Memory Flow Generation and Validation

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

Submitted By

Kothari Aman Dilip

14MCEC12

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2016

Memory Flow Generation and Validation

Major Project

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering

Submitted By

Kothari Aman Dilip

(14MCEC12)

Guided By

Mr. Vivek Garg	**Prof. Jitali Patel**
External Guide	**Internal Guide**

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 **"Memory Flow Generation and Validation"** submitted by **Kothari Aman Dilip (Roll No: 14MCEC12)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering of 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 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. Jitali Patel
Guide & Assistant Professor,
CSE Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Priyanka Sharma
Professor,
Coordinator M.Tech - CSE
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 ***"Memory Flow Generation and Validation"*** submitted by **Kothari Aman Dilip (Roll No: 14MCEC12)**, towards the partial fulfillment of requirements for the award of degree of Master of Technology in Computer Science and Engineering (CSE) of 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 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.

Mr. Vivek Garg
Senior Software Engineer & External Guide,
TRnD Department,
STMicroelectronics Pvt. Ltd.

Mr. Ashu Talwar
Project Manager,
TRnD Department
STMicroelectronics Pvt. Ltd.

Statement of Originality

I, **Kothari Aman Dilip**, Roll. No. **14MCEC12**, give undertaking that the Major Project entitled "**Memory Flow Generation and Validation**" 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.

Signature of Student

Date:

Place:

Endorsed by

Prof. Jitali Patel

Acknowledgements

First of all, I would like to wholeheartedly thank **Mr. Vivek Garg**, my mentor and external guide, for his constant guidance and support. The experiences and knowledge shared by him motivated me a lot to learn.

I would like to express my profound gratitude to **Mr. Ashu Talwar**, Section Manager, who had regularly reviewed my work and encouraged me to perform better. His valuable suggestions had helped me to improve.

It gives me immense pleasure in expressing gratitude to **Prof. Jitali Patel**, my internal guide and Assistant Professor, Computer Science Department, Institute of Technology, Nirma University, Ahmedabad for her valuable guidance and continual encouragement throughout this work.

I am highly grateful to **Mr. Rajamohan Varambally**, Director, TR&D department, STMicroelectronics and **Dr. Sanjay Garg**, Head of the Department, Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad and Dr. Priyanka Sharma, PG Coordinator, M.Tech CSE for allowing me to carry out my dissertation work at STMicroelectronics and gain industrial experience as well.

I would also thank the members of review panel and all the teachers who had given their valuable suggestions and guided me to improve.

A special thanks to all my friends especially **Kunal, Ajinkya, Sandip, Manthan**, who were also my roommates, for keeping me in good mood and supporting me.

I dedicate this work to my parents. Their blessings always gave me inspiration and strength to carry out this work and complete it.

- Kothari Aman Dilip
14MCEC12

Abstract

SoC chips are designed using IC design cycle. And Memory Layout design is an important step in the design cycle. It aims to generate various representations of memory cells, called views, which are used by different vendor tools for designing memory. Earlier, it took around 10-15 months to generate memory layout with given specifications and technology. To reduce this time, memory generators were developed which can easily generate layout for a given technology with different specifications. To generate any layout, the generators need corresponding package of products and the layout thus generated needs to be validated. Further, layout validation is a resource-intensive process and could not be carried out on local machines and thus require to be executed on cluster of computers with availability of load balancing service. This dissertation report focuses on automation of product compilation into package, monitoring generations launched on cluster computers and generating job execution statistics to analyze their performance. A scalable approach has been proposed for product compilation which overcomes the limitations of previous approach. New scripts have been written for monitoring generations verified with sufficient test cases. Scripts for generating job statistics have also been written and put to implementation.

Abbreviations

BE	Back End
FE	Front End
IC	Integrated Cicuit
CAD	Computer Aided Design
EDA	Electronic Design Automation
CDL	Circuit Description Language
GDS	Graphic Database System
RTL	Register Transfer Level
LSF	Load Sharing Facility
RAM	Random Access Memory
ROM	Read Only Memory
RTM	Real Time Monitoring
SoC	System on Chip
SGE	Sun Grid Engine

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

System-on-Chips(SoCs) are widely used today in mobile phones, modems, DVD players, television and many more consumer electronic products. It is generally an integrated circuit(IC) which includes all the components of a computer like microprocessor, memory blocks including peripherals, external interfaces like USB, Ethernet, power management circuits.

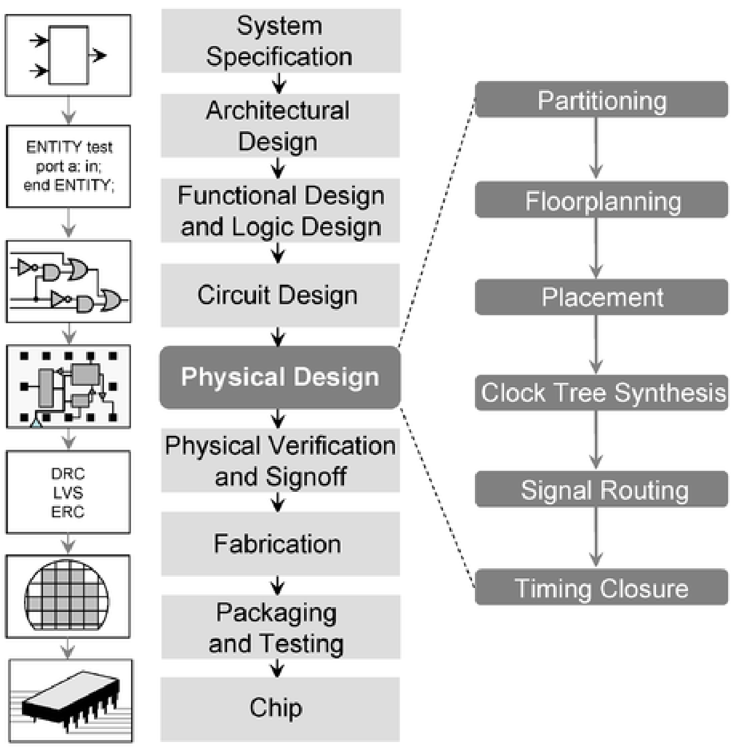


Figure 1.1: Memory Design Cycle[1]

For designing SoCs, the typical IC design cycle is to be followed[1]. The design cycle is shown in figure 1.1. We would focus on Physical Design Step. This step converts circuit description (schematics) into geometrical representations(layout). The physical design step is further split into: floorplanning, routing and placement, design optimization and validation.

We are concerned with designing memories for SoC. Memories are classified as primary memory like RAM and ROM, based on technology like 65nm, 45 nm, 32 nm, etc.

1.2 Motivation

The software team develops and delivers physical layout of memory as products. A product is a set of documentation, files to be used with commercial tools, or tools written in various programmable languages. Generally, the product specifications are stated by a client according to its requirements.With the manual approach, it took around 10-15 months to develop a complete product with given specifications. So, if a product has been designed with one set of parameters and a client comes with different set of parameters, the whole design process has to be repeated. This would again take around 10-15 months. In this way, the throughput using manual approach was less in terms of time compared to client needs. Thus, an approach to automate the physical design process was needed which could reduce overall product development time drastically.

A new design approach has been followed since then that is to develop memory generators. Memory generator is a set of scripts which when compiled and run would generate memory layout with configurations specified by user. This means generators are dedicated to memory. So, we save time on generating different configurations by developing generators for different technologies and kinds of memory. Having developed generators, we are further required to speed up the process of generating views and validate them. [2]

1.3 Objective

The objective of this project is to optimize the process of product generation in terms of execution time and scalability, support product generation infrastructure and implement job monitoring tasks.

[1]Courtesy:en.wikipedia.org/wiki/Physical_design_(electronics)

1.4 Scope of Work

Different BE views are generated using dedicated memory generator which make use of input specification files and EDA products provided by vendors. The views once developed are validated by repetitive cut generations(configuration), where different parameters are passed to them. These generations requires substantial resources in the form of processing power and storage, so they are submitted to a computing facility in the form of jobs. Further, the jobs need to be monitored and appropriate actions are to be initiated based on their status.

1.5 Project Work Flow

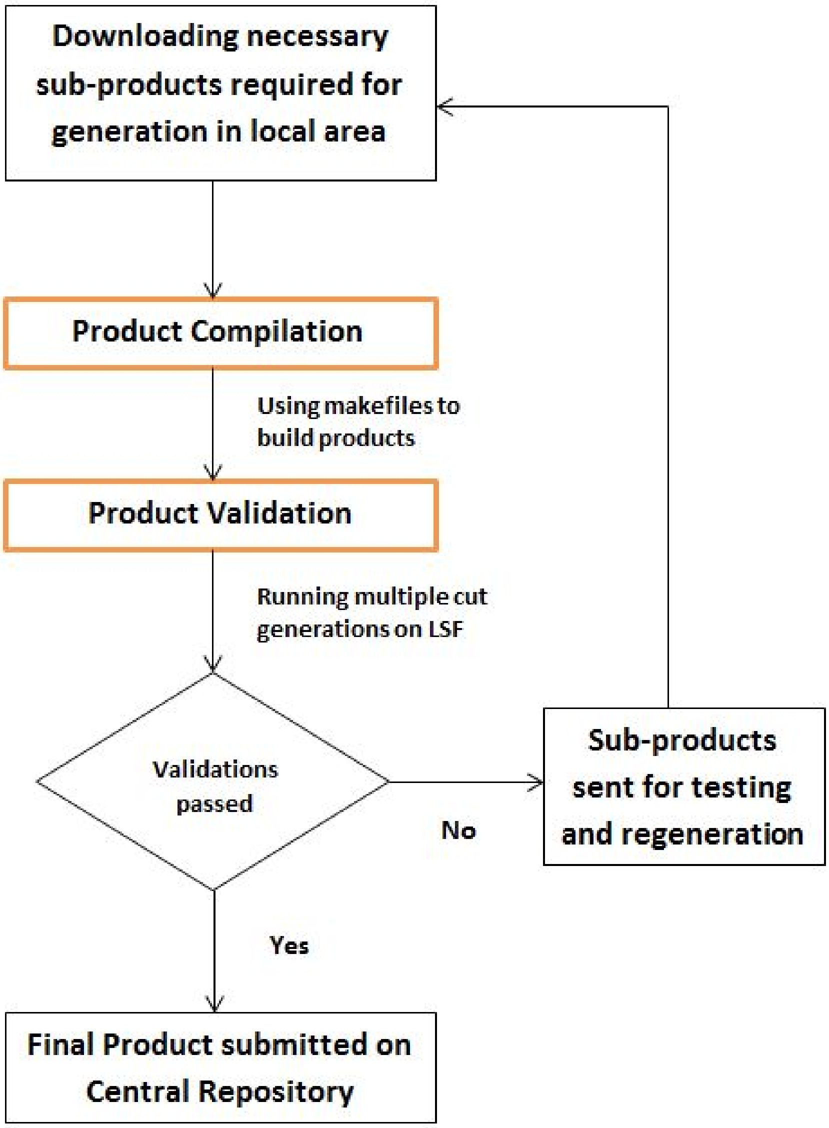


Figure 1.2: Project Workflow

1.6 Tools and Technology

- Operating System
  - RedHat Linux
- Languages
  - TCL - TCL stands for Tool Command Language. It is a fast, dynamic, powerful and easy to learn scripting language. It is capable of doing variety of file handling and text processing tasks. It is useful in web applications and desktop applications, simulation softwares, networking, testing and in many other areas. It is also open source, cross platform, easily deployable and highly extensible language.
  - Bash - Bash stands for Bourne-again shell. Bash, as the name suggests, is a replacement of shell(sh). Bash scripts usually run in Unix (or Linux) terminals and carry out utility tasks.
- Tools
  - VNC - VNC stands for Virtual Network Computer. Basically, VNC is desktop sharing system. It is based on client server model. Therefore, many clients can connect to main server simultaneously. VNC is widely used to provide technical support remotely or when someone has to access some files at different locations.

1.7 Thesis Organization

In chapter 1, an introduction is given about the project domain and motivation behind the problem definition is discussed. We also state the objective of the project and the scope of work along with general work-flow of the project.

In chapter 2, we have shown the literature survey done to gain knowledge about various tools and setups used and concepts related to the project.

In chapter 3, we discuss product compilation and the limitations of its existing approach. We also propose a new scalable approach for the same.

In chapter 4, we see the cut generation process and discuss about launching generations on cluster computers. We see in detail the purpose of all the scripts written for monitoring jobs.

In chapter 5, we see the accounting mechanism of two well known distributed resource manager - Sun Grid Engine and Load Sharing Facility. And we also discuss the approach for generating job statistics for both of these facilities.

In chapter 6, we investigate a new approach which is intended to replace existing approach of Steps Compilation.

In chapter 7, we conclude about the work done and also discuss what would be the future scope of work.

Chapter 2

Literature Survey

2.1 Standard Cell Libraries

Cells play an important role in chip designing. Cells are basic components which perform logical functions like AND, OR, NOT, NAND, NOR, XOR. The collection of such basic cells is called **Standard Cell Library**. A library may also contain complex functions like Full-Adder, Comparator, Multiplexer,etc [3]. They are required by most of the CAD tools for designing chips. The main purpose of such tools is to implement RTL-to-GDS flow.

- RTL - The input to physical design in the form of circuit description language.
- GDS - The final output from physical design process is full chip layout and it is in GDS2 format.

To produce a design which is functionally correct and meets all specifications, a combination of CAD tools is required in design flow. These tools require some specific information but in different formats for each of the cell in library provided for designing. Such different formats are made available in the form of *Views*.

A view is a particular representation of a cell. A standard cell is delivered as a set of views. The classification of views is shown in figure 2.1. Of these, the BE views are related to the physical design of a cell and in that we are concerned with the layout views. The layout views are represented in GDS2 format.

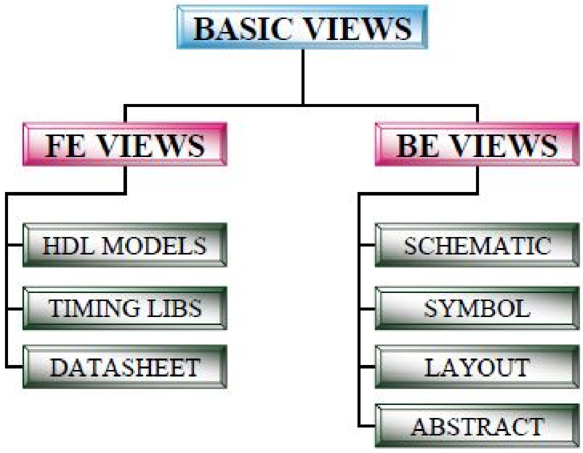


Figure 2.1: Classification of Views

In figure 2.2, we see a sample library structure. It is shown that how a set of views constitute a cell.

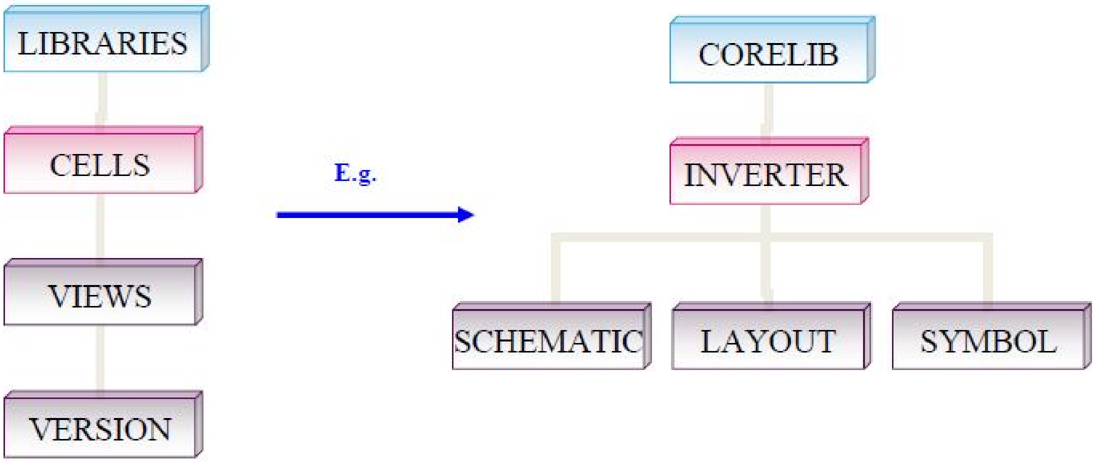


Figure 2.2: A Sample Library Structure

2.2 Semi-Custom Design Approach

This approach is preferred for automating the design process. The motive behind following this approach is to reduce the design cost as well as design time by reusing standard cells. If a cell is required repetitively, then rather than creating cell manually each time, the logic of previous cell instance is reused. The only limitation here is reduced scope of optimization as designer loses control of layout.It is rather advisable to manually create our own optimized cells and further use them in a semi-custom layout manner. [3][4]

2.3 Compute Farm and LSF

Compute Farm is a network of distributed servers which delivers high performance computing required for CAD design or software development.

A load balancing application by IBM, called Load Sharing Facility (Platform LSF), aggregates many servers to optimize the run time of users's jobs. LSF provides transparent access to all available resources, monitors activity, controls access and distributes the job workload for optimal performances.[5]

The setup is such that multiple LSF clusters are used. To determine the current LSF version number and cluster name, following command is used on UNIX terminal:

lsid

To see configuration information about the local cluster like number of LSF server hosts, administrator's account name, status of cluster , following command is used:

lsclusters

Jobs submitted to a LSF cluster are enqueued in 5 standard queues for depending on job patterns.

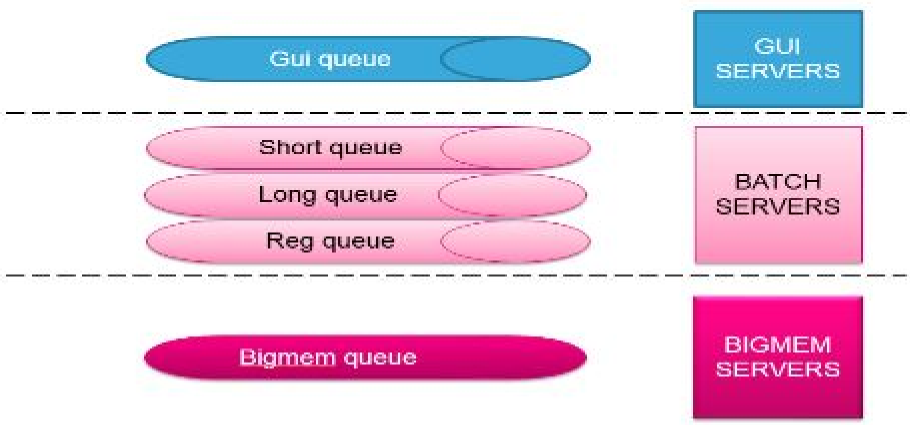


Figure 2.3: Queues on a LSF cluster

The queues are categorized as interactive queue, batch queues and bigmem queue and each have separated pool of servers. The standard queues are shown in figure 2.3.

- GUI queue - They are meant for interactive jobs that are neither CPU intensive nor memory intensive.
- Short queues - They are meant for jobs which consume less than 30 minutes of CPU time.
- Reg queues - They are meant for parallel jobs.
- Long queues - They are meant for jobs which consume more than 30 minutes of CPU time and not many of them are parallel.
- Bigmem queues - They are meant for jobs which require more than 16GB of memory.

All the queues provided by an LSF cluster can be viewed by running the following command on UNIX terminal:

bqueue

This command displays information like queue name, queue priority, queue status, and statistics related to jobs state.

A similar distributed resource manager is Sun Grid Engine. It has following features[6]:

- Scalability - It is highly scalable. There are customers using SGE with thousands of machines processing millions of jobs per month.
- Flexibility - It is customizable and fits to customer's needs.
- Reliability - It requires minimal maintenance effort and there are less chances of failures.
- Advanced scheduler - It provides variety of scheduling policies for fine-tuning job distribution. Using these policies, an organization could configure SGE to make its scheduling decisions match their business rules.

Few relevant use cases of these distributed computing facilities is as follows.

- Many EDA software vendors use these facilities to launch and manage large number of regression tests. The tests are submitted as thousands of jobs to be run on cluster. As soon as a test is completed on any of the machines, remaining tests are launched on it. This way the machines are kept busy until all the tests are completed.

- These facilities are also used to manage software licenses. During software simulations or tests, it is required to acquire license to use some external tools. The need for licenses could be reflected through job submission and there will be assurance that no more licenses are used than are available.

2.4 make and Makefile

When a large project is being developed with many of its modules dependent on each other, compilation becomes time consuming. Even if slight changes are done in a module, whole project needs to be recompiled. This causes the development of the project to slowdown. To avoid such an issue, a UNIX utility command 'make' is used. The make utility automatically determines which modules of a large project need to be recompiled, and issue the commands to recompile them.[5]

make searches for the information about dependency among modules. This information is stored in a file called Makefile. Thus, each time when changes are done to one or more module and project needs to be compiled, make would search Makefile first, extract the dependencies and only compile those files which are affected by the change. This helps in saving time for unnecessary compilation during development and debugging of modules. Structure of Makefile is as follows:

Target : Dependencies ...

Commands

....

....

- **Target** is the dependent file which needs to be created if there is any change in dependency or if it is an older file than the dependency. A target may have more than one dependency.
- **Dependency** It is a file that used as input to create the target.
- **Commands** Set of actions carried out by make

Chapter 3

Product Compilation

In section 2.4, we discussed about the make utility in UNIX and how it uses Makefile to compile large projects. We are using a similar utility, known as ***stMake***, which has some functionalities added over the traditional make command. stMake was created to provide ease-of-use to developers while dealing with memory designing projects. We will now see that how stMake is implemented and suggest improvements in its use.

3.1 stMake

stMake is used within UNIX systems and requires TCL 8.0 or higher versions. It is also mandatory that it finds command gmake. stMake is fully compatible with gmake.

To have specific stMake actions enabled the Makefile must be constructed according to an already defined template. stMake provides different set of functionalities depending on the type of actions to be performed. Each set of actions is called a required library and whichever libraries are strictly required must be mentioned in Makefile.

3.1.1 General Flow of stMake

Whenever we need to execute stMake command, we do not directly run the command on terminal. Rather we have made a wrapper file which initiates the stMake flow. The flow can be seen in 3.1.

- The wrapper file is named stMake.cmd. The reason behind using the wrapper is to provide abstraction layer. The user will run the command stMake.cmd on terminal. This which would invoke a call to stMake.Compile.

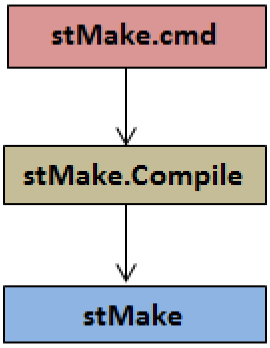


Figure 3.1: Flow depicting stMake call sequence

- In stMake.Compile, we check the path and existence of Makefiles and set all the required environment variables. Finally, stMake is invoked.

3.1.2 Existing approach for compiling products

Th directory structure is shown in figure 3.2. There are three main directories used during the entire compilation process:

1. source - It is the product source directory. It includes dedicated directories for all main products. All the subproducts corresponding to a main product are kept in dedicated directories, as shown in figure 3.3.

2. install - After compilation, the product is installed in this directory.

3. build - This directory stores log files and temporary files generated during compilation.

The approach here is to compile each and every subproduct manually to build the main product. That means user had to visit each subproduct directory and run the make command. It has following limitations:

1. Time consuming as it is a manual approach and makes it more difficult if any subproduct is added further.

2. The Makefile for same subproduct in different products were almost redundant. So, to make change in the common code of Makefile, all the Makefiles had to be modified. Thus, it did not support scalability.

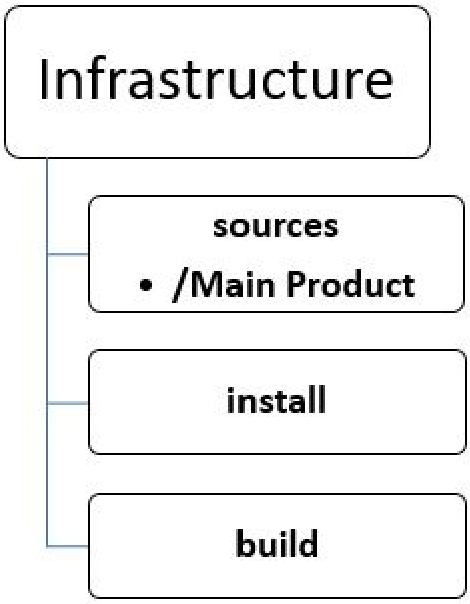


Figure 3.2: Existing Directory Structure

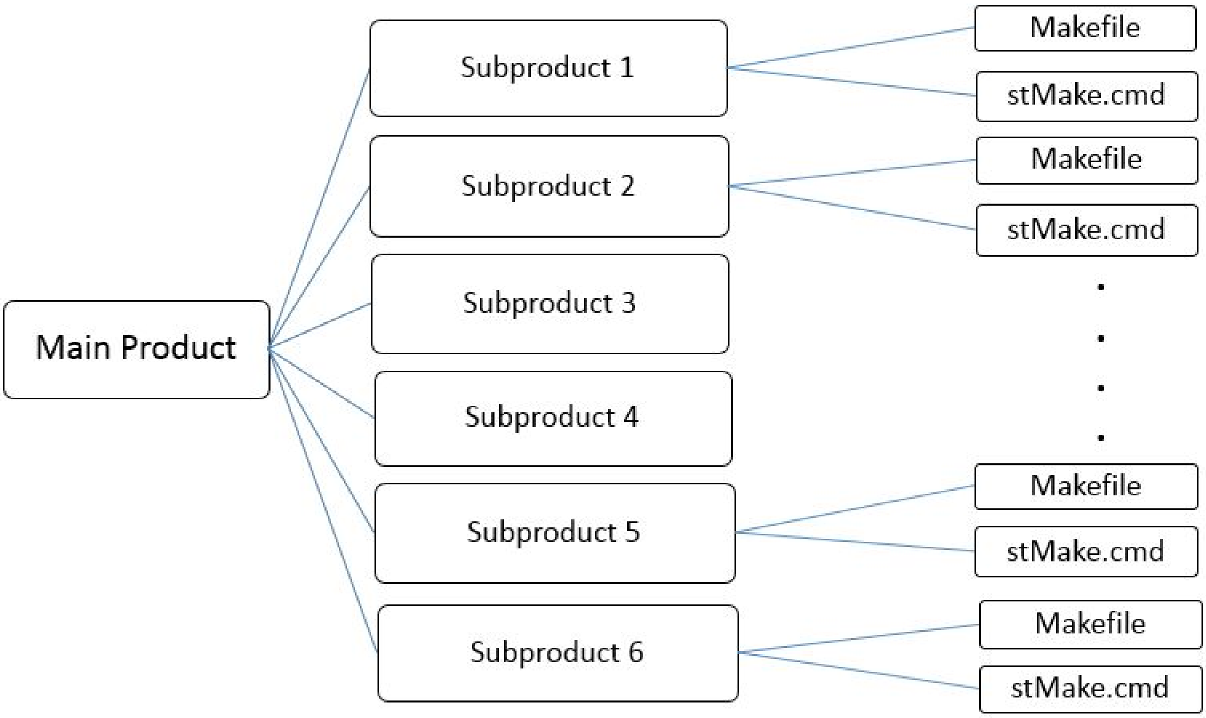


Figure 3.3: Product Directory View

3.1.3 Proposed approach for compiling products

In the proposed architecture, we aim to automate the compilation process and also remove redundancy of common code. The proposed directory structure is shown in figure 3.4.

A new directory named stCompilerTools has been added which will include all the makefiles for different subproducts, as shown in figure 3.5. This means that for any

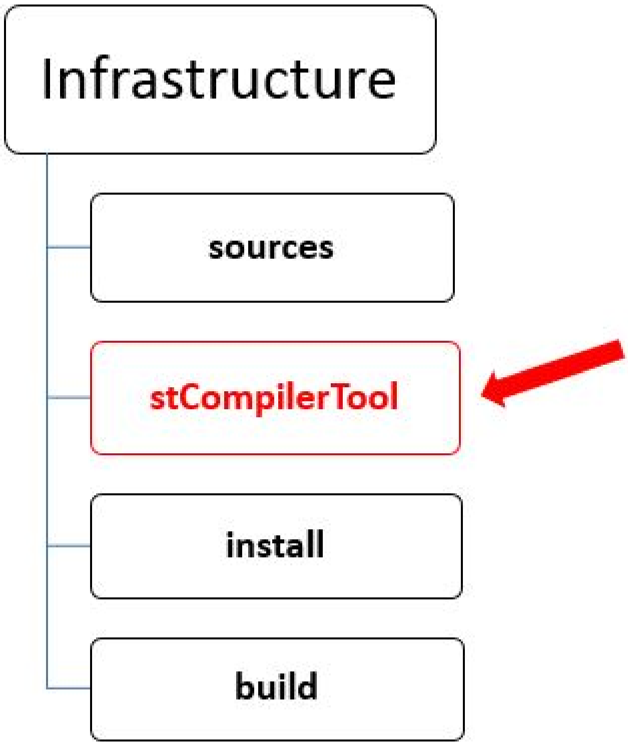


Figure 3.4: Proposed Directory Structure

number of products, there would only be one Makefile for their common subproducts.

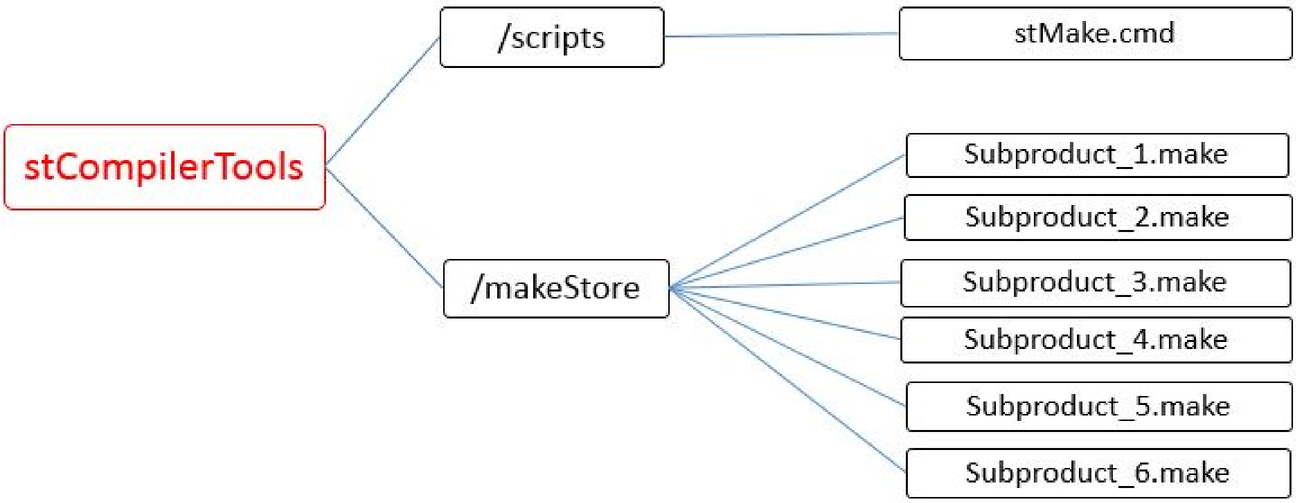


Figure 3.5: New Directory View

The advantage is that:

1. Now, the Makefiles for common subproducts would not be duplicated.

2. New Makefiles would be added only when new subproducts need to be compiled.

3. User does not need to visit each and every subproduct directory. A single command needs to be run with varying parameters for compilation.

4. Automatic generation of 'install' and 'build' area is also handled, in case they do not already exist.

The proposed automation approach fares better in terms of scalability and time consumption when compared to existing manual approach.

Chapter 4

Product Validation and Job Monitoring

After product compilation, next step is to validate the product. Validation checks are performed to check whether the product is properly generated i.e. no input files were missing, whether it can generate various configurations or not. This step is also called Cut Generation.

4.1 Cut Generation

The task of cut generation is carried out with help of memory generator. Another inbuilt product named ValidKit provides commands to initiate cut generation.

The detailed steps for initiating cut generation are as follows:

1. The latest configuration file of the generator is downloaded for which we want to run cut generation.
2. Generate prod and .prod files.
   - .prod file contains list of all the products needed by the generator
   - prod file contains detailed information such as full path of products and their version.
3. Create new area where the products mentioned in .prod would be sourced.
4. Source all the products required by the generator from common product repository.

5. Run command *createGenFile* to generate temporary files needed for cut generation.

6. Finally, run command *showGui* which would display GUI for specifying parameters and start cut generation as a background process.

The steps for cut generation with GUI are as follows:

1. On running the command showGui, the following window appears at first: As

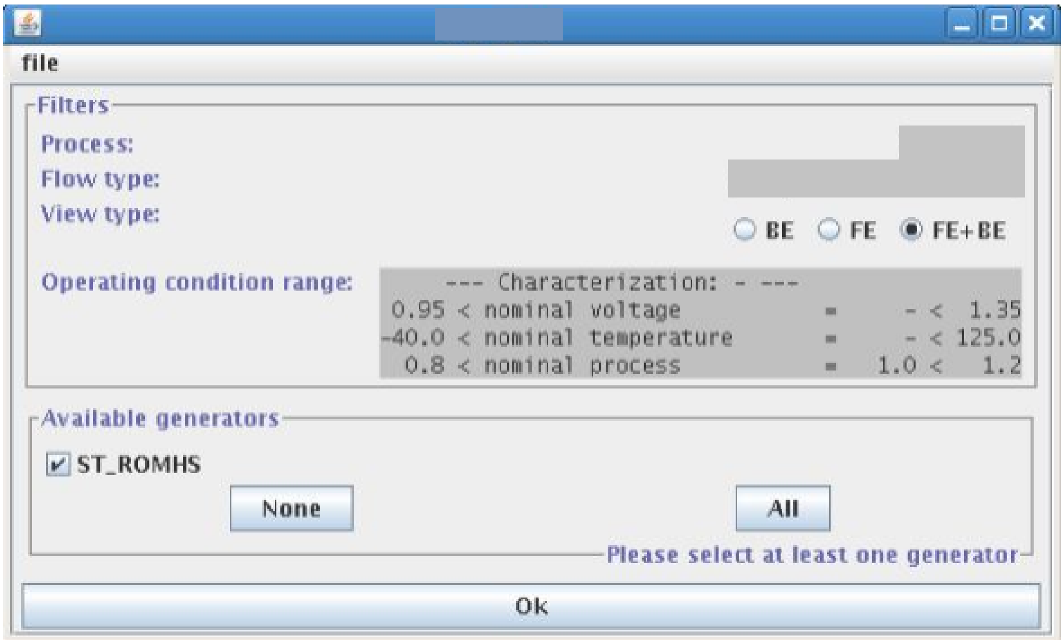


Figure 4.1: GUI Window

   shown in figure 4.1, we have the option of selecting which type of views have to be generated - Front End, Back End or both. The name of available generators are also mentioned, which will be used to generate the views.

2. After selecting type of views and generator, the following window appears, where we see options to select various parameters and feed their values.

3. In figure 4.3 and 4.4, we see parameter window for Flows Setup and Operating Conditions respectively.In Flows Setup, we provide path of the directory, where the library would be created, along with library name. While in Operating Conditions, we provide values for Temperature and Voltage. We also have the option of selecting default values and check where the values are valid or not.

Figure 4.2: GUI Editor Window

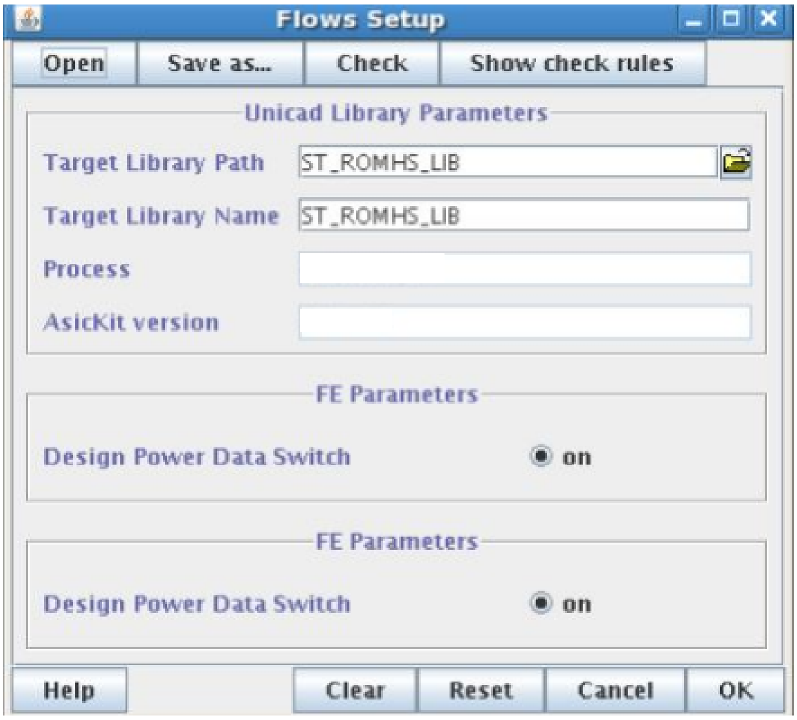


Figure 4.3: Flows Setup Window

4. In figure 4.5, we see Generator Parameters window. Here we specify Cut_name, words, bits, mux and other parameters. We also have the option of generating Cut_name and further compute other parameters for it.

5. In the last step, we select desired cuts and click OK to run generation in background, as shown in figure 4.6

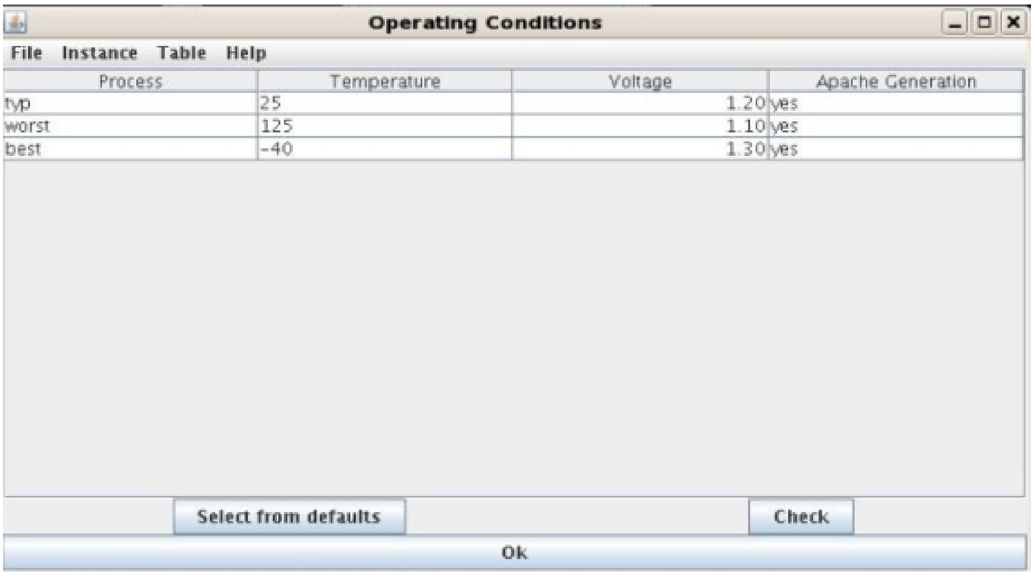


Figure 4.4: Operating Conditions Window

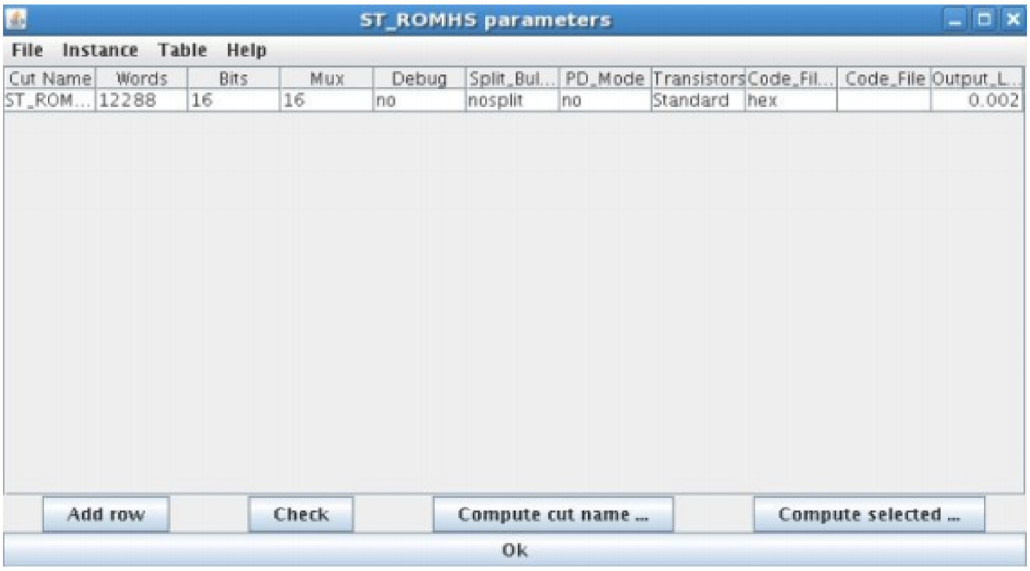


Figure 4.5: Generator Parameters Window

After completion of generation, we get a library containing all the views intended for product validation. All the views undergo validation checks like Design Rule Check and Layout versus Schematic checks. If they pass the checks, then the libraries are sent to the fabrication team for chip fabrication. But if any of the views are found incorrect, then the whole process of cut generation would repeat after debugging and fixing the bugs in the product.

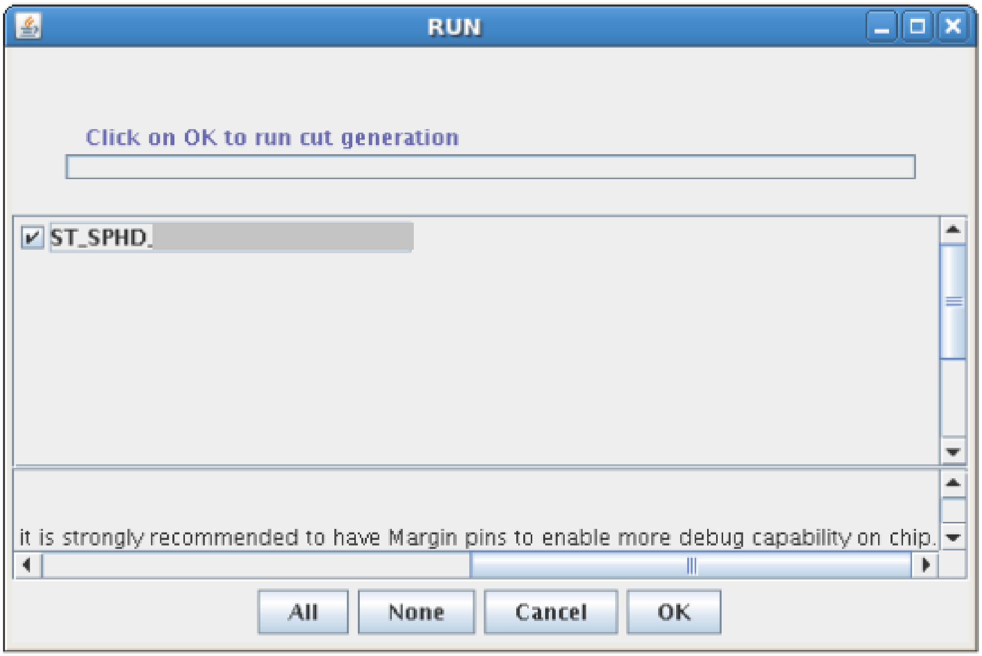


Figure 4.6: Cutgen Run Window

4.2 Job Monitoring

Cut generation is a resource intensive process. It requires enough processing power as various checks are performed. And when multiple cut generations are to be run simultaneously or otherwise, it is not feasible to run it on a single machine. Thus, cut generations are submitted as jobs on LSF. This process is also known as *launching generations*. As discussed in section 2.3, the jobs are en-queued in different queues depending on job pattern. Most of the generations are submitted as interactive jobs but they may not necessarily be en-queued in Interactive queue, depending on their requirement.

The following command is used to submit job on LSF in UNIX:

bsub

Here is an example of submitting a GUI job on LSF: *bsub -q long -p misc -I genGui*

- Option q denotes queue type, in this case it is long queue.
- Option p denotes name of project, in this case it is misc.
- Option I denotes that that it is an interactive job and takes the name of GUI application.

To see all the jobs submitted to LSF, following command is used:

bjobs

For each job it will display details like:

- JOBID - a unique id assigned by LSF to each job
- USER - name of user who submitted the job
- STAT - status of job like RUN, PEND, DONE etc.
- JOB_NAME - name of the job
- EXEC_HOST - host where job is executed
- EXEC_CWD - current working directory on EXEC_HOST
- SUBMIT_TIME - time of job submission

There are many more information headers provided by bjobs command which are useful in extracting details about jobs and use them as required.

We will now see the implementation of various scripts written for job monitoring and also performing tasks post cut generation.

4.3 Implementation

The following scripts are executed as terminal commands. Each of the scripts use bjobs command and extract details about the jobs:

1. *fetchWorkingDirectory*

   The options accepted by this script are:-

   - -area (directory) - the main directory where all the generations directories are kept
   - -run - seek directories for currently running jobs
   - -nrun - seek directories where no jobs are running
   - -workdir (directory) - seek specified generation directory
   - -trange (start date, end date) - seek generations from a specified time period
   - -failed - seek directories for failed generations

- -user (text/pattern) - seek directories for jobs launched by specified user
- -lib (text/pattern)- seek directories which contain specified library pattern
- -config (text/pattern) - seek directories which contain specified config pattern
- -echoDir - only display list of resultant directories

The task of this script is to fetch generation directories depending on the parameters passed. A generation directory contains all the information about the input specifications and cut generation. The resultant list of directories enable user to take further desired actions. A sample output for this script is shown in figure 4.7

```
### Project : TESTS GEN RON1:15741 , Library : test_all:136121,11,0 , requested by : user1 , is completed,
 => For directory : /work/amank/testing/generation/Gen_2015.10.17_10h48m13s
 => Genestatus:OK  Verifstatus:NOK .

### Project : TESTS GEN RON1:15741 , Library : test_all:136127,17,0 , requested by : user2 , is completed,
 => For directory : /work/amank/testing/generation/Gen_2015.11.03_11h14m48s
 => Genestatus:NOK Verifstatus:NOK.

### Project : TESTS GEN RON1:15741 , Library : test_all:136127,7,0 , requested by : user3 , is cancelled ,
 => For directory : /work/amank/testing/generation/Gen_2015.10.29_05h12m56s
 => Genestatus:OK  Verifstatus:NOK.

### Project : TESTS GEN RON1:15741 , Library : test_ash:136073,17,0 , requested by : user2 , is completed ,
 => For directory : /work/amank/testing/generation/Gen_2015.08.31_06h17m44s
 => Genestatus:NOK Verifstatus:NOK.

### Project : TESTS GEN RON1:15741 , Library : test_gen:136072,2,0 , requested by : user1 , is completed ,
 => For directory : /work/amank/testing/generation/Gen_2015.08.20_12h11m27s
 => Genestatus:NOK Verifstatus:NOK.

### Project : TEST GEN RON2:17954 , Library : test_lib:136102,4,0 , requested by : user3 , is cancelled ,
 => For directory : /work/amank/testing/generation/Gen_2015.11.18_06h30m52s
 => Genestatus:NOK Verifstatus:NOK.

### Project : TESTS GEN RON1:15741 , Library : test_ash:136071,23,0 , requested by : user2 , is completed ,
 => /work/amank/testing/generation/Gen_2015.08.10_09h47m14s
 => Genestatus:OK Verifstatus:NOK.
```

Figure 4.7: Output: fetchWorkingDirectory

2. *updateJobPriority*

   The options accepted by this script are:-

   - -priority (integer) - the new priority to be given to job
   - -workdir (directory) - update priority for jobs found in specified generations directory
   - -user - update priority for jobs launched by specified user

The task of this script is to update priority of pending jobs depending on the parameters passed. Higher number denotes higher the priority but maximum and minimum priority values are 100 and 1 respectively. Default priority is 50. Output is shown in figure 4.8.

```
{amank}414: updateJobPriority -priority 20 -workdir Gen_2015.12.08_10h28m45s/
### Info: Running on Gen_2015.12.08_10h28m45s/
### Info: Priority successfully updated for Job userLib (Pending Job) with Job ID 7210519
### Warning : Priority cannot be updated for Job cutStat (Running Job) with Job ID 7210504

### Info: Finished execution ....
```

Figure 4.8: Output: updateJobPriority

This script uses command *bmod*, available in LSF to change the priority. The command is run as follows: *bmod JOBID -sp new_priority_value*

e.g. *bmod 755891 -sp 51*

3. *findSubJobs*

   The options accepted by this script are:-

   - -area (directory) - find sub-jobs in the main directory
   - -workdir (directory) - find sub-jobs in the specified generation directory

   This script finds sub-jobs in the specified main directory. This script is also called by other scripts to use its functionality. A sample output for this script is shown in figure 4.9

```
{amank}416: findSubJobs -area /work/amank/testing/generation/


 JOBID    USER    STAT   JOB_NAME    EXEC_HOST    EXEC_CWD                         SUBMIT_TIME
 721134   user1    RUN   userlib      amank       ../Gen_2015.12.02_11h10m23s      Dec  2 11:10:20
 721135   user1    RUN   cutstat      amank       ../Gen_2015.12.02_12h20m08s      Dec  2 12:20:08
 721136   user1   PEND   vsubckt      amank       ../Gen_2015.12.02_12h15m26s      Dec  2 12:15:26
```

Figure 4.9: Output: findSubJobs

4. *findTankedJobs*

   The options accepted by this script are:-

   - -cutoffTimeMain (integer) - cut-off time for main jobs

- -cutoffTimeSub (integer) - cut-off time for sub-jobs
- -cutoffTimePend (integer) - cut-off time for pending jobs
- -orphan - find sub-jobs which do not have any parent job running

This script displays jobs which are taking more time than the specified cut-off time. In most of the cases, such jobs have to be killed. Output is shown in figure 4.10.

```
{amank}411: findTankedJobs -cutoffTimeMain 24
### Warning:  Job main (MainJob) with Job ID 7210111 should be terminated (24 hrs)
```

Figure 4.10: Output: findTankedJobs

5. *findLatestGeneration*

   The options accepted by this script are:-

   - -lt - enables search for latest generations
   - -workdir (directory) - search specified directory for generations

   This script checks and displays whether a generation is the latest one to be launched or is an older one. Output is shown in figure 4.11.

```
{amank}412: findLatestGeneration -workdir Gen_2015.12.04_13h44m23s/
### Info: /work/amank/testing/generation/Gen_2015.12.04_13h44m23s is older generation

{amank}413: findLatestGeneration -workdir Gen_2015.12.05_09h45m58s/
### Info: /work/amank/testing/generation/Gen_2015.12.05_09h45m58s is latest generation
```

Figure 4.11: Output: findLatestGeneration

6. *findJobsWithoutWorkDir*

   The options accepted by this script are:-

   - -area (directory) - work area where all the jobs are running
   - -kill - enable job killing

   This script finds those jobs whose current working directory has been deleted. If the user has passed option -kill, all such jobs would be killed.

7. *setObsolete*

   The options accepted by this script are:-

- -workdir (directory) - directory to be set as obsolete

This script allows to set the specified directory as obsolete so it that it does not appear in mailing list of failed generations.

8. *removeWorkingDir*

   The options accepted by this script are:-

   - -workdir (directory) - generation directory to be removed.

   This script checks whether in the specified directory any job is not running or it is not be kept for debugging or it is obsolete, and deletes them. A sample output for this script is shown in figure 4.12

```
{amank}422: removeWorkingDir -workdir ~/sample/generation/Gen_2015.08.10_09h47m14s
### Info: Running on /work/sample/generation/Gen_2015.08.10_09h47m14s
### Info: File .generationRunning is not found. Now,given directory will be deleted.
```

Figure 4.12: Output: removeWorkingDir

9. *dailyMailAlerts*

   The options accepted by this script are:-

   - -mail - enable mail transfer, if disabled, result is displayed on console.

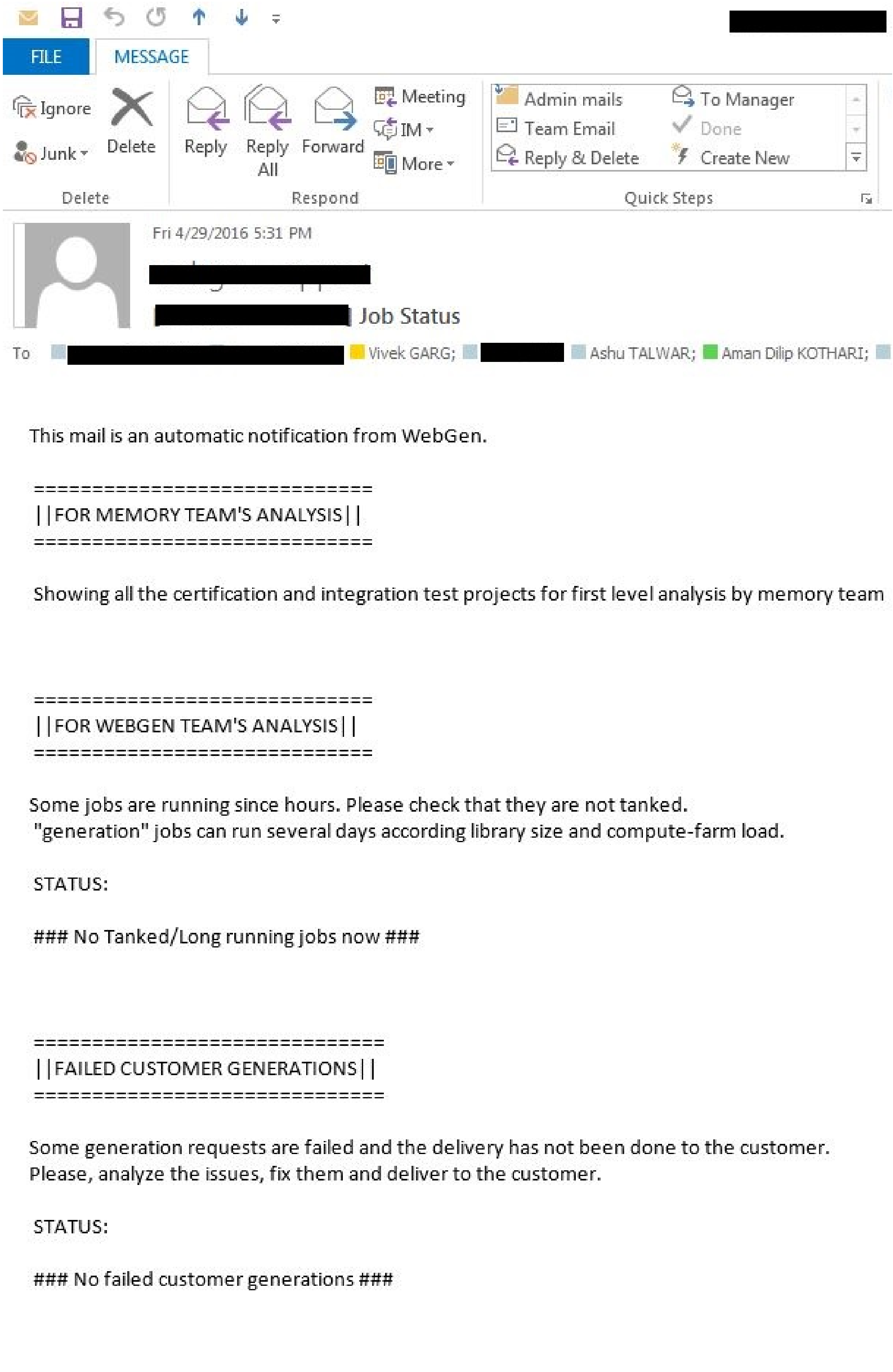
   Display tanked jobs and failed customer generations to stakeholders on a daily basis, enabling them to take appropriate actions. This scripts serves the purpose of daily reporting. It would run at specified time intervals, generating results and sending it over mail. Its scheduling is done using Crontab. A software utility Cron is a job scheduler available in Unix-based environments helpful for periodically running jobs. And Crontab (or Cron table) is a configuration file which specifies shell commands to run periodically.

   Internally, this script calls the following two scripts with filters:

   (a) findTankedJobs

   (b) fetchWorkingDirectory

   A sample mail is shown in figure 4.13.



Fri 4/29/2016 5:31 PM

[redacted]

[redacted] Job Status

To [redacted] Vivek GARG; [redacted] Ashu TALWAR; Aman Dilip KOTHARI;

This mail is an automatic notification from WebGen.

==============================

||FOR MEMORY TEAM'S ANALYSIS||

==============================

Showing all the certification and integration test projects for first level analysis by memory team

==============================

||FOR WEBGEN TEAM'S ANALYSIS||

==============================

Some jobs are running since hours. Please check that they are not tanked.
"generation" jobs can run several days according library size and compute-farm load.

STATUS:

No Tanked/Long running jobs now

===============================

||FAILED CUSTOMER GENERATIONS||

===============================

Some generation requests are failed and the delivery has not been done to the customer.
Please, analyze the issues, fix them and deliver to the customer.

STATUS:

No failed customer generations

Figure 4.13: Output: dailyMailAlerts

10. *findStatus*

    The options accepted by this script are:-

    - -workdir (directory) - generation directory whose status is to be found.

    This script finds the generation status of the input directory. The script is also called by *fetchWorkingDirectory*. A sample output is shown in figure 4.14

```
{amank}420: findStatus -workdir ~/amank/testing/generation/ Gen_2015.08.10_09h47m14s
### Info:   Genestatus : OK
            Packstatus : OK
```

Figure 4.14: Output: findStatus

11. *filterGenerations*

    The options accepted by this script are:-

    - -obs (yes/no) - filter obsolete generations
    - -can (yes/no) - filter cancelled generations
    - -fail - filter failed generations
    - -workArea (directory) - generations dump area
    - -echoDir - display generation names excluding other details

    The task of this script is to report generation details based on various filters applied by the user. This script may run independently or may be called from another script also.

    In figure 4.15, we see an example of filtering obsolete generations. The generations which are older (present in dump directory for a long period of time) and not relevant are marked obsolete. The figure displays all the obsolete generations only.

    Some generations may be cancelled by the user after launching them on cluster. Figure 4.16 shows an example for the same.

    Figure 4.17 shows an example of failed generations. The generations which may not complete due to some failures and need to be debugged are marked as failed.

```
{amank}702: filterGenerations -obs yes

### Info:
1). Project : TESTS INTEGRATION:14188, Library: test_RAM:108914,13,0, requested by : gen_support:7780 , is obsolete,
    debugged by : Ashu.Talwar,
    => In directory: /work/amank/testing/generation/WebGen_2016.02.24_12h14m33s
    => Generation is failed

2). Project : TESTS INTEGRATION:14188, Library: test_RAM:131412,14,0, requested by : ritu singh:7607 , is obsolete ,
    debugged by : Ashu.Talwar,
    => In directory: /work/amank/testing/generation/WebGen_2016.02.11_13h59m37s
    => Generation is success
```

Figure 4.15: Output: filterGenerations-obsolete

```
{amank}701: filterGenerations -can yes

### Info:
1). Project: TESTS GEN RON:15741, Library : test_1_cut:145627,2,0 , requested by : user6:7459 , is cancelled ,
    => In directory: /work/amank/testing/generation/WebGen_2016.02.04_16h07m25s
    => Generation is failed

2). Project: TESTS GEN RON:15741, Library : test_5_cut:143037,1,0 , requested by : user4:7780 , is cancelled ,
    => In directory: /work/amank/testing/generation/WebGen_2015.11.20_07h46m06s
    => Generation is failed

3). Project: TESTS GEN RON:19283, Library : test_7_cut:146264,1,0 , requested by : user2:2180 , is cancelled ,
    => In directory: /work/amank/testing/generation/WebGen_2016.04.05_00h40m51s
    => Generation is failed
```

Figure 4.16: Output: filterGenerations-cancelled

```
{amank}703: filterGenerations -fail

### Info:
1). Project : TESTS GEN RON:15741 , Library : test_1_cut:145627,2,0 , requested by : user6:7459 , is latest run ,
    => In directory: /work/amank/testing/generation/WebGen_2016.02.04_16h07m25s
    => Generation is failed

2). Project : TESTS GEN RON:15741 , Library : test_5_cut:143037,1,0 , requested by : user4:7780 , is older run ,
    => In directory: /work/amank/testing/generation/WebGen_2015.11.20_07h46m06s
    => Generation is failed

3). Project : TESTS GEN RON:19283 , Library : test_7_cut:146264,1,0 , requested by : user2:2180 , is older run ,
    => In directory: /work/amank/testing/generation/WebGen_2016.04.05_00h40m51s
    => Generation is failed

4). Project : TESTS INTEGRATION:14188, Library: test_RAM:108914,13,0, requested by : gen_support:7780 , is latest run,
    => In directory: /work/amank/testing/generation/WebGen_2016.02.24_12h14m33s
    => Generation is failed
```

Figure 4.17: Output: filterGenerations-failed

Figure 4.18 shows an example where user wants to exclude obsolete and cancelled generations from the results. Also, until now the default dump diretory was used to fetch results but in this case user has provided a different dump directory.

```
{amank}708: filterGenerations -obs no -can no -workArea /home/amank/genDir/

### Info:
1). Project: TEST_FN100:19287 , Library : test_rom_01:146228,1,0 , requested by : amank:1252 ,
      => In directory: /home/amank/genDir/WebGen_2016.03.29_23h25m49s
      => Generation is success

2). Project: TEST_FN100:19287 , Library : test_rom_01:146228,3,0 , requested by : amank:1252 ,
      => In directory: /home/amank/genDir/WebGen_2016.04.28_23h01m18s
      => Generation is success

3). Project: TEST_FN100:19287 , Library : test_rom_03:146217,2,0 , requested by : amank:1252 ,
      => In directory: /home/amank/genDir/WebGen_2016.04.19_23h14m20s
      => Generation is success
```

Figure 4.18: Output: filterGenerations-workArea

Chapter 5

Generation Stats

Statistics or Stats of a mechanism in operation are important with the perspective of enabling stakeholders to analyze performance and take steps to improve it if required. In this chapter, the algorithm and flow for generating statistics for all the relevant generations launched on the cluster computing facility - SGE and LSF are discussed.

5.1 Purpose

As the jobs are run on cluster of machines, it happens that they take longer time than usual for completion. There could be many possible reasons for this delay such as machine failure, heavy workload on clusters or low priority of jobs. The statistics generated provide ample information to the administrator to identify such issues and take necessary actions, for example kill the job, increase priority of desired jobs, or assign the job to a different machine. One of the crucial factors for monitoring performance of jobs is their run time and we will focus on calculating this time from avaiable values.

5.2 SGE Stats

Figure 5.1 depicts statistics generation mechanism for SGE. On completion of each job fired over SunGrid, SGE makes a record in the accounting file. The size of an accounting file is fixed and as soon as one file size limit is reached, SGE creates another file and starts writing into it. Also, SGE has its own file naming structure. An accounting entry for a job includes details like job id, job name, name of queue in which job has run, hostname, submit time, finish time, project assigned to the job, cluster id and many other details. Thus, SunGrid itself generates wide array of statistics for the jobs and maintains it over

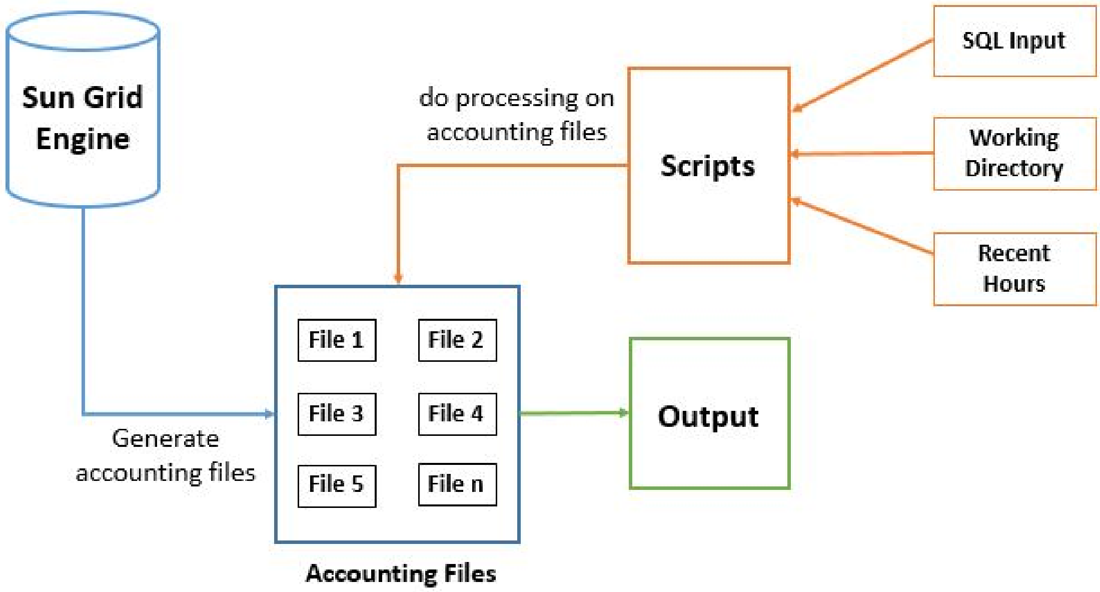


Figure 5.1: SGE Stats Working Mechanism

a large duration.

Now, there could be lots of jobs running in SunGrid carrying out different tasks under different projects. The task is to fetch records of all the relevant jobs i.e. those jobs which carried out generation and which fall under desired set of projects. As seen in the figure, this task is accomplished with the help of scripts with varying options in input parameters. There are three types of input options which can be passed to the scripts:

1. SQL input - In this case, an sql query is made to local database and details of all the generations are fetched. These details include project name, libraryId, submit time, finish time, generation status and few more details excluding job details. For querying, an sql template is used where all the fields to be fetched are already defined. Administrator defines the period over which the statistics generated are to be obtained in the form of start date and end date. For example, start date is specified as 12/8/2015 and end date is 12/1/2016, then all the generations completed within this duration are considered.

2. Working Directory - Here, path of the dump directory for generations is passed. All the generations found in the directory are considered. Each generation will have unique libraryId which helps to fetch corresponding job details from accounting files.

3. Recent Hours - To get statistics about the generations completed in recent time this option is used. Administrator specifies the duration in terms of hours, so that all the generations completed in the recent given hours are considered. For example, administrator wants to get statistics about generations completed in last 5 hours.

All the further implementation is done using SQL input.
Figure 5.2 shows the flow of fetching statistics data and filtering it before displaying it to the administrator.

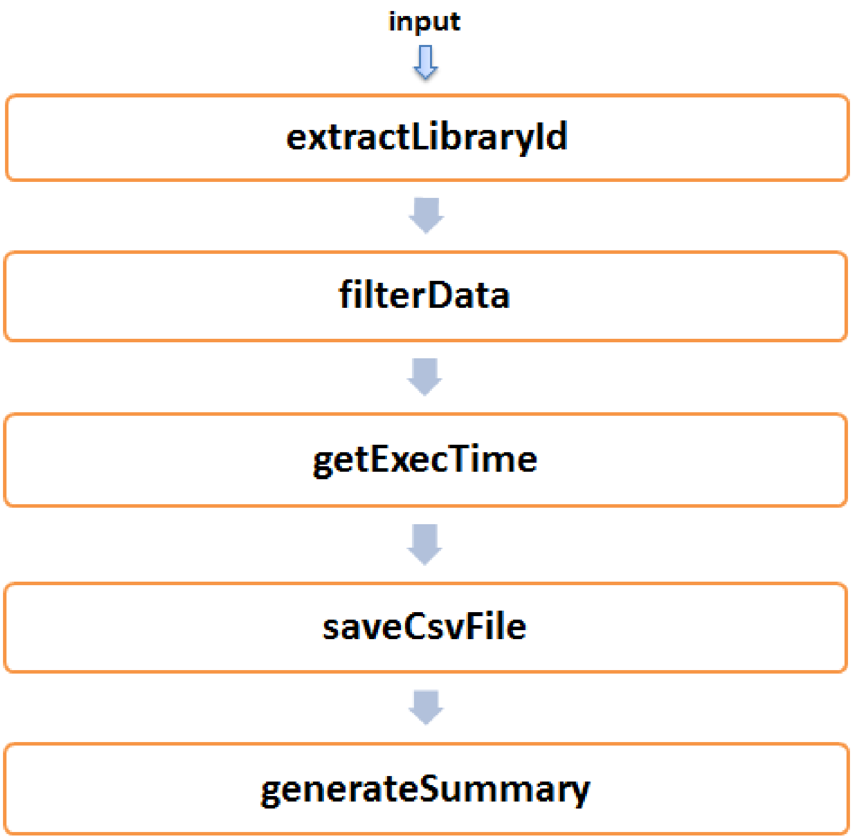


Figure 5.2: SGE Stats Flow

Let us see basic steps of the flow in detail:

- extractLibraryId - After running the sql query, the result generated is stored in a log file. One of the columns of the result is libraryID and it is needed to map to the corresponding jobs. Thus, in this step the libraryID is extracted from the result and saved for future requirements.

- filterData - In this step , we filter out the required accounting files on the basis of file modification time. The time range is obtained from the sql log file and all those accounting files whose modification time falls under this range are considered. Now, as a generation is launched, there could be single job or multiple jobs(one main job along with multiple sub jobs) for it. But all the jobs corresponding to the generation are identified by a single libraryId. libraryId is always unique for a generation. Also the job details could span across multiple accounting files.

- getExecTime - After getting the set of accounting files, the next step is to calculate the execution time for each job. For each generation, a new csv file is created which stores all the job details and execution time is appended to it.
- saveCsvFile - The csv files are saved at a particular location according to their naming convention.
- generateSummary - All the relevant fields necessary to be displayed are fetched and summary is created.

5.3 LSF Stats

As SGE and LSF are different clustering facilities, their accounting mechanisms also differ. LSF uses Platform RTM to generate job statistics. RTM is an operational dashboard provided by IBM for LSF environments which monitors workload and generates reports. Data is fetched from RTM and saved in a CSV file with the help of curl command in UNIX.The RTM dashboard is shown in figure 5.3.

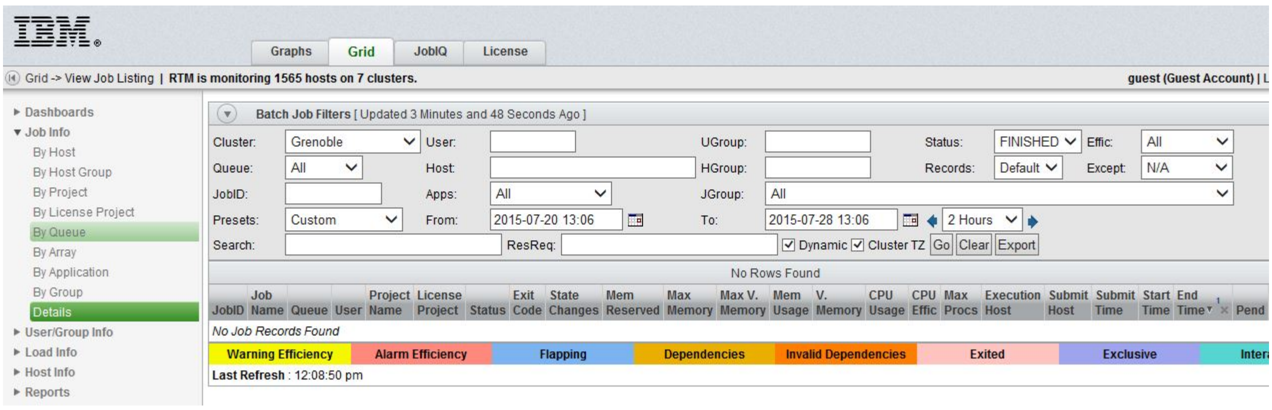


Figure 5.3: RTM dashboard

The algorithm implemented for generating LSF stats is as follows:

Algorithm: getLSFStats

if workArea is valid directory then
    set processingDir = workArea
else if workDir is valid directory then
    set processingDir = workDir
else
    terminate
foreach directory in processingDir:
    if file directory/lsfLibraryId exists then
        fetch libraryId from directory/lsfLibraryId
    elseif libraryId not found in directory/lsfLibraryId then
        fetch it from directory/libraryName
    else
        terminate
    fetch generation_type from directory/generation_type
    set start_date = timestamp of file directory/started
    set end_date = timestamp of file directory/ended
    fetch list of various webgen groups from RTM through curl query and set it as grpLst
    foreach group in grpLst:
        run the curl query
        process the output and store the result in a csv file
    concatenate all csv files into one csv file corresponding to directory
    dump the csv file at csvDumpDir/libraryID/

End of algorithm

A curl query is shown in figure 5.4.

```
curl "http://rtm.gnb.st.com/cacti/plugins/grid/grid_bjobs.php?clusterid=1&ajax_user_query=webgen&
user=webgen&user_clusterid=1&usergroup=-1&usergroup_clusterid=1&status=FINISHED&host_clusterid=1&
row_selector=20000&jobid=&jgroup=/webgen/$group/$generation_type/$libraryId&date1=$start_date&date2
=$end_date&dynamic_updates=on&go=Go&jobs_page=1&report=jobs&export=EXPORT"
```

Figure 5.4: Curl Query

Figure 5.5 shows a snapshot of the logfile generated. Proper message generation and error handling is done and a log file is maintained for this purpose. As seen in the figure, an entry is made for each generation directory encountered and the final status for that generation is noted. The result stored in csv file is shown in figure 5.6. The execution

```
Log generated on 2016-04-08+11:37
==================================

In sample_Generations//WebGen_2016.02.21_21h30m03s/, found library id 136351_8_0 in .library,No data found in xa;, No csv file generated for sample_Gen
erations//WebGen_2016.02.21_21h30m03s/.

In sample_Generations//WebGen_2016.03.08_14h13m31s/, found library id 145997_2_0 in .lsf_group_id,No data found in flow;xa;eldo;, No csv file generated
 for sample_Generations//WebGen_2016.03.08_14h13m31s/.

In sample_Generations//WebGen_2016.03.21_11h22m24s/, found library id 146112_1_0 in .lsf_group_id,No data found in xa;, No csv file generated for sampl
e_Generations//WebGen_2016.03.21_11h22m24s/.

In sample_Generations//WebGen_2016.03.31_07h09m20s/, found library id 146006_4_0 in .lsf_group_id,Duplicate library id, csv file successfully generated
 at /prj/webgen_unicad/memValidKit/2.4-AMAN-00/AMAN//14/6/0/0/6//library_146006,4,0.csv.

In sample_Generations//WebGen_2016.04.01_05h55m24s/, found library id 146006_5_0 in .lsf_group_id, csv file successfully generated at /prj/webgen_unica
d/memValidKit/2.4-AMAN-00/AMAN//14/6/0/0/6//library_146006,5,0.csv.

In sample_Generations//WebGen_2016.04.01_05h55m36s/, found library id 146006_5_0 in .lsf_group_id,Duplicate library id, csv file successfully generated
 at /prj/webgen_unicad/memValidKit/2.4-AMAN-00/AMAN//14/6/0/0/6//library_146006,5,0.csv.

In sample_Generations//WebGen_2016.04.05_11h47m59s/, found library id 146265_2_0 in .lsf_group_id,Duplicate library id, csv file successfully generated
 at /prj/webgen_unicad/memValidKit/2.4-AMAN-00/AMAN//14/6/2/6/5//library_146265,2,0.csv.
```

Figure 5.5: Stats Log file

time for each job is seen in the results. It is calculated as the difference of start time and end time of the jobs.The run-time or execution time indicates the time taken for actual job execution and excludes the waiting time in queues. All these stats generated also reflect on the Webgen portal in tabular format. This conversion from csv-to-table is handled by Webgen team.

```
# job_group:submitted:started:ended:exec_time(seconds):job_name
/webgen/main/sge_user/146265_2_0;1459854269;1459854270;1459860439;6169;main
/webgen/flow/sge_user/146265_2_0;1459857474;1459858354;1459859341;987;macro_noise_charac
/webgen/other_jobs/sge_user/146265_2_0;1459854768;1459854769;1459854786;17;verilog_packed
/webgen/other_jobs/sge_user/146265_2_0;1459854718;1459854725;1459854750;25;mat10_hdl_emul
/webgen/other_jobs/sge_user/146265_2_0;1459854680;1459854697;1459854731;34;cdl_packed
/webgen/other_jobs/sge_user/146265_2_0;1459854683;1459854697;1459854721;24;gds2_packed
/webgen/other_jobs/sge_user/146265_2_0;1459854680;1459854697;1459854716;19;mat10_tetramax
/webgen/other_jobs/sge_user/146265_2_0;1459854307;1459854595;1459854683;88;mcf_setup
/webgen/other_jobs/sge_user/146265_2_0;1459854622;1459854654;1459854679;25;tetramax_packed
/webgen/other_jobs/sge_user/146265_2_0;1459854627;1459854659;1459854677;18;spec_template_packed
/webgen/other_jobs/sge_user/146265_2_0;1459854787;1459854817;1459854842;25;pt_verilog_mapping_packed
/webgen/other_jobs/sge_user/146265_2_0;1459854678;1459854694;1459854711;17;emulator_simvision_packed
/webgen/other_jobs/sge_user/146265_2_0;1459854728;1459854732;1459854757;25;verilog_stim_packed
/webgen/other_jobs/sge_user/146265_2_0;1459854621;1459854653;1459854678;25;emulator_verilog_packed
/webgen/other_jobs/sge_user/146265_2_0;1459854308;1459854597;1459854623;26;ip-xact_bus_definitions
/webgen/other_jobs/sge_user/146265_2_0;1459854621;1459854653;1459854679;26;cdl/ST_DPHD_BB_256x80m4_aTMlmr
/webgen/other_jobs/sge_user/146265_2_0;1459854622;1459854654;1459854681;27;gds2/ST_DPHD_BB_256x80m4_aTMlmr
```

Figure 5.6: Stats CSV file

Figure 5.7 shows the library lookup form on portal. User could search for generated libraries by giving any of the search parameters.

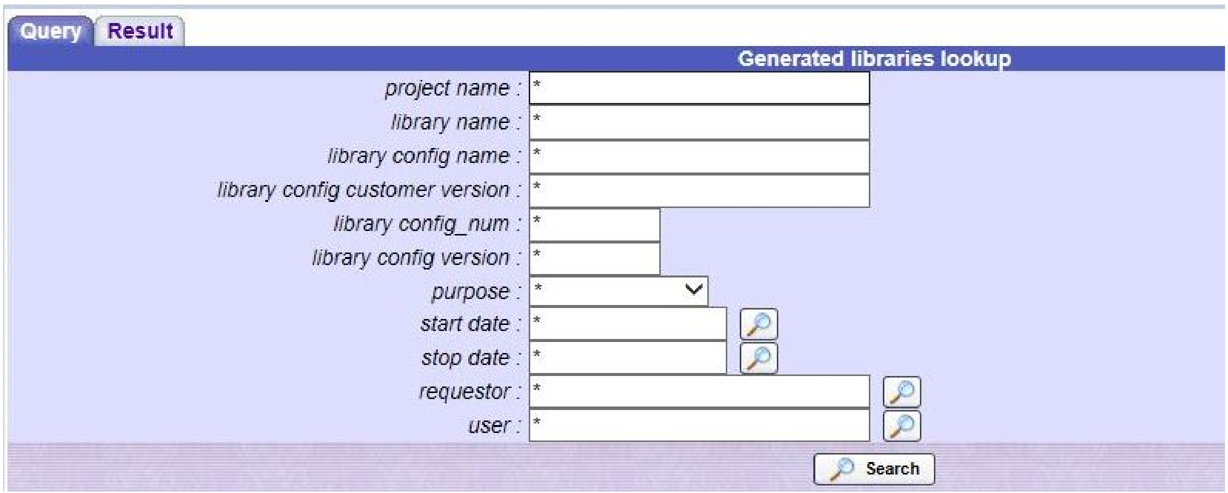


Figure 5.7: Generated Libraries Lookup

Chapter 6

Investigation of a new approach in Steps Compilation

6.1 Problem Statement

A makefile representing compilation flow consists of various steps. A step denotes a target and its dependencies. While compilation, all the steps call a common function. This is depicted in figure 6.1.

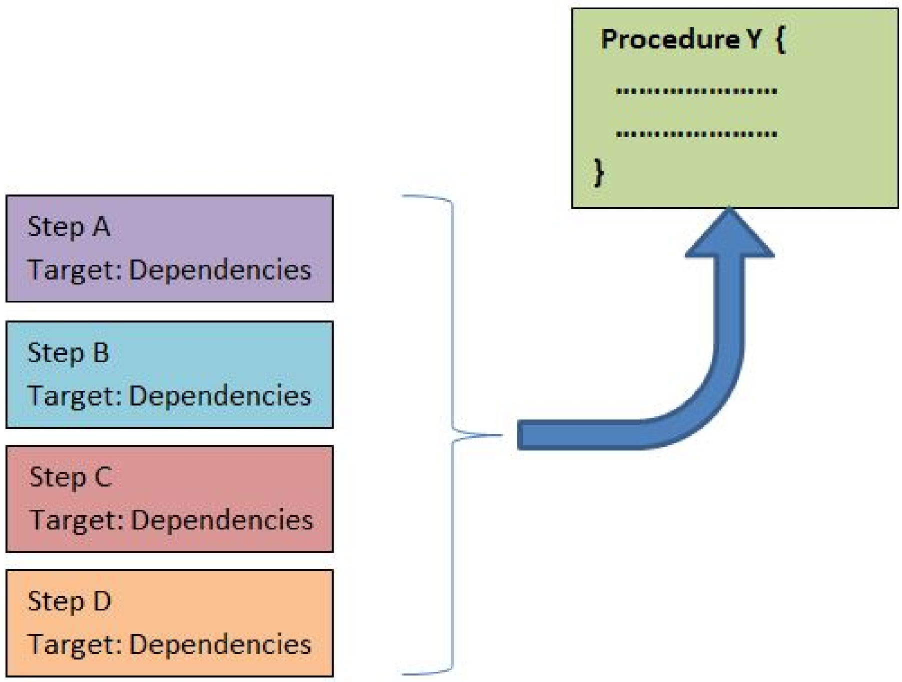


Figure 6.1: Procedure call in Steps Flow

The algorithm for the procedure is as follows:

Procedure stepCall

```
store current process id in file pidList
loop:
    read pid from pidList
    if pid exists i.e. process is running then
        break
end of loop
if pid = current process id then
    //execution of internal logic
    create target file and delete file step.failed
    if failure in target file creation then
        create step.failed file
    append log messages in step.log file
    delete file pidList
else
    loop:
        check if pid exists then
            make current process sleep
        else
            break
    end of loop
```

End of procedure

From the algorithm it can be inferred that each step creates a set of files when the procedure is called. The files are created only for status indication and logging purpose. But as the number of steps would increase, more space is consumed by the files. This issue is one of the main reasons to look out for a better approach. The new approach is intended to do the following:

1. It should replace existing approach but exhibit same functionality.

2. As can be seen in algorithm, multiple processes could execute the same script simultaneously. This property is called concurrency. The existing algorithm synchronizes

them to avoid any conflicts. In the same way the new approach must support mechanisms to handle concurrent access and maintain proper results.

3. The drawbacks of existing algorithm must be addressed properly.

6.2 Proposed Solution

- After rigorous discussion and analysis, it was decided to replace files with tables. A suitable database must be used according to requirements. Databases could manage concurrent access with locks. Also, managing data becomes easier as it provides structured storage.

- Instead of creating files, creating a single table and using the services of database seems a convenient method. Table columns would represent files and rows would represent set of values corresponding to each process. Now, such a database was needed which was lightweight and had negligible communication overhead along with lesser consumption of space. The constraints were laid as it were not any high end application.

- One of the suitable databases for such needs is SQLite. It has following important features[7]:

  1. Zero configuration - There is no setup process for configuration and initiation of SQLite unlike other databases. Thus starting with SQLite is simple.
  2. Serverless - There is no intermediate server and processes accessing database read and write database fils directly on the disk.
  3. Integration with TCL - It is designed to be used easily with TCL scripts. This gives flexibility to user to embed sql code in TCL.

6.3 Implementation Details

Before proceeding with sql code replacement for steps compilation, some standard procedures using SQLite were created and tested. These procedures carried out tasks like connecting and disconnecting with the database, creating a new table, adding columns to an existing table, inserting and deleting data from the table.

Figure 6.2 depicts code for connecting with a database in TCL. A database name is passed to the procedure. The sqlite3 command checks that if database exists then it will connect with the database else it will create a new database and then connect with it. "db" is a handle name which will now control the database.

```
proc connectDB {fileName} {
    if { [file exists $fileName] == 1} {
        puts "## Database $fileName already exists."
    } else {
        puts "## Creating database $fileName"
    }
    sqlite3 db [file tail $fileName]
    puts "## Connecting to the database!!!"
}
```

Figure 6.2: Procedure connectDB

Figure 6.3 depicts code for fetching column names from an existing database. There is a table named sqlite_master which holds information about all the tables in the database. One of the field names of sqlite_master is table_name which stores name of all the tables created in that database. So while fetching column names, we initially check the existence of table by querying sqlite_master. The "eval" method enables to run sql queries in TCL.

```
proc fetchExistingColumns {tableName} {
    set schema [db eval {select sql from sqlite_master where tbl_name=$tableName}]
    regexp {.*\((.*)\).*} $schema match columns
    regsub -all " text" $columns "" columns

    ## removing spaces from list of column names
    set columns [string map {" " ""} $columns]

    return $columns
}
```

Figure 6.3: Procedure fetchColumn

Figure 6.4 depicts code for adding column to a table. Table name and list of columns to be added to it are passed to the procedure. If the table exists then its existing columns are fetched and compared to the column list. Any extra columns are added to the table. And in case the table does not exist, new table is created. The default data type is "text". There is no separate procedure for creating a new table as this procedure provides the functionality for the same.

```
proc addColumn {tableName colList} {
   if { [db exists {SELECT tbl_name FROM sqlite_master WHERE tbl_name=$tableName}] } {
       puts "## Table $tableName already exists. Checking columns..."

       ## fetching column names
       set columns [fetchExistingColumns $tableName]

       ## checking columns and adding new columns
       foreach colName $colList {
           if { [lsearch [split $columns ","] $colName] == -1 } {
               set alter_query ""
               lappend alter_query "ALTER TABLE $tableName ADD COLUMN $colName text"
               db eval [lindex $alter_query 0]
           }
       }
   } else {
       ## creating new table
       set colList [join [split $colList] " text, "]
       lappend colList "text"
       set create_query ""
       lappend create_query "CREATE TABLE $tableName\($colList\)"

       db eval [lindex $create_query 0]
       puts "## Created table $tableName!!!"
   }
}
```

Figure 6.4: Procedure addColumn

Figure 6.5 and Figure 6.6 depict the code for inserting data into the table. Table name and and an array consisting of column name-value pair is passed to the procedure. It is divided into two parts. The first part checks if there is any mismatch in the number of input columns and existing columns. And it also checks the validity of column names. The second part checks if the same set of data already exists in the table. If it does not exist then data is inserted into the table.

```
proc insertData {tableName db_arr} {
   if { [db exists {SELECT tbl_name FROM sqlite_master WHERE tbl_name=$tableName}] } {
        set dataExists "FALSE"
        set columns [fetchExistingColumns $tableName]
        set colNames ""
        set colValue ""
        array set arr $db_arr

        ## checking columns
        if { [llength [split $columns ","]] != [llength [array names arr]] } {
            puts "Error: Mismatch in number of input columns"
            disconnectDB
            exit 1
        }

        foreach val [array names arr] {
            if { [lsearch [split $columns ","] $val] != -1 } {
                lappend colNames $val
                lappend colValue \"$arr($val)\"
            } else {
                puts "Error: Invalid column name \"$val\""
                disconnectDB
                exit 1
            }
        }
```

Figure 6.5: Procedure insertData - Part1

```
## checking data
set select_query ""
lappend select_query "SELECT * FROM $tableName"
db eval [lindex $select_query 0] tbl {
    set counter 0
    foreach val $tbl(*) {
        if { $tbl($val) == $arr($val) } {
            incr counter
        }
    }

    if { $counter == [llength $tbl(*)] } {
        puts "## Data already exists in table."
        set dataExists "TRUE"
        break
    }
}

 ## inserting values
 if {!$dataExists} {
     set insert_query ""
     lappend insert_query "INSERT INTO $tableName\([join $colNames ,]\) VALUES \([join $colValue ,]\)"
     db eval [lindex $insert_query 0]
     puts "## Data \([join $colValue ,]\) inserted in columns \([join $colNames ,]\) "
 }
```

Figure 6.6: Procedure insertData - Part2

Figure 6.7 shows the code for deleting data from a table and disconnecting with the database.

```
proc deleteData {tableName} {
   set delete_query ""
   if { [db exists {SELECT tbl_name FROM sqlite_master WHERE tbl_name=$tableName}] } {
       lappend delete_query "DELETE FROM $tableName"
       db eval [lindex $delete_query 0]
   }
}

proc disconnectDB {} {
   db close
   puts "## Disconnecting from database"
}
```

Figure 6.7: Procedure deleteData and disconnectDB

The procedures were tested and following are the conclusions:

- Sample databases and tables were created and all the operations were performed successfully over them.
- They were also put to use in Steps Compilation for a single process but without the internal logic. The results were achieved as inteded.
- Further, testing is to be done with multiple processes running simultaneously and locking mechanisms are to be explored.

Chapter 7

Conclusion and Future Scope

We conclude that the work done till now conforms to the objective of the project. The proposed approach for product compilation has been implemented and found preferable in comparison to existing approach. The job monitoring scripts have also been implemented and tested with various test cases. The scripts have been improved for optimization and proper results were achieved.

Also the job statistics were successfully generated from LSF. Its verification was done through internal web portal and statistics were properly reflecting over there.

The investigation of new approach for steps compilation is under progress. Basic procedures have been created and tested for initiation and further the locking mechanisms are to be explored in SQLite. Not limiting to a single solution, we intend to explore other approaches also to get the best possible solution.

Bibliography

[1] Wikipedia, "Physical design (electronics) — wikipedia, the free encyclopedia," 2015. [Online; accessed 19-December-2015].

[2] A. C. Cabe, Z. Qi, W. Huang, Y. Zhang, M. R. Stan, and G. S. Rose, "A flexible, technology adaptive memory generation tool," *University of Virginia*, 2006.

[3] H. Poornima and K. Chethana, "Standard cell library design and characterization using 45nm technology," *IOSR Journal of VLSI and Signal Processing (IOSR-JSVP)*, vol. 4, pp. 29–33, Jan 2014.

[4] E. Jansson and T. Johansson, "Creation of standard cell libraries in sub-micron processes," 2005.

[5] STMicroelectronics, "Stmicroelectronics internal files and training manuals," 2015.

[6] templedf(DanT's Grid Blog-Oracle), "Sun grid engine for dummies," 2009.

[7] sqlite.org, "Distinctive features of sqlite." [Online; accessed 10-May-2016].