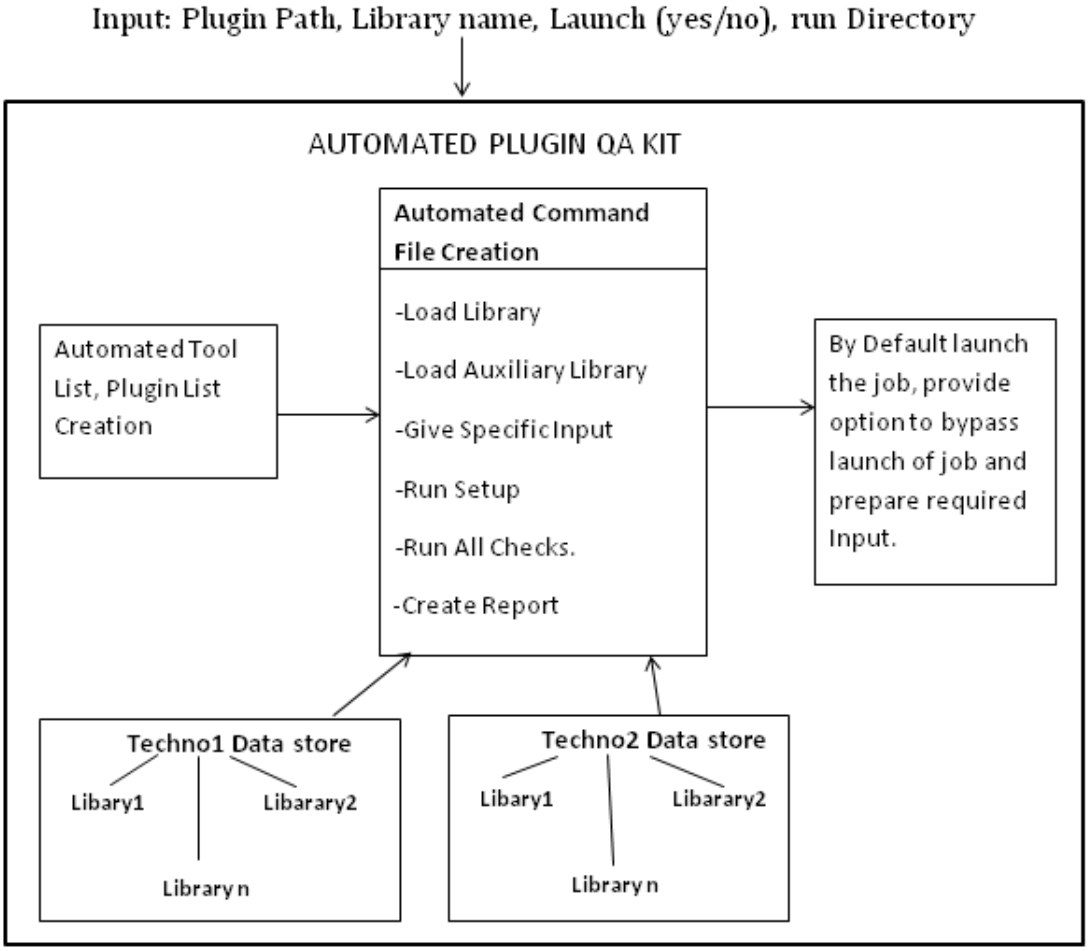

Figure 5.3: Block diagram of PluginQAKit

Examples of input command given :

- qa.sh -pluginpath $<$ *path_pluginA* $>$ $<$ *path_pluginB* $>$ $<$ *path_pluginC* $>$ -testLib $<$ *Lib*1 $>$ $<$ *Lib*2 $>$ -runDir $<$ *Directory_name* $>$

  Using the above command, the user runs all the Validations checks of pluginA, pluginB and pluginC on Lib1, Lib2 by creating the directory Directory_name. If no directory is specified then by default current directory is used. The submission of job is done automatically.

- qa.sh -pluginpath $<$ *path_pluginA* $>$ -testtechno $<$ *techoA* $>$ -setupOnly

  Using the above command only input will be prepared for all the IPs present inside

technoA ie : plugin list, tool list, command file, and the required environ ment will be sourced, and the user can latter manually launch the validation job.

- qa.sh -pluginpath $< path\_pluginA >$ -testtechno $< technoA >< technoB >$ -runDir .

  This command will prepare a command file that will have commnands to Load all the Libraries under technology technoA and technoB in the current directory and will launch the validation for the same.

Figure below shows the run area generated after execution of on single library (C32_SC_12_CLK_LR). As it can be seen in figure the plugins and ipscreen are symbolically linked to there actual location.

```
dlhl2104{garga2}50:lkj FINAL_QA_WITH_IO_FIXES/output/Libraries/28FDSOI_LIBSUIT/C32_SC_12_CLK_LR/
total 2715
drwxr-xr-x  7 garga2 mst     261 Jan  8 17:35 ../
lrwxrwxrwx  1 garga2 mst      59 Jan  8 17:35 SyntaxCheck@last -> /data/mass/CLVS/DELIVERY_TO_ARNAUD/PLUGINS/SyntaxCheck.v4.2/
lrwxrwxrwx  1 garga2 mst      60 Jan  8 17:35 Modelization@last -> /data/mass/CLVS/DELIVERY_TO_ARNAUD/PLUGINS/Modelization.v4.2/
lrwxrwxrwx  1 garga2 mst      58 Jan  8 17:35 CrossCheck@last -> /data/mass/CLVS/DELIVERY_TO_ARNAUD/PLUGINS/CrossCheck.v4.2/
-rw-r--r--  1 garga2 mst     363 Jan  8 17:35 .plugin.list
drwxr-xr-x  2 garga2 mst      58 Jan  8 17:35 .ipscreen/
-rwxr-xr-x  1 garga2 mst     794 Jan  8 17:35 run.csh*
-rwxr-xr-x  1 garga2 mst    1822 Jan  8 17:35 setup.csh*
-rwxr-xr-x  1 garga2 mst    2374 Jan  8 17:35 ipsSetup_warpper*
-rwxr-xr-x  1 garga2 mst   23950 Jan  8 17:35 ipsSetup_src*
-rwxr-xr-x  1 garga2 mst     365 Jan  8 17:35 extraxt.sh*
lrwxrwxrwx  1 garga2 mst      70 Jan  8 17:35 ipscreen@last -> /home/gargv2/Regressions/upt_cache/ipscreen/ipscreen_Christophe/5.0-00/
-rw-r--r--  1 garga2 mst     417 Jan  8 17:35 .ucdprod
-rw-r--r--  1 garga2 mst    1195 Jan  8 17:35 ucdprod
-rw-r--r--  1 garga2 mst    7692 Jan  8 17:35 .env_before_ukerEnv
-rw-r--r--  1 garga2 mst     335 Jan  8 17:35 uk-lib-link.log
drwxr-xr-x  3 garga2 mst      34 Jan  8 17:35 LIBRARIES/
-rw-r--r--  1 garga2 mst    7012 Jan  8 17:35 ipscreen.cmd
-rw-r--r--  1 garga2 mst       0 Jan  8 17:35 ipscreen.res
-rwxr-xr-x  1 garga2 mst      50 Jan  8 17:35 .flexlmrc*
-rw-r--r--  1 garga2 mst   31001 Jan  8 17:37 tools.log
drwxr-xr-x  2 garga2 mst      25 Jan  8 17:38 .testLSF/
drwxr-xr-x  2 garga2 mst    3640 Jan  8 18:16 .fontconfig/
drwxr-xr-x 44 garga2 mst    3333 Jan  8 19:16 tools_log/
drwxr-xr-x 25 garga2 mst     696 Jan  9 09:36 Modelization/
drwxr-xr-x 51 garga2 mst    1951 Jan  9 11:56 SyntaxCheck/
-rwxr-xr-x  1 garga2 mst    2796 Jan  9 15:59 .tool.list*
drwxr-xr-x 19 garga2 mst     505 Jan  9 16:14 CrossCheck/
-rw-r--r--  1 garga2 mst 1181382 Jan 14 15:10 ipscreen.log
-rw-r--r--  1 garga2 mst    9053 Jan 14 15:10 ipscreen.his
drwxr-xr-x  4 garga2 mst      46 Jan 14 15:10 archives/
-rw-r--r--  1 garga2 mst   53774 Jan 14 15:10 ipsSetup.log
drwxr-xr-x 13 garga2 mst     968 Jan 14 15:10 ./
drwxr-xr-x  3 garga2 mst      27 Jan 14 15:10 session_report/
drwxr-xr-x  5 garga2 mst     562 Apr 22 17:16 DB_FUNNEL/
dlhl2104{garga2}51:
```

Figure 5.4: Run Area generated for a single library (C32_SC_12_CLK_LR) showing that all the plugins are symbolically linked

5.4.1 Automated Command File Creation Module

This module is a part of Intermediate PluginQAKit. It manages creation of all the type of commands that can be given as input to ipscreen. The tasks performed by automated

Command creation module are as follows:

- The commands that are created are Command to load an IP, command to load any supporting libraries if present. Command to load IP includes the path till index file, its name as specified in the product description file and its IPtype as specified in the IP index file.
- Command to choose the desired plugins.
- Command to specify which plugin(s) to load on a specific IP.
- Automatically specific the RunTask commands that specify the name of the plugin, "task name" that is specific to plugin and the IP on which it is to be executed.
- At the end, when all the tasks are executed, the generation of xls report or portable document format (PDF) report may be optionally generated in the input command file.
- If the user wants to run all the tasks then a single option on command line writes all the Runtask command in the command file.

5.4.2 Algorithm for Automated Setup Creation using PluginQAKit

Below is the description of how all the input preparation is handled by PluginQAKit, which is developed in BASH.

1. Check if PluginQAKit is sourced, if not raise an error and exit.
2. Source the script to parse all the input arguments given by user i.e. parse Plugin path (-plugin), library under test (-testLib), run directory(-runDir), -checkname, technology name (-testtechno), setup Only Option. If Usage of arguments is incorrect, show sample correct usage.
3. Set inputDir, outputDir, ipscreen_data, where
   inputDir, is path to library repository.
   outputDir, is path to generated run areas for each library.
   ipscreen_data, path to configuration required at run time by ipscreen.

4. If **testLib** is present in any of the library present in any techno repository, forcefully remove existing **outputDir**, and create new outputDir.

5. If there exists tool list for the testLib in inputDir, copy the tool list to the new outputDir.

6. For each plugin path given by user,

   (a) Extract the plugin name and version from the plugin product table present inside plugin. Remove existing symbolic link for the plugin and then create a symbolic link "pluginname@version" in run area to the plugin path given by user.

   (b) Create plugin list file in this run area, write plugin name, version entries in plugin list.

7. Create a script (run1.csh) in the run area, having set pluginlist= {pluginName {} {} {checkname} *} . This is an argument to automated command file creation module as seen in Section 5.3. This script will be executed at runtime and will invoke automated command creation module. This script works as a wrapper for automated command file creation module. The wrapper is in csh as the default SHELL is csh.

8. Append in the script, run1.csh, library names and supporting library names, from the configFile maintained in **inputDir/testLib**. (example lib= A; support Lib = B C D).

9. Copy the IPscreen preferences from ipscreen_data, environment variables from the ipscreen_data that are required by the run time to launch jobs on LSF, project name, wait time for licenses. Check that all these required files are made available in the run area. Create symbolic Link for the IPScreen in this run area.

10. Grant execution permission to run1.csh in the work area.

11. Invoke parallel job execution module.

Below figure shows an example of data maintained in the library repository(datastore), that will be used at run time to build a wrapper script, and which will further invoke automated command creation module.

Figure 5.5: Example of a single main library and its supporting libraries maintained in library datastore.

5.4.3 Automated Log Analysis Module

After all the checks are executed on the IPs, it is the responsibility of the developer to check the logs for any issue or unexpected behavior. Till now, the QA was done by using xxdiff utility, which is provided under GNU GPL open license. It is a graphical, file and directory comparator. Using this utility, the new logs and primary logs generated at the time of QA during previous release are compared manually for each plugin, IP pair. If corresponding logs of both the QA sessions were same then it was considered as correct. However, if the new log showed difference in content, then the new secondary level logs are to be manually analyzed separately. For each task a single primary log is generated. The secondary log is the main log and its size depends on the functionality of task and the number of cells on which the check executes. The size of these logs varies from 1000 lines to 10,000 lines. The primary report log is made after extracting the desired information from secondary log. If the check uses some CAD tool then the secondary report is prepared from extracting required tool outputs from tool generated log and finally the Check writes only the useful information in Primary report log.

Figure 5.3 shows the technique in which the log are created and compared for QA. The

figure shows comparison of logs generated by two different versions of the same plugin (PLUGIN_1_OLD,PLUGIN_1_NEW) on validating the same IP (Library1).

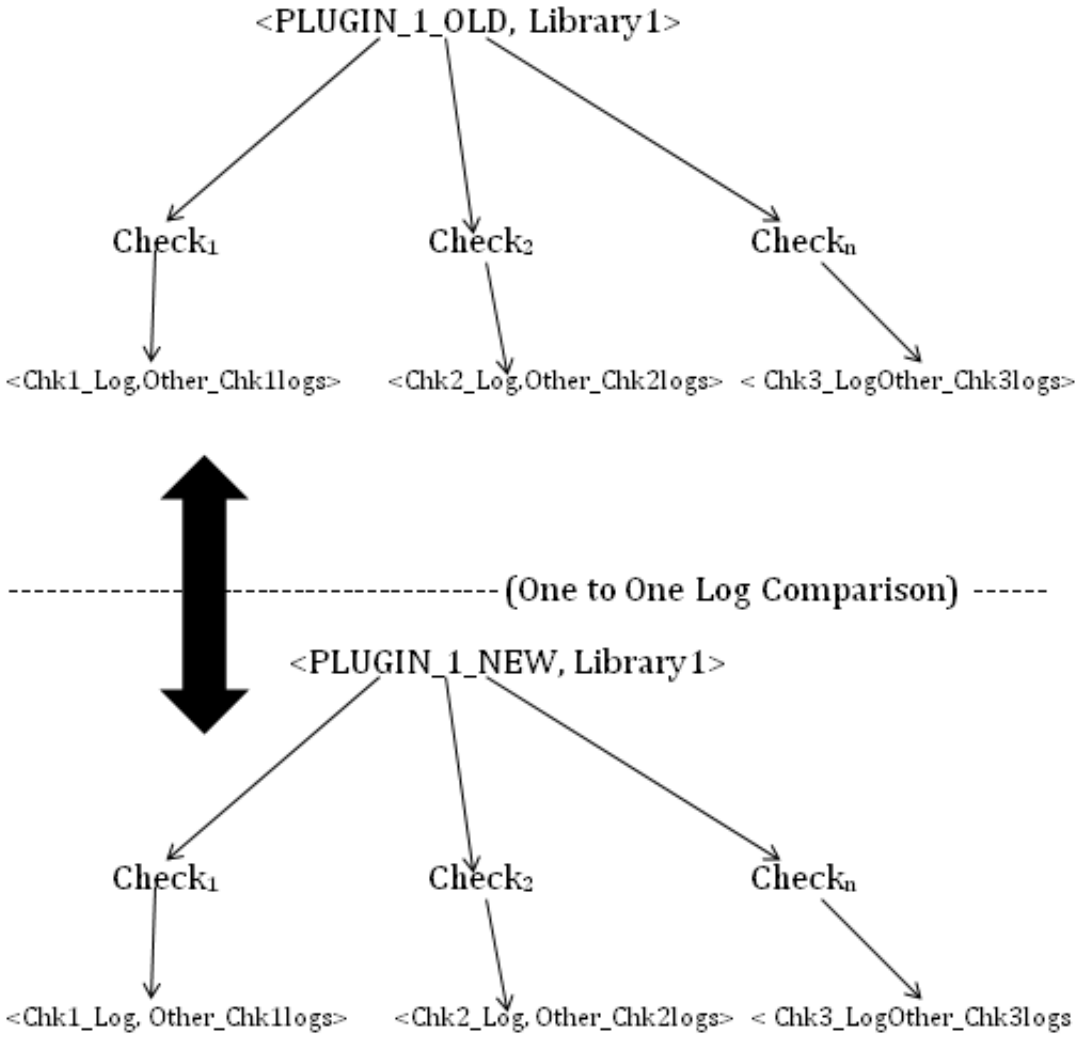

Figure 5.6: Basic Principle of Log Analysis

Depending upon the size of log it was tedious process and time consuming. With the new automated approach the automation, compares the in depth logs of new QA with logs of previous QA done at the time of previous release. The intent of this module was to reduce the logs analysis time. The output of this Automated Log Analysis module is only the log messages that represent the new functionality of plugin. It has been done by ignoring the log messages that are same in both the QA sessions and also the messages that are to be ignored. We have found patterns and formed regular expressions for them. The patterns were formed for messages that could be ignored as follows :

- Job submission message specifying the host ID on which job is submitted.
- Submission and Finishing of execution of Job on L.S.F. showing the submission time and finish time along with date, log generation time.
- Pattern for check in (showing submission time) and check out (showing finish time) of CAD tools during execution of tasks,CAD License information.

All the remaining log messages of primary logs had information of the shell scripts commands statements when they are executed and when input is read, so these messages showed no difference on comparison with the old QA logs. The final output of automated log analysis was only the log messages corresponding to new functionality coded in the new plugins. So, all the messages that were same in both the QA regressions were ignored, and only the new messages were analyzed manually.

Log Analysis : Undetected errors.

In order to perform fool proof analysis of logs and avoid any false positives in the report, patterns of behavior shown by TCL commands on wrong input or on failing have been identified. Generally on failure of specific command the TCL script do not abort and exits, but continues its normal execution. Few examples of such incorrect execution of commands are:

- Usage of a undefined variable in command like regexp, regsub, puts, etc. The error thrown by TCL interpreter is UNDEFINED VARIABLE.
- Pattern for array out of index error.
- Pattern for unavailability of CAD tool licenses.
- Pattern for indefinite loop due to non availability of license.

With the help of regular expression on above messages, after all the regression are run, we run this module to detect any such undetected errors.

5.5 Automated Launching of Parallel IPScreen Jobs

After all input is prepared, in order to launch the validation of IPs on Load Sharing Facility, the command used is

"Change Directory to work area where wrapper script is present. Run wrapper script, which further invokes the command file creation module."
This is to be done for each library specific run area prepared by automated setup creation.This task of manually launching the wrapper script has been automated and multiple IPs can be validated at same time. We have implemented two techniques to launch jobs, as explained below

- In the first approach we launched a certain number of predefined jobs in background using **"eval command & "**. & allows process to be executed in background and allow the current shell to be free for further inputs. Trap has been defined on signals like Ctrl+C, user issues a quit signal Ctrl+D. These traps will kill all the running jobs. To further kill a specific ipscreen window, user may use the bkill command. I have used "wait PIDLIST" command in order to wait for the executing jobs to finish, where PIDLIST is the list of all the processes that are currently running. The disadvantage of this approach was that it may lead to starvation for jobs yet to be submitted as there is "wait" until all the old jobs are finished.
  Below are the steps to launch the multiple Jobs parallely on LSF, JobFile contains invocation of each of the wrapper script for each library. I have set the maximum parallel jobs as 5, as it becomes difficult to manage more number of IPScreen windows.

  1. Set job_count as 1. Set max_count as 5.
  2. While job_count is less than alljobs in Jobfile
     (a) Extract command(job_count) from jobfile.
     (b) Increment job_count.
     (c) Launch the command(count) .
     (d) Store the process id of the launched process in process_list.
     (e) Define trap to kill all jobs in process_list on inputs like Ctrl+C, Ctrl+Z.
     (f) If job_count is greater than max_count, Wait for all jobs submitted to finish.

  This method of launching parallel jobs was later not used as we had to wait for all jobs to get finished, ie a new job could be launched only if all the previous five jobs were completed.

- In this implemented approach, I have used gmake utility that allows us to automatically send new job to LSF if an submitted job is finished. The -j option of make utility tells how many jobs to execute parallely, by default it is one. Option -f of make utility, specifies the name of the make file to execute having all the rules. The advantage of this method is that it fully utilizes the LSF resources, as whenever a job is completed the gmake utility automatically sends another job on execution. So, this approach avoids starvation for waiting jobs.

  The command used is **gmake -f processedJobFile -k -j count_paralleljobs**, where

  processedJobFile is the make file where each line has rules to launch wrapper script for each different run area. This processedJobFile is prepared after processing all the wrapper script.

  count_paralleljobs is the decided number of parallel jobs sent.

5.6 Difference in Quality Assurance of Plugins done manually vs Automated PluginQAKit

The table below shows the basic difference in traditional QA method versus new Automated PluginQA Approach (see next page).

Table 5.1: Differences in Validation using manual approach vs PluginQAKit

Point	**OLD Manual Approach to launch plugins using GUI.**	**Automated Approach to launch plugins using PluginQAKit.**
Specifying Library, auxillary library	Manually find the index file of each library, its methodology and its reference libraries and write the commands in command file.	The library's central location is found out, and then its name, Methodology, reference library etc. are extracted and are written in input command files.
Preparation of plugin record	File having plugin and its path was manually prepared.	The path is provided as an argument and the plugin file is automatically generated on run time.
Launching of Job on LSF	The validation job was manually launched on the LSF.	By default launches the job on compute farm and also provides user an option to bypass the launching of validation and just prepare the input.
Selection of tools	Each time to validate a library all the tools required had to be written in a separate file .The correct tools depending on the techno of library had to be taken care of.	Now the correct tools are automatically picked up depending on the techno of library.
Creation of Work space	For each library validation a different work space were manually created having command file, plugin record, tools list.	Now, for n number of libraries of same Techno n workspaces are created by PluginQAKit, in an automated manner in a flat file structure.
Report Generation	Manually generate after all tasks complete using user interface.	Automatically Generated after tasks completion.
Scope of Human Error	High	Low

5.7 Time Difference achieved after automating Input Preparation

The table below shows the time saved by automating the task of input preparation.

Table 5.2: Time saved in input Preparation for validation

Number of Libraries(of same technology)	**Manual Approach of Input Preparation**	**Using PluginQAKit**
One Library	15-20 minutes	2 -3 minutes
Two Library	25-40 minutes	2 -3 minutes
Five Library	1.5 hour - 2 hours	5-7 minutes

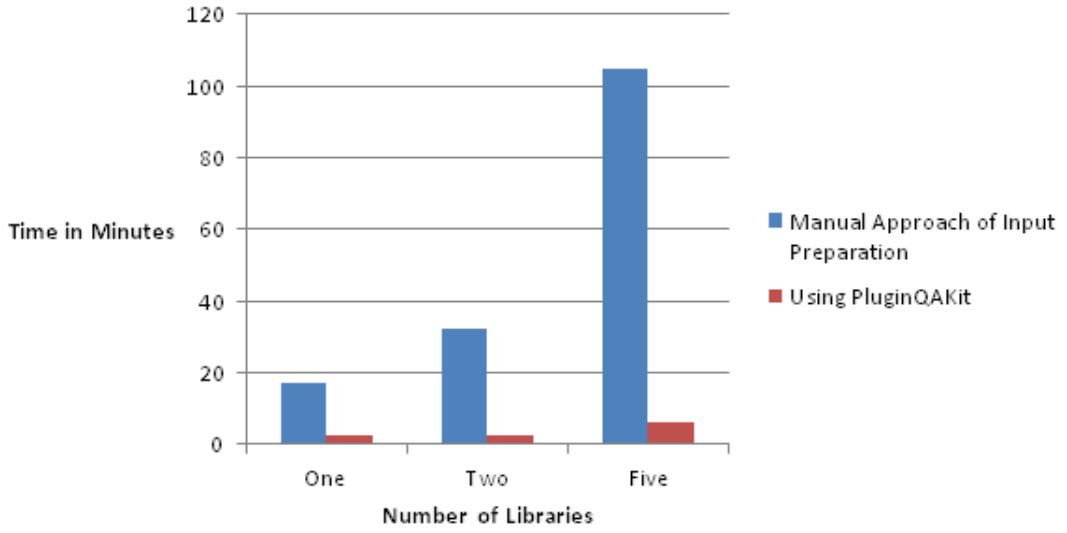

Figure 5.7: Time Difference achieved after automating Input Preparation

5.8 Time Comparison GUI mode vs PluginQAKit

The table below shows the time saved by for running n number of tasks manually versus pluginQAKit.

Table 5.3: Time saved for running n number of Tasks.

Number of Libraries	Average Number of Tasks per Plugin	Manual GUI mode of Validation	Using PluginQAKit
One Library	10	1-1.5 hours	2-3 minutes
One Library	20	2-2.5 hours	2-3 minutes
Two Library	10	2-2.5 hours	2-5 minutes
Two Library	20	3-4 hours	2-5 minutes

5.9 Time Difference Achieved by Automating in primary level Log Analysis

The table below shows the time saved in analysis of all the output logs during QA of plugins.

Table 5.4: Time saved using PluginQAKit during Log Analysis.

Number of Libraries	Number of Tasks per Plugin	Manual Approach of Log Analysis	Using PluginQAKit
One Library	10	50 minutes	10 minutes
One Library	20	100 minutes	20 minutes

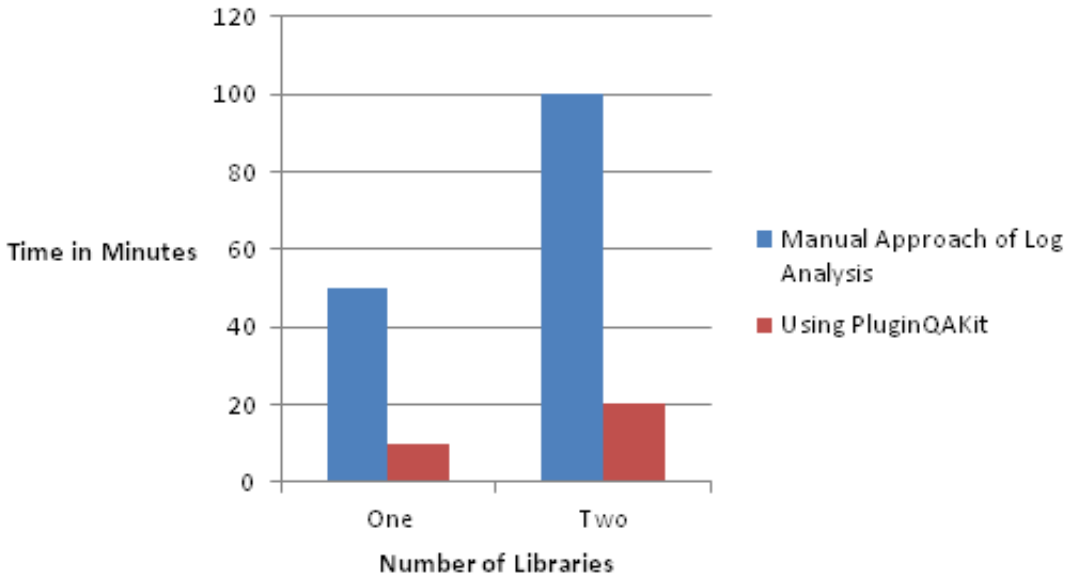

Figure 5.8: Time Difference achieved by automating primary level log analysis

Chapter 6

Conclusion

6.1 Conclusion

Plugins play a crucial role in validating an IP. With the development of PluginQAKit, the time required for regressive QA of the plugins before the release has drastically been reduced. Prior to development of PluginQAKit, it was difficult and tedious task to manually launch QA sessions for each IP and check all the logs of Q.A. sessions to check for any failure. With the use of PluginQAKit, the time to prepare and launch Q.A. sessions has been drastically reduced also it has eased the task of cross checking the behavior of plugins. Scope of human error and issue of incompatible tool, missing tools has been solved using Automated PluginQAKit.

References

[1] STMicroElectronics Internal document on Library.

[2] STMicroElectronics Internal document on Views.

[3] TCL/Tk CookBook by L. Sastry

[4] GNU GMake Manual by Richard M. Stallman, Roland McGrath, Paul D. Smith.

[5] STMicroElectronics Internal document on IPScreen.

[6] STMicroElectronics Internal document on Plugins.

[7] STMicroElectronics Internal document on TagsEvaluator.