MEMORY COMPILER DEVELOPMENT AND OPTIMIZATION

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

Master of Engineering
in
Computer Science and Engineering

Submitted By
Mehzabeen
Roll No. 801632026

Under the supervision of:
Prof. A.K. Verma
Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
PATIALA – 147004

June 2018

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Memory Compiler Development and Optimization*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology, Patiala, is an authentic record of my own work carried out under the supervision of *Prof. A.K. Verma* and refers other researcher's work which are duly listed in the reference section.

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

Signature: Mehzabeen

Mehzabeen

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

Prof. A.K. Verma

Professor, CSED

ABSTRACT

Memory is represented by different views as an example Timing views, Model view, Layout views. For generating memory cell different view is useful. Backend compiler is product which is used to generate memory for various technologies and various specifications. This process is known as cut generation. Two of views, layout view and netlist view are mostly useful. All the technological, architectural and other information are encoded in BE Compiler. GDS file contains the layout information. CDL file contains the schematic information. It means every memory generator configurations have different Backend compiler, but all have a lot of commonly used code. To reduce BE Compiler development time is one aspect of uniBe. To provide single interface for various type of file is also another aspect of uniBE. It supports multiple technology like 28, 14 nm etc. It also Support multiple architecture. User can insert data using single interface. Multiple criteria for compiler are also supported using this product. In uniBe, another major module is ‘Checker’, which checks the input template files, developed by ECE engineers, of a technology with all the cut configurations in such a manner that all the errors in files are detected. Data in files are inserted by end user. Technical data can be incorrect. Checker`s functionality is to verify different kind of checks. UniBe must be faster enough and accurate.

ACKNOWLEDGEMENT

I, student of Thapar Institute of Engineering and Technology, Patiala, have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organization. I would like to extend my sincere thanks to all of them.

The author is highly grateful to Dr. S.S. Bhatia (DOAA), Thapar Institute of Engineering and Technology, Patiala for providing her with the opportunity to carry out her Six Months Internship at STMicroelectronics, Greater Noida.

The encouragement received from Dr. Maninder Singh, Dean CSED, Thapar Institute of Engineering and Technology, Patiala, has been of great help in carrying out the training program and is duly acknowledged.

The author would like to whole heartedly thank Mr. Ashu Talwar, Mrs. Tanvi Ahuja and Mr. Nirav M Patel for their constant guidance and direction. Without their help and motivation, it would not have been possible for this internship to go as smoothly and rewardingly as it did.

It is my absolute honor to place on record my best regards and deepest sense of gratitude to all the mentors for their judicious and precious guidance and inspiration both of which were instrumental in growth as an engineer during my internship.

Finally, I would thank my fellow interns who have always uplifted my spirit and assisted in so many ways to my project.

Mehzabeen
(801632026)

ABBREVIATIONS

CDL	Circuit Description Language
DRC	Design Rule Check
GDS	Graphical Data Stream
JS	JavaScript
LSF	Load Sharing Facility
LVS	Layout Vs Schematic
REGEX	Regular Expression
SoCs	System on Chips

TABLE OF CONTENTS

List of Figures

Chapter 1

Introduction

1.1 General

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.

For designing SoCs, the typical IC design cycle is to be followed. Out of which, we would focus on Physical Design Step. This step converts circuit description (schematics) into geometrical representations (layout). We are concerned with designing memories for SoC. Memories are classified as primary memory like RAM and ROM, based on technology like 65 nm, 45 nm, 32 nm, etc. This project is used to generate memory cuts of different technologies and different configurations. Two main views are created during cut generation which are as follows:

- GDS II view or Layout view
- CDL view or Net List view

Other views can also be created.

- PLEF view
- AREA view

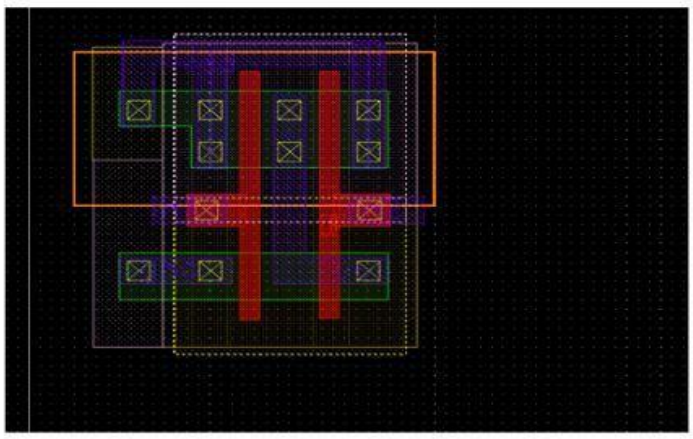

Figure 1.1: Layout of Circuit

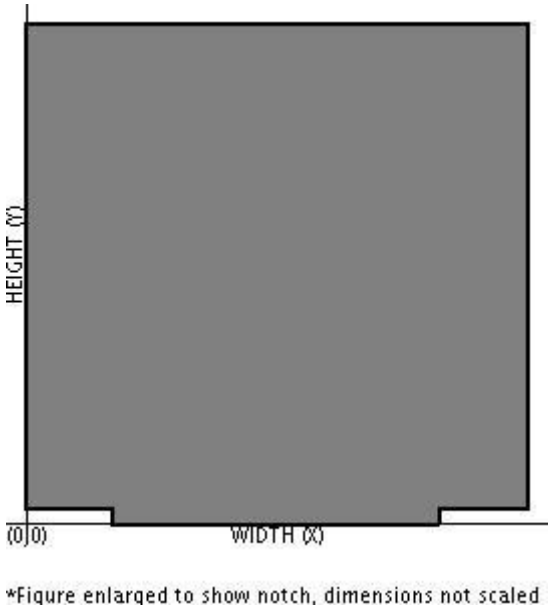

Figure 1.2: Area of Circuit

1.2 Objective of Work

UniBE, which would be a subsystem of Memory Generator to generate a memory cuts. UniBE product supports all technology with Product Standardization guidelines with limited control to user (memory designer).

Currently working on gds and cdl views. Making Universal BE that supports different technologies. Automating different tasks of product generation which are done manually today and reducing the product validation and generation time as optimal as possible.

1.3 Motivation

A user will provide the specification and based on that we develop the memory. Again for different memory needs, we need to change the compiler configuration which is taking more time. We need something generalized structure for all memory architecture. For that we need to concentrate on layout part of memory to make the process fast and reliable. So our aim is to optimize the code and automate the thing. The input should be simpler so that customer can develop the memory cut and if there is any issue found ,can be solved by the expert person. Thus reduces the load at the expert and provide satisfactory solution to customer.

1.4 Scope of Work

The guideline method of reasoning of the project is to give an interface between others instrument which makes the memory cuts and diminish the compiler advancement time. Making the one to one BE compiler into an Universal BE compiler, i.e. UniBe thing that supports all designing. UniBe will have the ability to make view for every technology, no need to use different compiler to create view.

To automate the process we write one text file or provide GUI interface which has all data about to generate the view of memory. The layout structure will parse the file and generate the product. So our task is to generate the different cut and validate it. Thus it will saves the design time. We just need to validate the cut and found any issue then fix it.

1.5 Activities

Main use of BE Compiler is generation of layout and net list as per data of specification. For different technology different BE compiler is come into picture. Every Webgen Configuration have different BE Compiler, but all have a lot of commonly used piece of code. UniBe is used as a Universal for generating layout and net list for almost each kind of technology, for which we need to give following inputs:

- Template files of a technology (For generating different cuts)
- Cut Configurations (For specific cut).

In UniBe, one of the other major module on which I worked is 'PINS'. PINS are classifies as:

- **Signal Pins:**
  Used for transmission of data signals.
- **Power Pins:**
  Used to provide power supply.

There are two basic views on which I am working:

- **GDS:**
  It shows the cut in a layout form, i.e. in a drawing way which we can be seen using Virtuoso tool.
- **CDL:**
  It shows the schematic view, i.e. output is generated in a textual format.

Activities are as followed:

- Understanding various processes involved in the Circuit Design flow
- Understanding Technology that can be useful in project infrastructure
- Learning the Reflection API, understanding the usage of Reflection API and analyses Java pattern.
- Learning shell script to validate the product.
- Understanding the Memory Structure
- Understanding of pins
- Understanding the virtuoso tool
- Learning local cut generation and cut generation on UniGen.
- Understanding Collection Framework and Regex for Pattern Matching in java.
- Implement pins for UniBe for different views.
- Done improvements in UniBe.
- Validation of UniBe.
- Understanding the Template files.
- Done improvements in Template files.
- Done improvements in other files of UniBeKit, i.e. in .js files.

UniBE, which would be a subsystem of Memory Generator to generate a memory cuts. UniBE product supports all technology with Product Standardization guidelines with limited control to user (memory designer).

- User Control.
- Support different interface.
- Support various bank architecture.

Main use of BE Compiler is generation of layout and net list as per data of specification. For different technology different BE compiler is come into picture. Every Web gen Configuration have different BE Compiler, but all have a lot of commonly used piece of code.UniBe is used by some technologies for generating layout and net list, for which we need to give following inputs :

- Template files of a technology (For generating different cuts).
- Cut Configurations (For specific cut).

In UniBe one of the other major module on which I worked is ‘Checker’, which check the input template files, developed by ECE engineers, of a technology with all the cut configurations in such a manner that all the errors in files are detected. Data in files are

inserted by end user. Technical data can be incorrect. Checker`s functionality is to verify this kind of checks:

- Find checks and classify it.
- User must define data in given form.
- Dependency check, Ex- pair.
- Data range.
- Possible value.

Checker is further divided in two different parts:

- Static Checker.
- Dynamic Checker.

1.6 Existing System

1.6.1 UniBe

Existing system for UniBe was different java project for unlike technologies and different mode like High Speed mode memory, single port memory, dual port memory, etc. But, UniBe is a single project which take input of different template files for each technology and can product desired output files according to the requirement.

In order to develop the memory Compiler, The user will provide INPUT PARAMETERS Configuration file along with needed other things like FE product, BE-Lib, Modules, Memory-Cell. Using these six products including BE product generated by our team will provide to COMPILER and develop the view of memory cut as PRODUCT.

To generate one mature memory cut will take 10-12 months and then it is deliver for fabrication.

This thing we have done for one particular architecture. Now parameter change then we need to repeat the whole process to generate memory cut although it is belong to same technology. The existing approach as shown in figure. This approach is trust-worthy but taking long time to generate the memory cut. We need one another approach which maintain the flexibility of process and take less time compare to conventional approach. So it is demand to develop new approach should be feasible, efficient terms of time and flexible.

Process flow:

- To begin with Memory Designer solicitation to produce the memory cut.
- Web gen send request with data to the unibe.
- Unibe Module working flow.
    - Take command.
    - View call.

    - Unibe request to create object of top structure from the parser.
    - Parser return object of structure.
    - This object is converted into the specific object using jar file.
- Pass object to view generator.
- Generate views.
- View is returned (cdl or gds).

1.6.2 Checker

Present system of Checker was kind of a manual system. Data in template files are inserted by end user. Technical data can be incorrect. Checker is functionality to verify different kind of checks. User are allowed to write mathematical condition or ternary condition. Now challenge is how to evaluate that conditions when there is lot of combinations possible and all these values are depends on specific counter value.

Suppose counter value is 100 and total equations are 100 and two possible values of each equation. We will get (1000.100) possible combination. In the Present system user used to manual check all the different and possible values for each attributes and check if any do have any invalid value for that corresponding attribute or not from the template files.

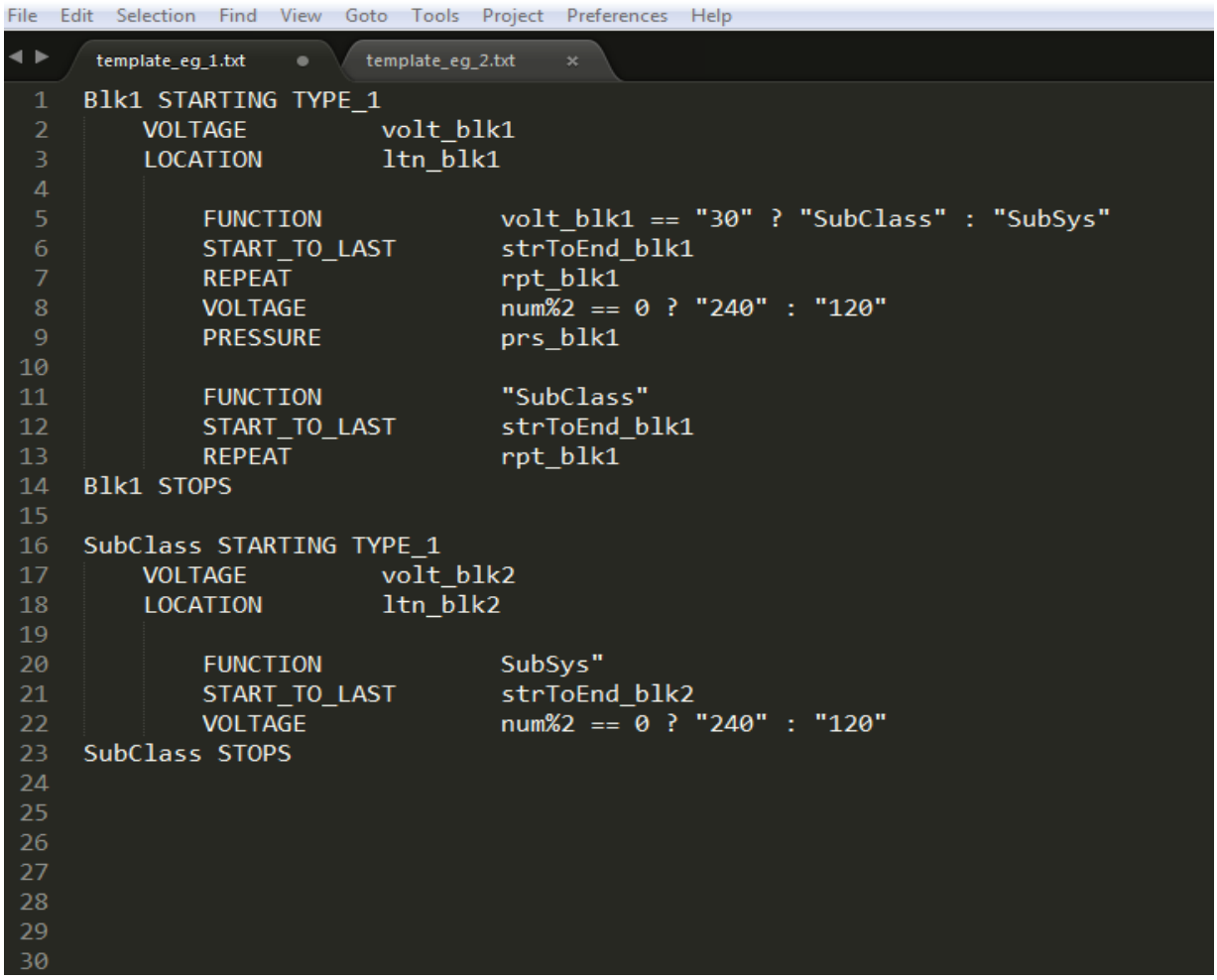

Figure 1.3: Template File Example 1

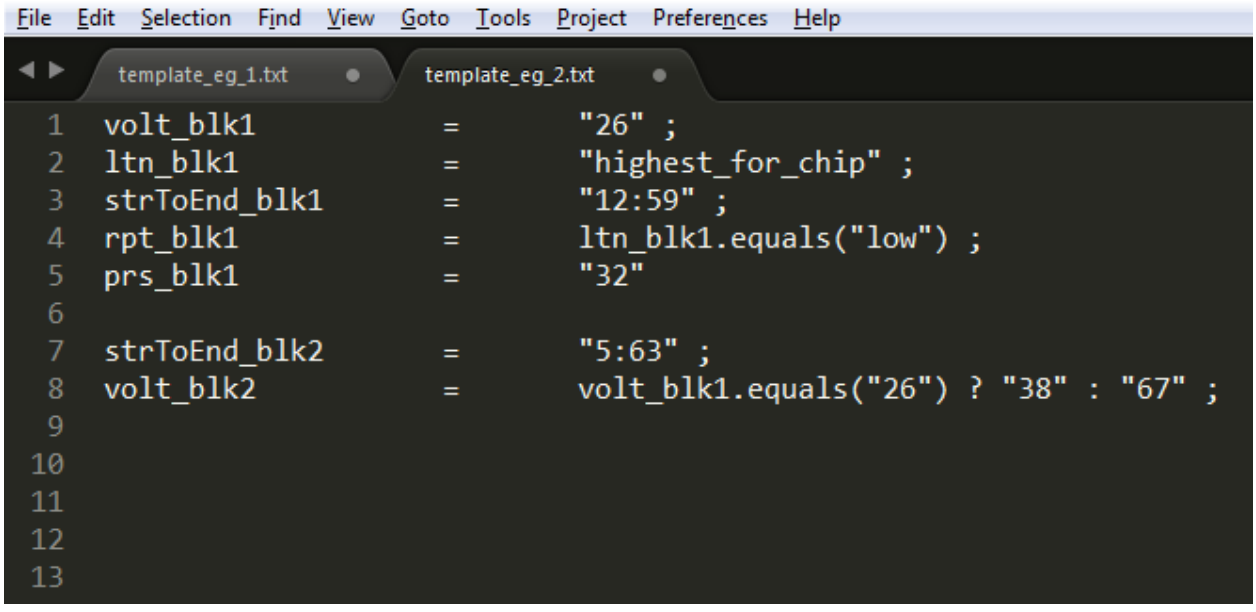

Figure 1.4: Template File Example 2

Chapter 2

Literature Survey

2.1 Brief Overview of Memories

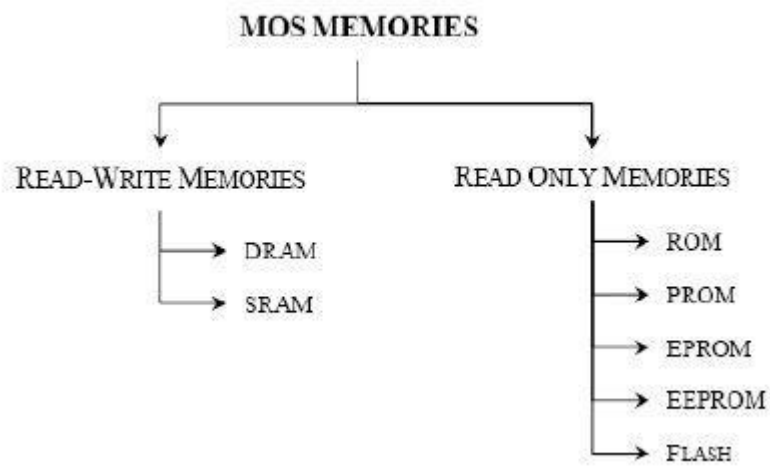

Figure 2.1: MOS Memories

Above is the hierarchy of memory provided by my team.

Read-Write memories: Dynamic RAMs and Static RAMs, allow the user both to read information from the memory and to write new information into memory while it is still in the system

There are two types of memory Read only memory and read-write memory. Dynamic RAM and Static RAM which is read-write memory which allows user to read and write information from memory and into the memory.

ROMs, EPROMs, EEPROMs are read only memory which are generally used to store data. However we can write limited time into EEPROMs while it is in the system. Read only memories are nonvolatile it means that it can retain its data when power supply is off. This is not the case with RAMs.

RAM is volatile memory as data is lost when power supply is o this is because RAM stores data as charges on capacitor as capacitor requires constant power supply so when power is o capacitor gets discharge and information gets lost. There are two types of RAM:

1) Static RAM (SRAM): Its cells are made up of latch composed of cross-couple inverters to store information. Because of this data remains in the cell as long as power supply is on.

2) Dynamic RAM (DRAM): DRAM uses capacitor to store the data charge in capacitor represent data. So we need refreshing circuit which periodically refresh the charge in capacitor.

ROM stores information on the basis of transistor's that joins rows and columns. ROM has good speed of reading compare to read-write memory. All ROMs are non-volatile but the difference is in the way that writes data in memory. One of the simple approach is to write information is when the ROM is developed once the information is written it cannot be altered this type of rom is called mask programmed ROM [1].

2.2 Architecture of Memory

This organization mostly developed Random Access Memory architecture. It is called random access because we can access any address (Location) in random order with fixed time. If we want to access first location or last location it will take same time independent of physical location. Memory is made up of CORE, IO, ROWDEC and CONTROL unit [1].

Core is 2D-array of memory cells. Each cell is a basic memory cell which is capable of storing 1 bit of information that is 0 or 1. So horizontal line is rows and vertical line is columns. Horizontal line is also known as wordline as it is used to select particular word (row). Vertical line is also known as bitline which is used to select particular bit. By combination of wordline and bitline we can select a bit or a cell [1]. Rowdec is used to select wordline while IO will select bitline control will provide information to rowdec and IO to respective lines.

There are mainly three types of memory architecture:-

1) Mono Architecture

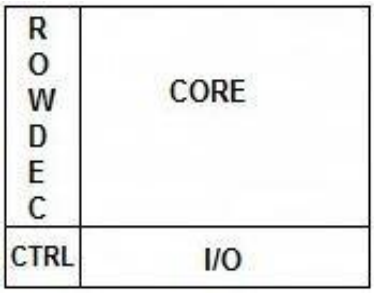

Figure 2.2: Mono Architecture

Here we have single core, rowdec, I/O and control block.

Core is made up of group of memcells which is capable of storing a single bit information.
Group of memcell will form a vertical strap and group of vertical strap will form a core.
To form a vertical strap memcell is placed on top of another memcell so in this way a column is form we will continue this process till words we have.
Once vertical strap is created we will form another vertical strap and will about them on right of another vertical strap. We will continue this process for number of bits time. In this way whole core is generated.
Rowdec is used to select a particular row or wordline. While I/O is used to select particular cell or bit. Control block is used to control rowdec and I/O block.
Advantage: - It's very simple and easy to implement.
Disadvantage: - When number of bits and word increase power consumption to access last bit also increases.

2) Split Architecture

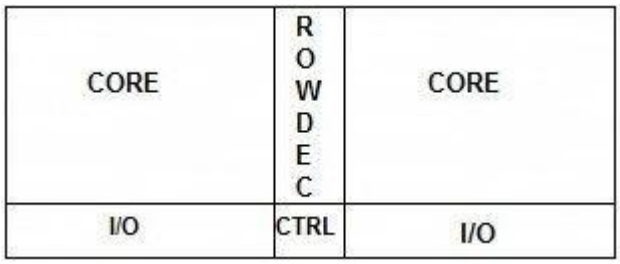

Figure 2.3: Split Architecture

Here we have two core Left core and right core both are created in same way as above core is created. Rowdec will provide access to core and wordline.

We will have two I/O left I/O and right I/O. control block will control the operation of rowdec and I/O

Advantage: - It will reduce the complexity when number of bits are more, so power consumption will also reduce.
Disadvantage: - Problem for number of bits is solved but for problem for number of rows is not solved yet.

3) Bank Architecture

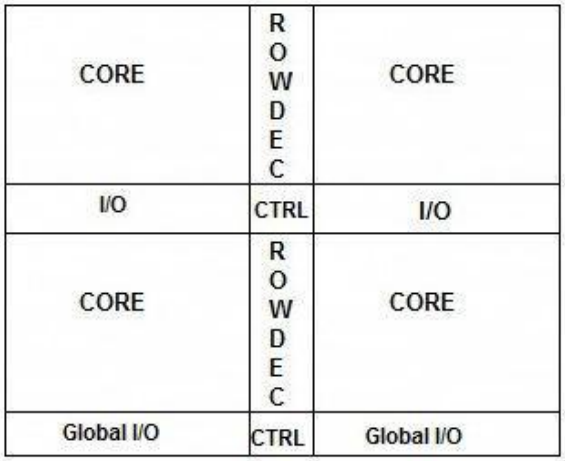

Figure 2.4: Bank Architecture

Here we have multiple Block of core, I/O and control.

Advantage: - Here can see that words are also divided in the form of bank so now we can access fast and power consumption is reduced.

2.3 Some Important Terminology

Memory is always given in terms of words*bits. But when we develop we convert it into rows and column.

Rows = words/mux; cols = bits*mux;

1) **Significance of MUX:** Lets we have Words = 512, bits = 16 and mux = 2.

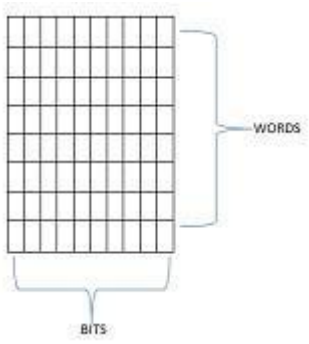

Figure 2.5: Actual Memory

Now suppose the height of memory is not suitable for SOC. We can reduce the height of memory and increase the width by maintaining the same size in terms of words*bits. Memory

=rows*cols = ((words/mux)*(bits*mux)) = words*bits. Although we have converted words*bits into rows*col still we are maintaining same memory size. With mux=2 Rows = 512/2 = 256 Cols =16*2 =32

So now width is twice and height is half. Now if we use mux = 4, height will be reduced to 1/4 and width will be increased by factor of 4. Thus we can adjust the aspect ratio with the use of mux.

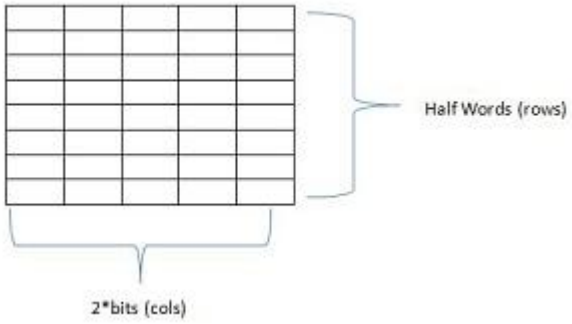

Figure 2.6: Aspect Ratio change of Memory using MUX

2) **Rowdec:** It is used for selecting particular row. Now single rowdec can select from predefined set of rows. For Ex: 4 thus rowdec can select row from 4 rows only. The set of rows from which a rowdec can select a row is known as rowdec unit. Thus we required multiple rowdec to access all rows of memory. Why we can't select all rows with single rowdec? Because the number of rows are not fixed it depends on words and mux so if we create single rowdec it will not work when configuration changes. Thus number of rowdec is calculated on basis of rows.

No. of Rowdec = (rows/rowdec unit) + (rows % rowdec unit == 0?0:1);

3) **IO Cells:** Io cell depends on number of bits. Number of IO's will be placed as many times as bits. In mono architecture bits are even or odd it doesn't matter but in case of split and bank architecture it depends on logic on which side we have to put one extra IO. As here our left and right part will not be identical.
4) **End cells:** These are isolation cells because every IO cells has input and output connection. Left IO corner cell has input connection open and right corner IO cell has output connection open so corner cell's connection need to be closed. Same case with basic memory cell also. So to close this connection we generally use end cells which are also refer as half memcell.

5) **Environment Cells:** Environment cells are used in between core and io cells. There are two types of Environment cells. First one is Vertical Environment strap which is used in core and IO while other is Horizontal Environment strap which are used in Core and Rowdec. Environment Cells are used for to absorb electromagnetic energy generated by the group of cells. So after every predefined number of cells strap cells are used so that effect of electromagnetic is reduced which does not affect.
6) **Control unit:** It is single cell which is placed below Rowdec. Control cell will control Rowdec and IO operation.

2.4 Ways to Design Memory

There are two ways to design a Memory:-

- **Layout Generator or GDS (graphical design system):-** Validated only virtuoso. This will generate one .gds file which will contain all information of memory about its architecture, its abutment number of mux, words, bits etc. .gds file is in non-readable format. This file will be open in virtuoso tool in which we can see the memory and its architecture.

Below image is of .gds file which is open in Virtuoso tool. Here we can see top level of memory.

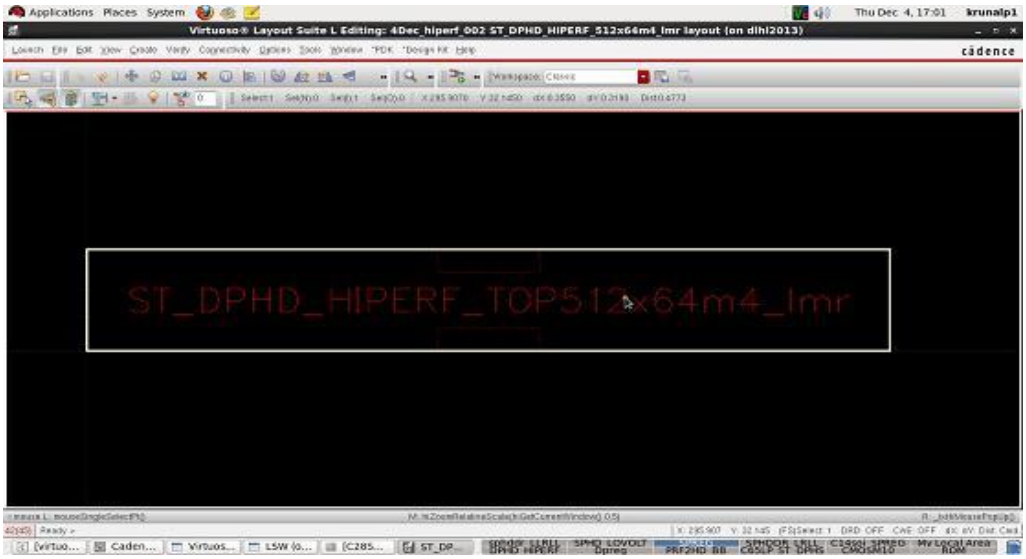

Figure 2.7: Top Level of Memory

This is one hierarchy below the top level. Here we can see the CORE, IO, ROWDEC block. We are able to see two IO's because the cut is generated for Dual port memory.

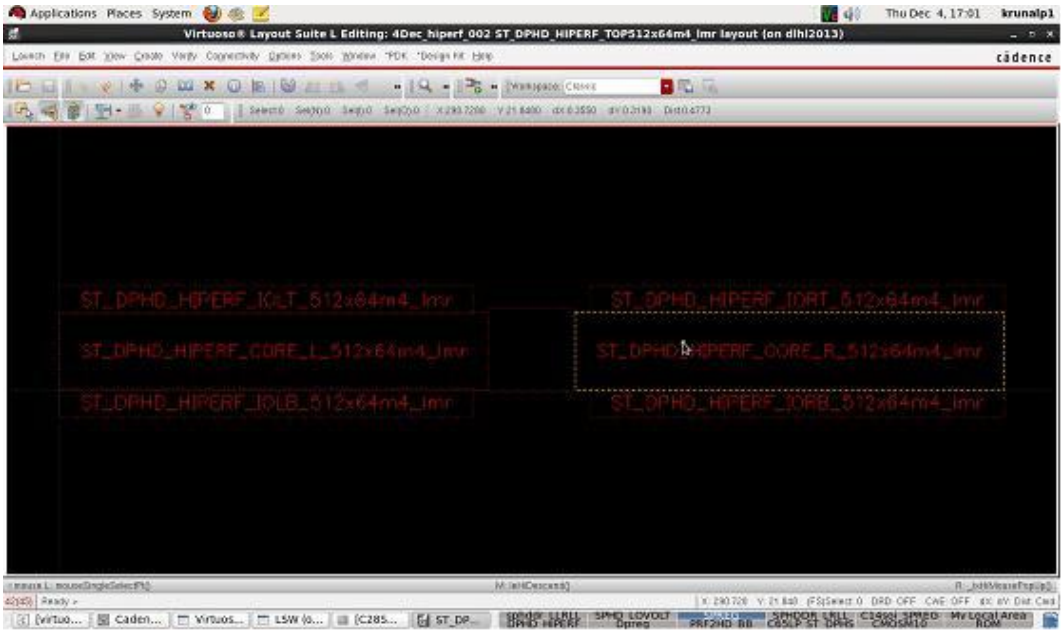

Figure 2.8: Dual Port Memory

Now here we can see the cell which are connected to each other and whole memory in the form of cells.

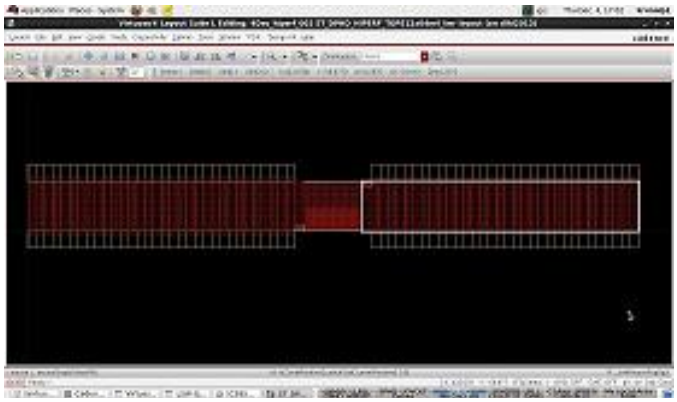

Figure 2.9: Memory in form of Cells

Below figure shows the pins, i.e. power pins, the larger ones in green color and the signal pins, the smaller ones in pink color.

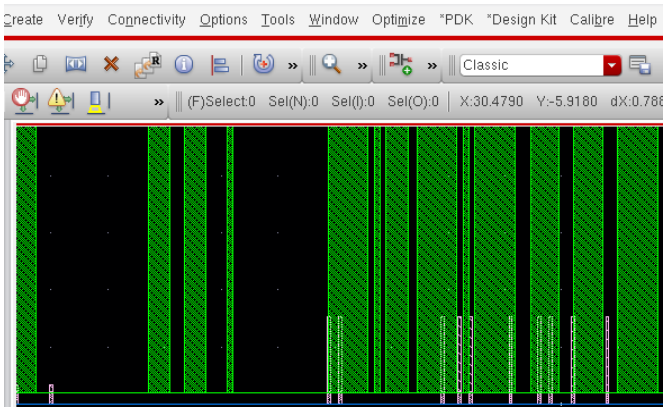

Figure 2.10: Pins in Memory

In the figure below, we can see a 4*4 memcell. The cell which is selected is single memory cell. This basic cell is provided by the memory team. We use this basic cell to implement whole memory. There are other cells as well which are used.

Figure 2.11: Memcell

- **Netlist Generator or CDL (Circuit description Language):-** Validated using Diff with previous copy. This will contain all circuit level information. It contain sub-circuit which have all pin information and how they are connected. This is in readable format we can verify changes by taking diff with previous version file.

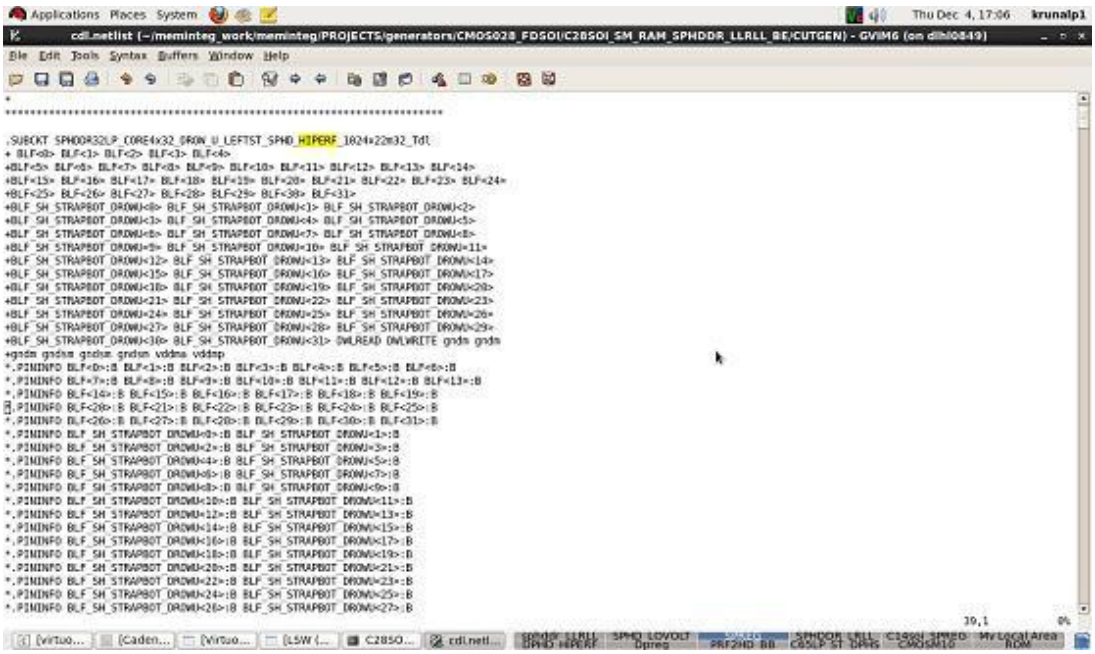

Figure 2.12: CDL File

2.5 Regular Expression

2.5.1 Basic of regular expression

A Regular Expression are patterns used to match character combinations in strings. java.util.regex package is used in java to match patterns using regex. Regular Expression remain same as in shell Scripting or java or perl. Regular expression literals provide compilation of the regular expression when the classes is loaded. When the regular expression will remain constant, use this for better performance. Regex Package in java has a following classes:

- Pattern object: is create by compiling string containing regex. It uses static method to create pattern object. We have to pass regular expression argument in static methods.

- Matcher: is the regular expression engine object that matches the input String pattern with the pattern object created. This class does not have any public constructor and we get a Matcher object using pattern object matcher method that takes the input String as argument. We then use matches method that returns boolean result based on input String matches the regex pattern or not.

- PatternSyntaxException: it indicates the syntax error in a regular expression pattern.

Regular Expression	Description
`\d`	Any digit, short for `[0-9]`
`\D`	A non-digit, short for `[^0-9]`
`\s`	A whitespace character, short for `[ \t\n\x0b\r\f]`
`\S`	A non-whitespace character, short for `[^\s]`
`\w`	A word character, short for `[a-zA-Z_0-9]`
`\W`	A non-word character `[^\w]`
`\S+`	Several non-whitespace characters
`\b`	Matches a word boundary where a word character is `[a-zA-Z0-9_]`.

Figure 2.13: Regex Aliases

Chapter 3

Problem Statement

Essential use of BE Compiler is to generate Layout and netlist as per data specification. For Different technology particular BE compiler is come into picture. We need each technology to go ahead as single platform. As per the examination, A lot of commonly piece of code used in developing the memory cut for different technology. Following are the motivation behind UNIBE:

- Provide single interface for user inputs data.
- Memory designer just fill the template according to as their necessities.
- Support different technology, architecture.

3.1 GDS (Layout) and CDL (Netlist)

A standard cell is a mix of transistor and it interface inward structures. The basic cells includes NAND, NOR, and XOR Boolean limit. Using this basic cells the memory cells are developed and group of this cells are called the **BELib** which is the most precious input to develop the SoC. below Figure shows Schematic view of circuit.

As the name suggest, GDS is the graphical view of memory which elaborate the number of layers in SoC, How is the connection established and what is the area covered by all the components of memory, What is the size of signal pins how the power is driven from in circuitry.

CDL, Circuit Description Language is a kind of netlist, a description of an electronic circuit. It is usually automatically generated from a circuit schematic. It is used for electronic circuit simulation and layout versus schematic (LVS) checks. The Layout Versus Schematic (LVS) is the class of electronic design automation (EDA) verification software that determines whether a particular integrated circuit layout corresponds to the original schematic or circuit diagram of the design.

While designing the SoC, It is mandatory to take a note for DRC value which rule check. As the current flows in the circuitry, it create the magnetic field. So we need to minimize it. Moreover it is also happened that there might be voltage drop between cells while we fetch or retrieve the data. [2]

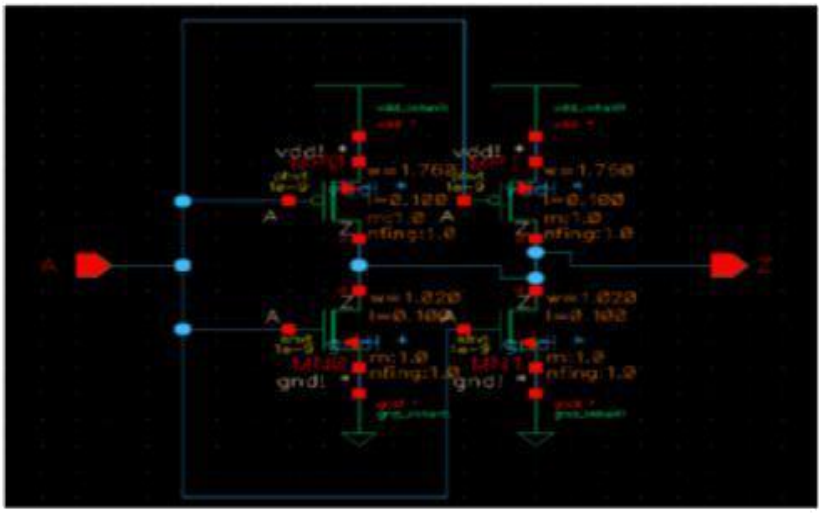

Figure 3.1: Schematic view of circuit

A schematic, or schematic diagram, is a representation of the elements of a system using abstract, graphic symbols rather than realistic pictures.

3.2 UniBe Process Flow

UNIBE process flow is as shown in below figure:

- In the first place Memory Designer requesting to create the memory cut.
- Webgen send demand with information to the unibe.
- Unibe Module working flow

  – Take command

  – View Call

  – Unibe solicitation to make object of top structure from the parser.

  – Parser return object of structure

  – The generated object is converted into the specific object using .jar file.

– Pass object to view generator

– Generate views

– (GDS or CDL) View is returned

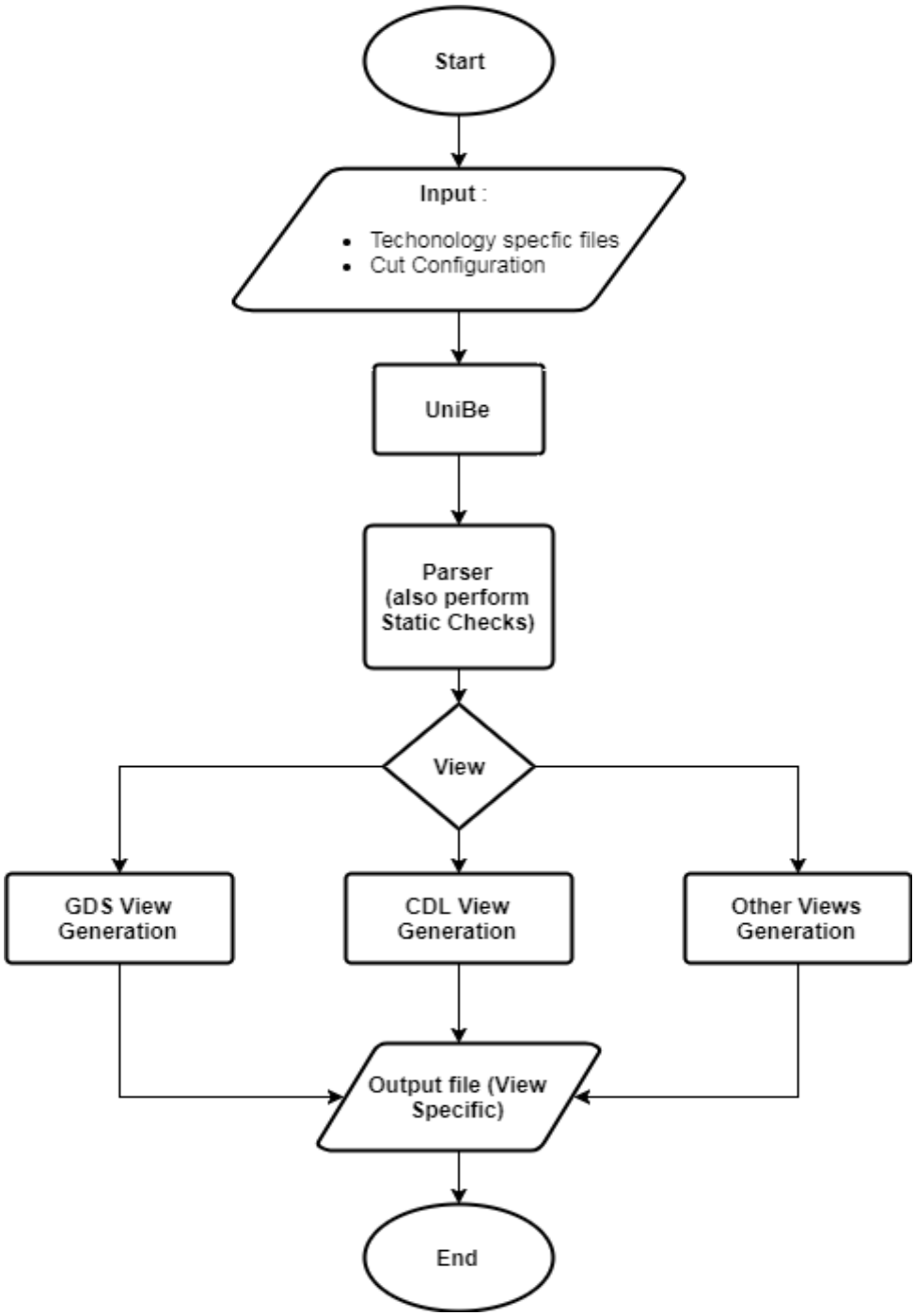

Figure 3.2: Flow Diagram of UNIBE [1]

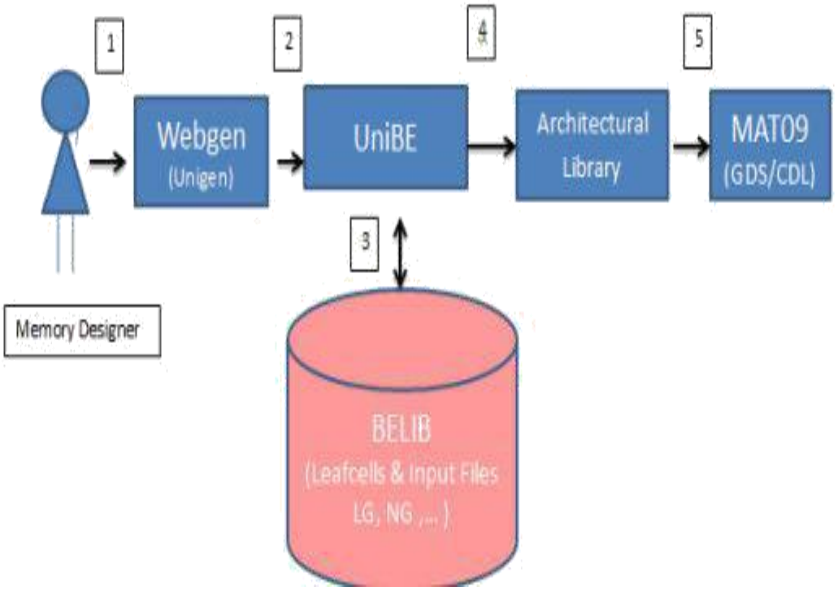

Figure 3.3: Process Flow [3]

3.3 Object Generation stream for Architectural Library

- Demand for building a Structure.
- Demand for building Alias by Structure is sent.
- Request to assemble Block particular square Object.
- Request is sent to generation of Sub Blocks.
- Sub Block Object is returned which is to be set on top of BLOCK.
- Object of Block is returned.
- STRUCTURE Object is Return

3.4 Template Files

These are data files given by end user. User will define detail with specific format. Example of sample template file is shown in following figure

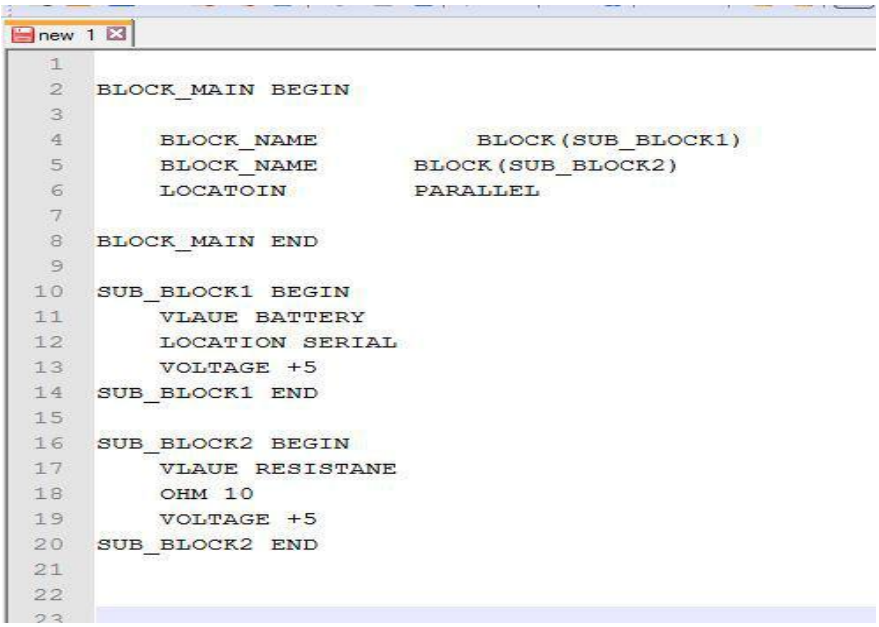

Figure 3.4: Sample Template File

3.5 Functionalities provided by Checker

Checker can run in two modes and gives us power of checking template files in different ways and both of them run in different flow. Different modes of Checker are:

- **Static Checker**: Checking syntax of defined structure. For example it should start with BEGIN and END prefix followed by name of the parameter.
- **Dynamic Checker**: Checking defined parameter and evaluate it. For example, it is checking parameter inside the block like NAME, REF etc. parameter defined properly or not. It is checking the parameter which has variable value and it should be as per domain of the data.

Dynamic Checker also provide us with some functionality which are mentioned and shown in the figure below:

- **Check Type**: User can select if he or she wants to run static checks or dynamics checks.
- **View**: One can select to run checks on GDS or CDL view.
- **Techno**: We are provided with the option of selecting the Technology on which we want to run the checker.
- **ATC**: ATC`s kit should be on or not can be selected by this option.

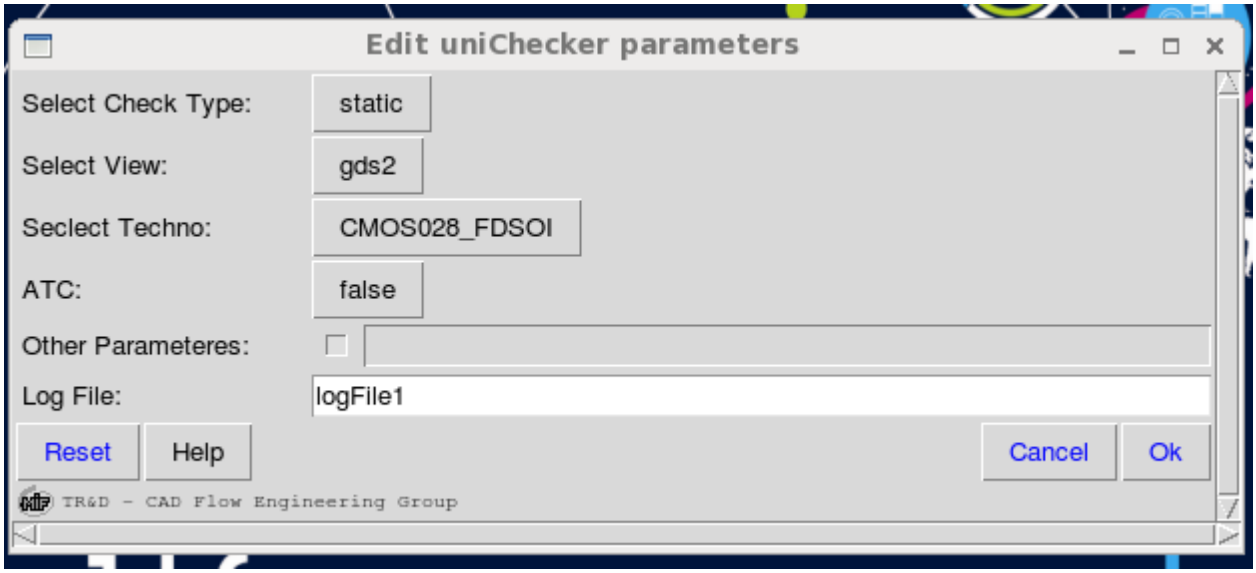

Figure 3.5: Dynamic Checker`s GUI

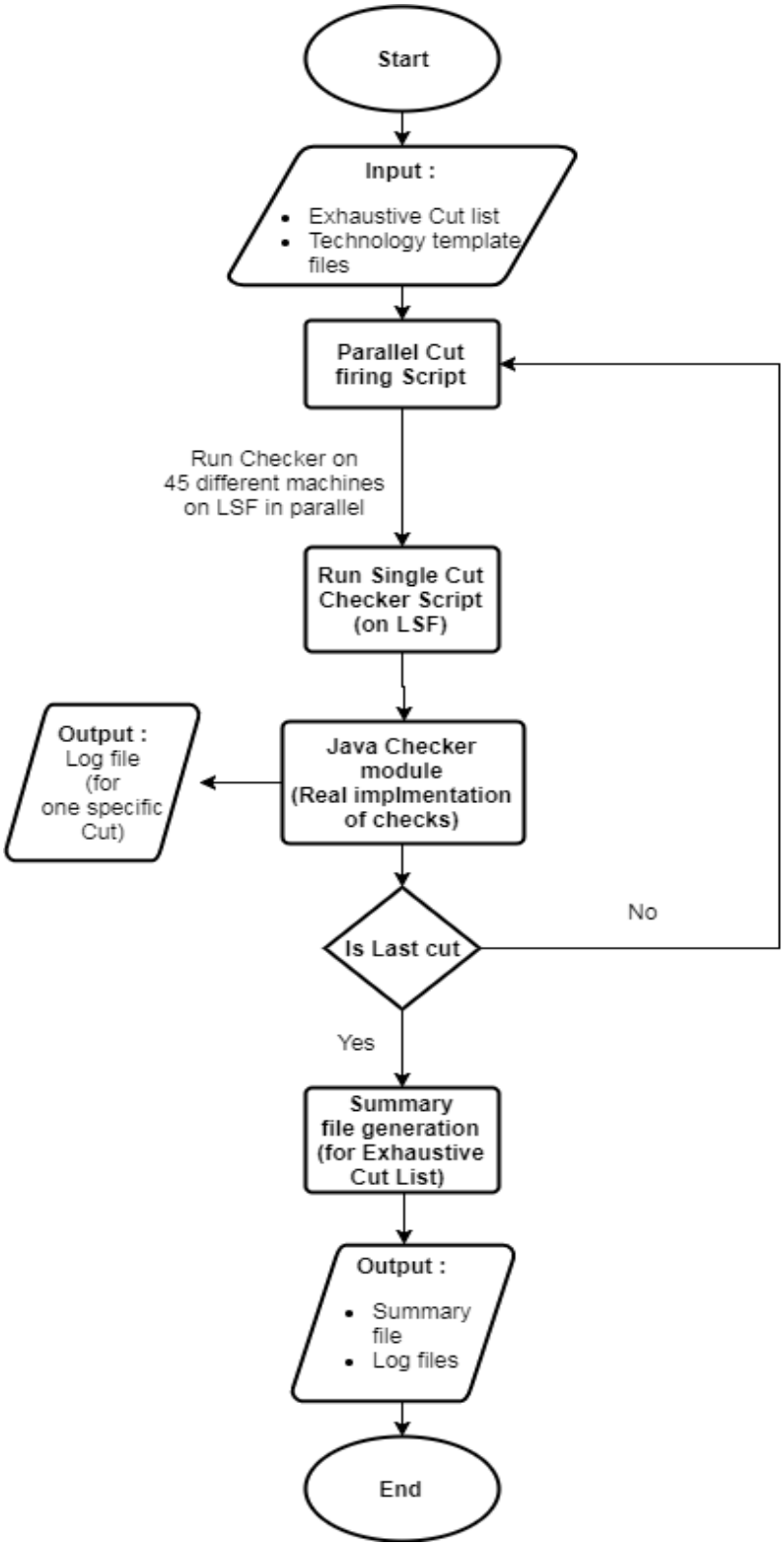

Figure 3.6: Checker Flow Diagram

Chapter 4

Automation to be done

4.1 Multiple Deliveries of product

When first time product is sent to customer we provide only LEF (Library Exchange Format) first we provide prelim delivery.

LEF is a view in which we provide text le in which area of memory with all its co-ordinates as well as position of pins is given. In LEF view area may increase or decrease 10-15% but not more than that. Pin position may also change.

LEF view provides approximate area of memory. Customer is also developing SOC on its end so customer will get idea that whether memory will t in its SOC or not as this process is parallel so changes in area as well as pin may come.

Once customer approves prelim LEF view. Then we provide final LEF view in which area and pin position is fixed. Now customer gets final area and it will check its SOC with respect to our memory whether it will t in that SOC with given pin position or not. Once customer finalizes LEF then we provide him test Chip.

Test Chip contains one or more cuts (memory). When new compiler is created, we need to check its working, how it works after getting fabricated. So for this first we need to provide test chips which are generally large compare to actual memory. This is because we need to check voltage across pins but as we know memory is very small in size so we are not able to tap at exact position to check voltage. If we extend metal from pin then there may be changes of getting sort with other pins metal.

So to solve this problem we increase the size of memory so that we can add contacts which are connected with pins. We add this with particular pins only for which we want to check its voltage.

If more than one test chip cuts are to be send at that time we need to fabricate chip for every cuts which is very costly. So we fabricate only one chip which contains all the test chip cuts that's why it is called test chip. This chip is then tested. If the result is positive then we will send actual cuts to Customer else again improvements are done and again test chip is generated.

Thus after Validation of Test chip. We provide customer actual memory with all its requirement and improvements.

4.2 Technologies Used

The majority of work I have done for this projects, both modules (UniBe and Checker) is in Java along with Shell scripting as supporting language as it is deployed in Unix environment. In Java also the major heart of this project is JavaScript Engine provided by java`s API, which is used to evaluate different expressions in Template files.

A **JavaScript engine** is a program or interpreter which executes JavaScript code. A JavaScript engine may be a traditional interpreter, or it may utilize just-in-time compilation to bytecode in some manner.

Rhino is a java library for java script native object and script handling for java developer. In java version 6 is embedded implicitly as a default script engine. In UniBe and Checker we use Rhino with JDK 7. In JDK 8 we are provided by Java`s Nashorn Script Engine, but due to better memory management and performance of Collection framework we have to use JDK 7, which gives better results for this project then JDK 8.

I am working on **Layout generator (LG)**, **Netlist generator (NG)** and **Checker.**

Languages, on which I have been working: JAVA, shell Script.

Tool Used: Virtuoso, Unigen

Virtuoso Tool:

It is used to view, explore, analyze and verify the .gds files or other circuit designs against design goals so that they can maintain design intent throughout the design cycle.

Launch Virtuoso

- Source the Virtuoso
- Launch it

To view .gds files

- Stream in/Stream out .gds files for Cut Generation

Frequently used keys in Virtuoso

- F - fit or to view single screen
- ctrl-z - zoom in
- shift-z - zoom out
- shift-X – to go inside layer by layer
- shift-B – to go back
- ctrl-f – outer layer only/to come out
- click on Box & type ‘Q’ to view Properties

- click on Box & type ‘m’ to move it
- Type ‘k’ to view width; and many more…

4.3 Implementation Flow

4.3.1 UNIBE

Here user gives input as one single memory configuration for particular technology. He gives input in form of words, bits, mux and other options which he wants. So on basis of this requirements we select architecture. This architecture selection is done by discussing with client. Once architecture is final we calculate number of rowdec where strap cells to be placed, path, via and pins progammation, number of banks, all these things are calculated and design the code to fulfill these requirements. Here I/O block, Control block, Core block, Rowdec block are designed independently and then they are connected. This connection is done on the basis of architecture.
Finally power pins and signal pins are promoted as per the configurations. Calculation for number of rowdec, abutment, strap cell position etc. is done for specific given cut.

We get different requirements from different customers, based on that, we need to design memory for them. Customer needs different memory for different architecture with different technology. So to write code for every memory is infeasible. To save cost and Time we need to automate this thing.
We get different requirements from different customers, based on that, we need to design memory for them.
For one memory to become fully mature takes approx. 12 months. If customer demands a memory then we create and deliver it.
But again same customer needs same memory but with different configuration so again to develop a memory is a very time consuming process.
For example:- If customer requires 16*2 memory and 4096*512 memory then code can implement it without any changes as well as it has to support multiple option (Redundancy, Power supply).

To save cost we design compiler for every memory which take approx. 14 months of time. So we invest 2 more months to design compiler which is capable of generating any memory configuration (cuts).

Following are the figures showing the work flow of old approach and new approach in memory cut generation:

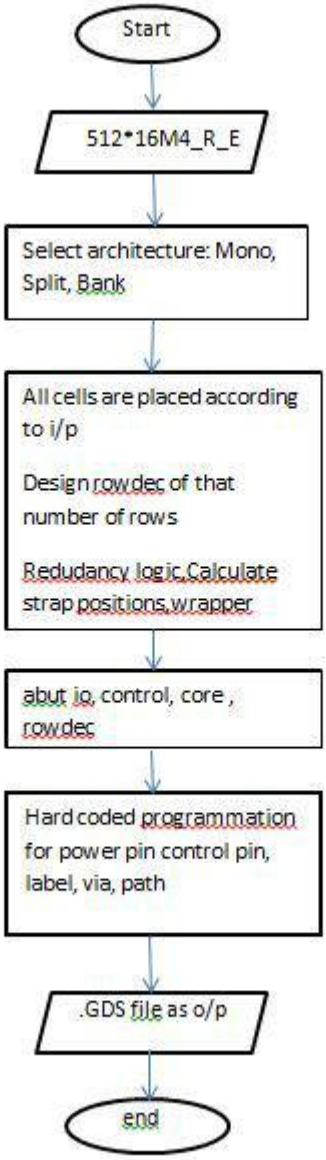

Figure 4.1: Old Approach without Automation

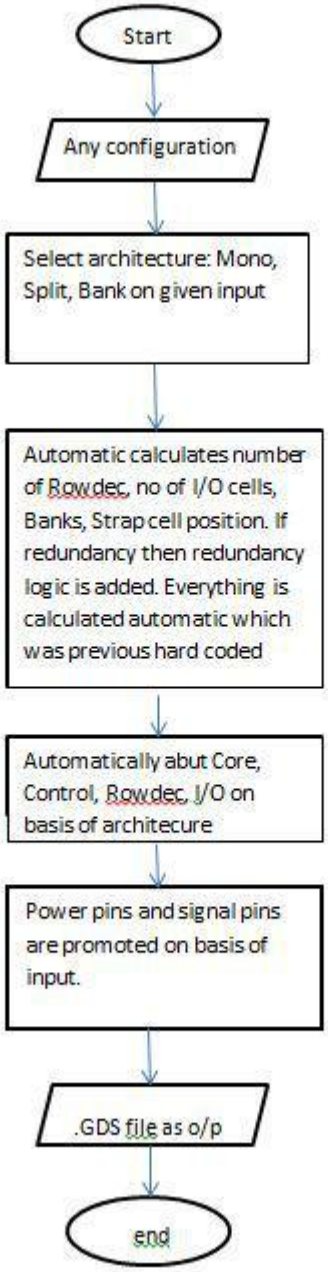

Figure 4.2: New Approach i.e. with Automation

Now customer can generate any cut with any configuration. So this will reduce time, cost and efforts. With the help of this user can also generate area and bitmap. Here user can provide different option on basis of this option it will generate cut. User will be able to generate any cut with any options in given range.

So all these options as well as all its configurations must be supported with single code of compiler.

I am working in converting hard coded programmation of pins into automated one. Pins are the important Module of UniBe. There are two types of Pins:

Signal Pins

- Used for transmission of data signals

Power Pins

- Used to provide power supply

Do important updations in pins code and make the static compiler a generic one by reducing the multi-level hierarchy and eliminate the redundancy from the code by making more generic functions.

4.3.2 CHECKER

The main use of UniBe is in generating design of a Cut of a technology for memory, which is future used on a SoC..UniBe is designed in such a manner that it can generate number of Cuts for a technology with different flavors like Single port, Dual port, High speed, etc. Before we used to have different product or code for each technology and its flavors, but with the use of UniBe we will have one single product used for all technologies supporting UniBe, we will just need to change the set of template files only for each technology.

4.3.2.1 Pseudo-code for Cut Generation using UniBe

- WebGen or UniGen send request with datato the unibe.
- Unibe Module working flow.
    - Take command along with Cut Configuration and Template files (Extracted data and files using Shell scripting)
    - Validate Input command (using Shell Script for UniBe only)
    - Set a specific View (Like GDS, CDL)
    - Unibe request to create object of top structure of template files from the parser.
    - Generate required output file (using object create in last step)
    - Store Output file in current directory (along with other files).
- Generate output file is Streamed in Virtuoso tool.
- Generated Cut can be view in Virtuoso.

- Produced Cut is validated by the Thrid party tool and Designer of Memory.

Cut generation through UniBe is also depicted in the flow diagram below.

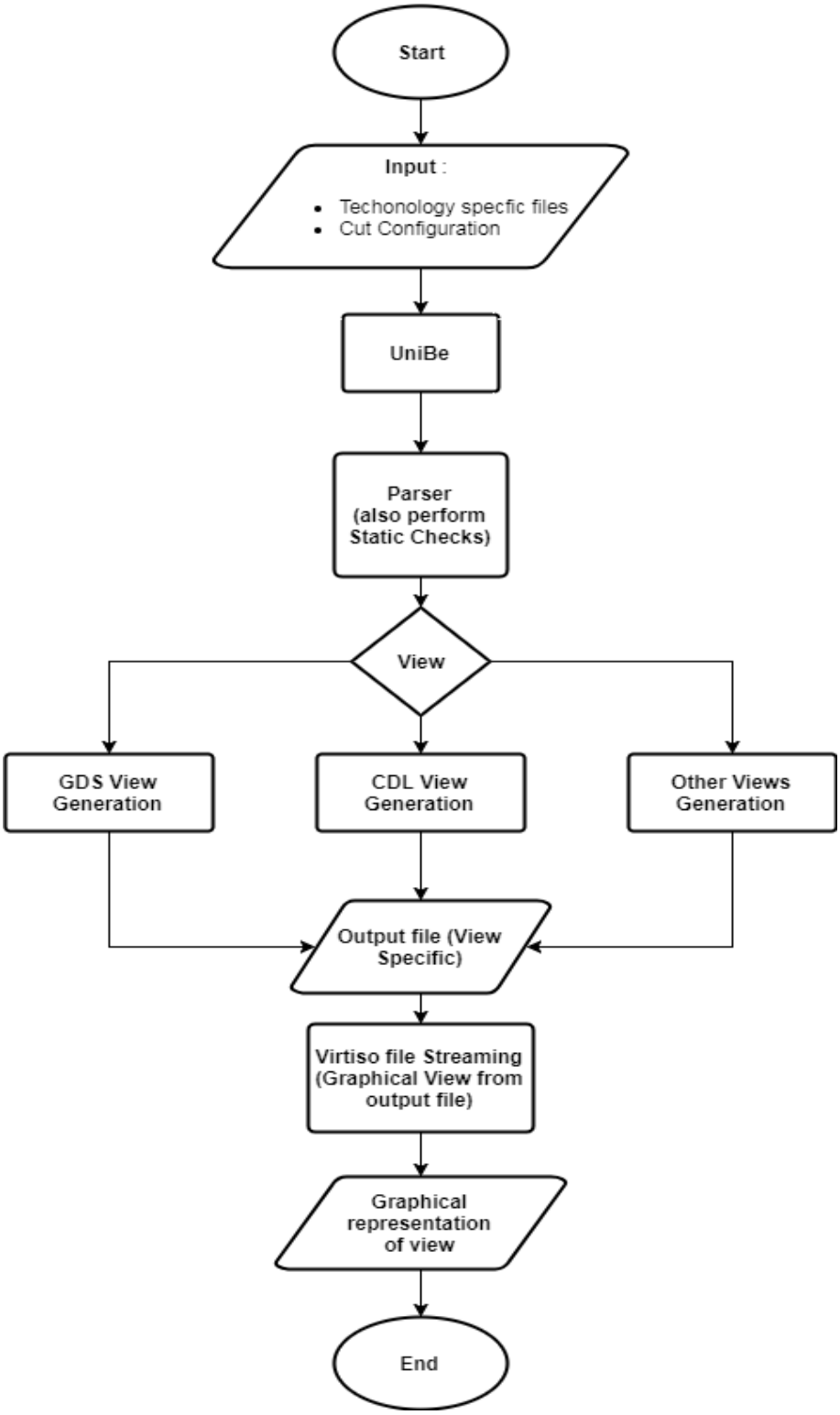

Figure 4.3: Flow for Cut Generation

After the process of cut generation of a particular Cut Configuration and specific technology, we know it is streamed in virtuoso tool for viewing pictorial representation of Cut.

4.3.2.2 Algorithm for running Checker

- Filtering all possible Cut configurations for a technology.
- Storing all these cut configurations in a specified format (to be used by LSF)
- Call Exhaustive Checker Script (for the technology who`s exhaustive cut list is prepared)
    - Take Template files of technology as input. (We can either give path of each file explicitly or we can source that technology on the terminal, where we want to execute Checker)
    - Along with Exhaustive cut list.
- **Exhaustive Checker Script** submits each Job (in Exhaustive cut list) to **LSF**, which parallel distribute jobs on 45 different machines according to the load
- In Each job, **Single Cut Checker script** is called (in which technology template file are verified for a single cut configuration in the list)
- This Single Cut Checker script calls **Java Checker module** after verifying all inputs.
- Java Checker module do the real work of checking the template file. It check all the possible attributes in the template file with the defined checks.
- After checking template file thoroughly, it generate and store all the errors in log file for that specific cut.
- All these log files, in which their Success status is written along with their cut configuration, are stored in a folder.
- Then a script runs over this folder, which uses ‘grep’ command, to find the failed Cuts along with their cut configuration for this set of template files.
- Now the Errors in the set of template files can be reviewed and corrected very easily.

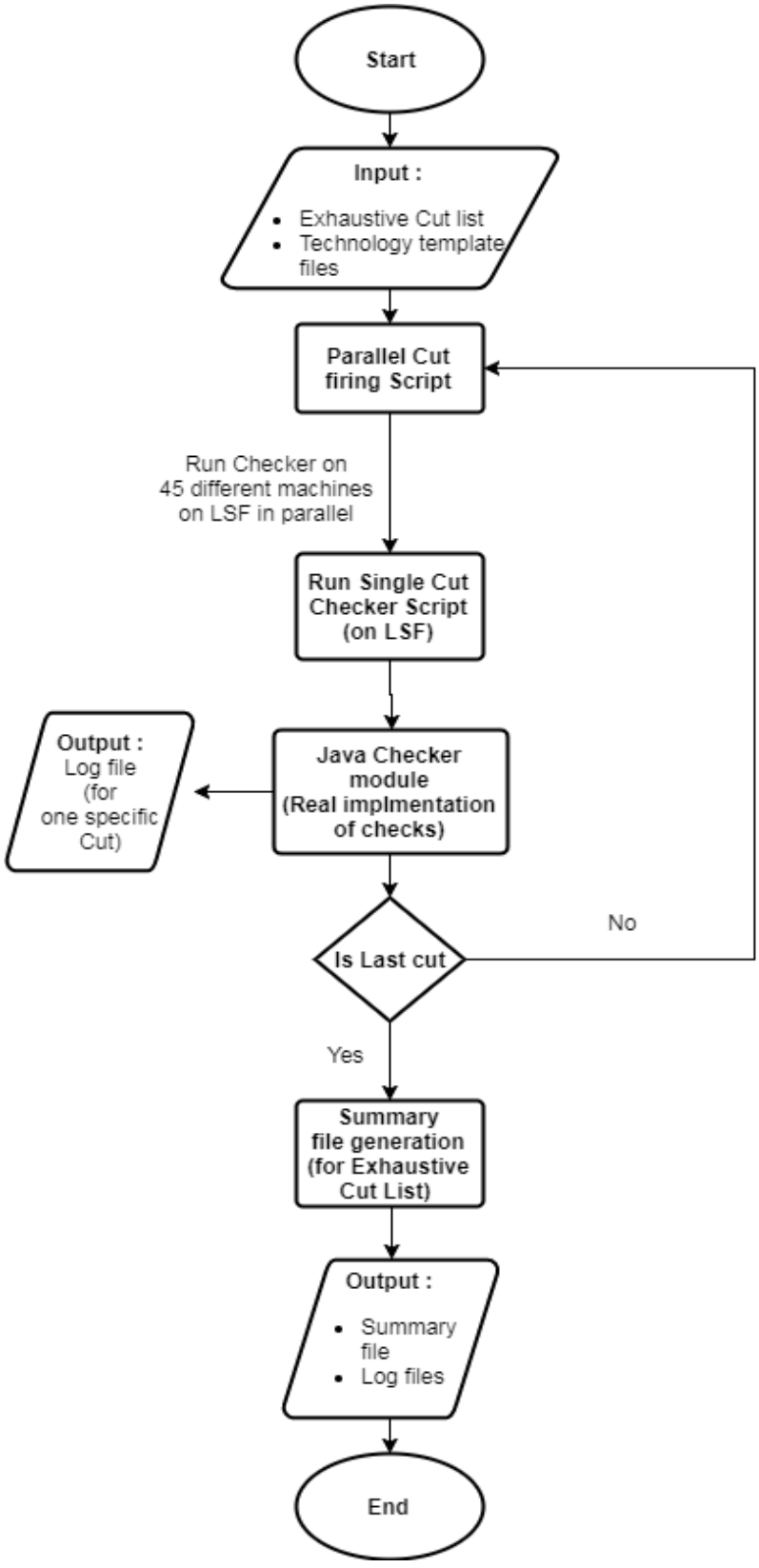

Figure 4.4: Flow Diagram for Checker

Below is the GUI populated if you want to check template files for single Cut by **Single Cut Checker script**

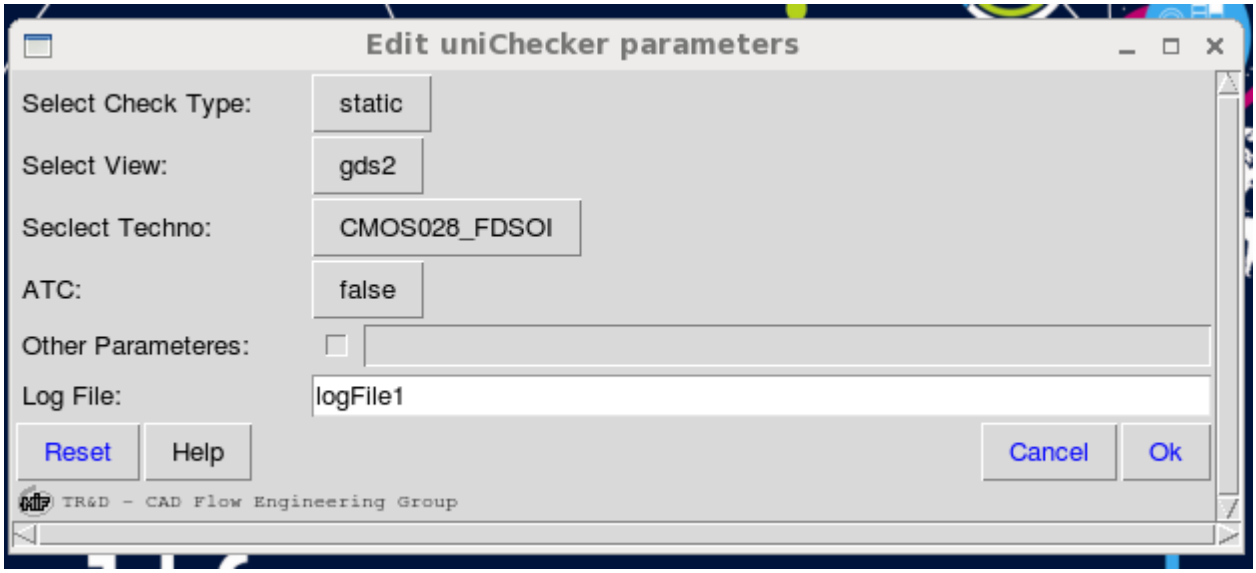

Figure 4.5: GUI for Single Checker script

4.3.3 Understanding JS reloading in project

JavaScript (JS) engine is heart of this project and is used by both UniBe and Checker. It is the module on which the run time of the program depends as there are lot of JS. Let us first understand JS Evaluator which uses this JS engine.

4.3.3.1 JS Evaluator

This is javascript evaluator which is used for evaluation of variable value. The need for this evaluation engine is explain here : For instance above mention variable value is use in let's say on ten place and the value of this variable is different. We can define this variable as constant in ten different place but again we need to do manually which is not desirable.to automate the value of the value of this variable is generic so it can be done through java evaluator. We are using "Rhino" javascript Engine. It is open-source and time taken by it is small.The need for this evaluation engine can be explained better using template files below.

File Edit Selection Find View Goto Tools Project Preferences Help

template_eg_1.txt template_eg_2.txt

```
Blk1 STARTING TYPE_1
    VOLTAGE         volt_blk1
    LOCATION        ltn_blk1

        FUNCTION            volt_blk1 == "30" ? "SubClass" : "SubSys"
        START_TO_LAST       strToEnd_blk1
        REPEAT              rpt_blk1
        VOLTAGE             num%2 == 0 ? "240" : "120"
        PRESSURE            prs_blk1

        FUNCTION            "SubClass"
        START_TO_LAST       strToEnd_blk1
        REPEAT              rpt_blk1
Blk1 STOPS

SubClass STARTING TYPE_1
    VOLTAGE         volt_blk2
    LOCATION        ltn_blk2

        FUNCTION            SubSys"
        START_TO_LAST       strToEnd_blk2
        VOLTAGE             num%2 == 0 ? "240" : "120"
SubClass STOPS
```

Figure 4.6: Template code file Example

File Edit Selection Find View Goto Tools Project Preferences Help

template_eg_1.txt template_eg_2.txt

```
volt_blk1           =       "26" ;
ltn_blk1            =       "highest_for_chip" ;
strToEnd_blk1       =       "12:59" ;
rpt_blk1            =       ltn_blk1.equals("low") ;
prs_blk1            =       "32"

strToEnd_blk2       =       "5:63" ;
volt_blk2           =       volt_blk1.equals("26") ? "38" : "67" ;
```

Figure 4.7: Template JS file example

4.3.4 Optimizing project`s JS Evaluator

After understanding JS Evaluator and knowing that it is part the which takes the major time in the process of Cut generation in this project, it was very importance to reduce it`s time, which I did by using Global and Local Bindings in JS Engine of Evaluator. Before, only one Local binding was used, but I introduced Global binding also so that the variable which remain constant throughout the flow of the execution of the program can be initialized once in Global binding only and need not to load them again and again in local binding.

4.3.5 Reducing JS file evaluations

Some files in the set of template files of a technology are js file, which are evaluated number of thousands of times in one cut generation flow. I reviewed the algorithm used before along with my mentor and proposed a newer version of the previous version of algorithm, in which we eliminated the cases where the js file evaluations was not needed.

Chapter 5

Results and Validation

5.1 Results

Comparison between old Approach (without Automation) and new Approach (with Automation):

Old Approach	New Approach
All the inputs are specific for single memory	Inputs are specific to compiler
Development is simple	Development is complex
It requires 11-13 months to get fully mature	13-15 months to get fully mature
Different cut will take 1 - 1.5 month	Different cut will take 20 min
Ex: for 100 cuts time will be 100*1.5 months	Ex: for 100 cuts 100*20 min

Figure 5.1: Comparison Table

5.2 Validation

For this project, the main output i.e. view specific output file generated by **UniBe** using template files of a technology and cut configuration are verified or tested using a third party tool which is customized by our team only. These output files are streamed on Virtuoso tool for viewing the output design generated by the software and is also being varied by the electronic memory team that it is according to their requirement or not. So, once the product is developed

we need to validate all the changes that are given by the customer are implemented correctly or not. So for this we need to generate the cuts. We have two methods for cut generation: 1) local cut generation and 2) Cut generation through GUI (ugnGUI).

- **Local cut generation**: This is Method is used by developer. Here we have to manually pass all the possible values through commands. We have to run number of commands to generate cuts. By using local cut generation we are able to generate only one cut at a time. So to generate new cut we again need to pass all the arguments to all commands.

Now for customer it is an infeasible approach to learn all the commands to generate a cut. So we develop GUI for customer i.e. ugnGUI.

- **Cut generation through GUI (ugnGUI)**: This is a GUI in which customer has to run only one command and pop-up will appear.

Following are steps which customer has to follow to Generate cuts.

- **Step 1**: Following window appears as soon as ugnGUI command is launched.

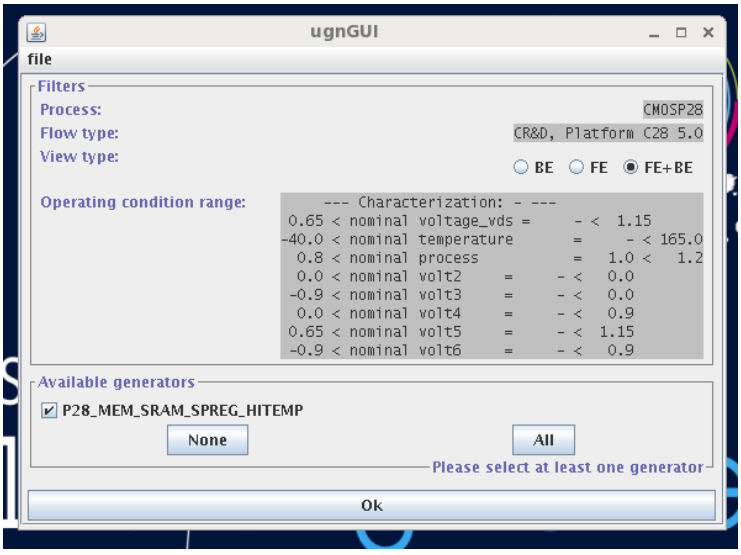

Figure 5.2: ugnGUI Tool used for Cut Generation

Here we can see many options which are explained below:-

PROCESS: - This field indicates technology name. If you have started validation of cuts then it is of particular technology that technology name is shown here.

FLOW TYPE: - This indicates the platform type and its version.

VIEW TYPE: - this indicates number of views supported. If you have not used FE product then here only BE is shown. BE is our product which we have developed.

OPERATING CONDITION TYPE: - these values are derived from CONFI (config) product. If the CONF product is not sourced then these values will not appear here and cuts will not generated.

AVALIBLE GENRATORS: - This field will contain compiler for which you are generating cuts. If this is missing then we have not sourced BE product. If this field shows some other compiler's name then we have sourced wrong BE Product.

- **Step 2**: After clicking on OK. Following GUI appears.

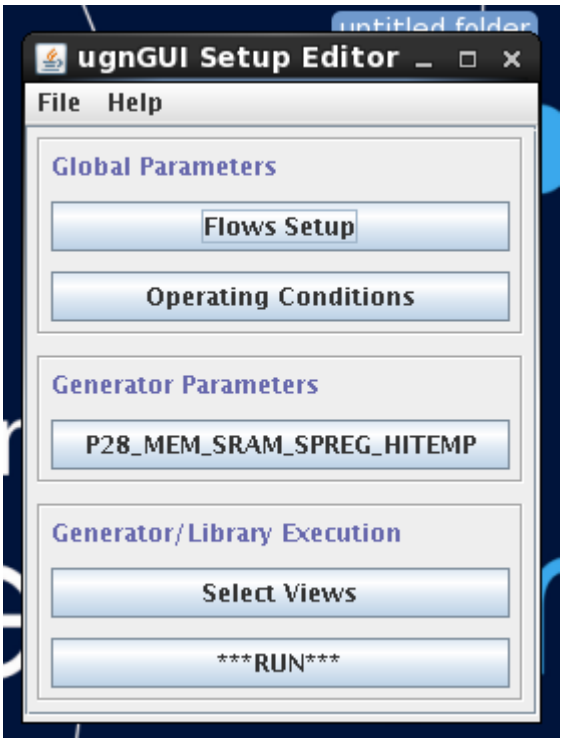

Figure 5.3: ugnGUI Setup

Here you need to first click on flow setup which will open following GUI opens. This will create Library which will contains all your views and all other parameters. You need to give value to the Target library name.

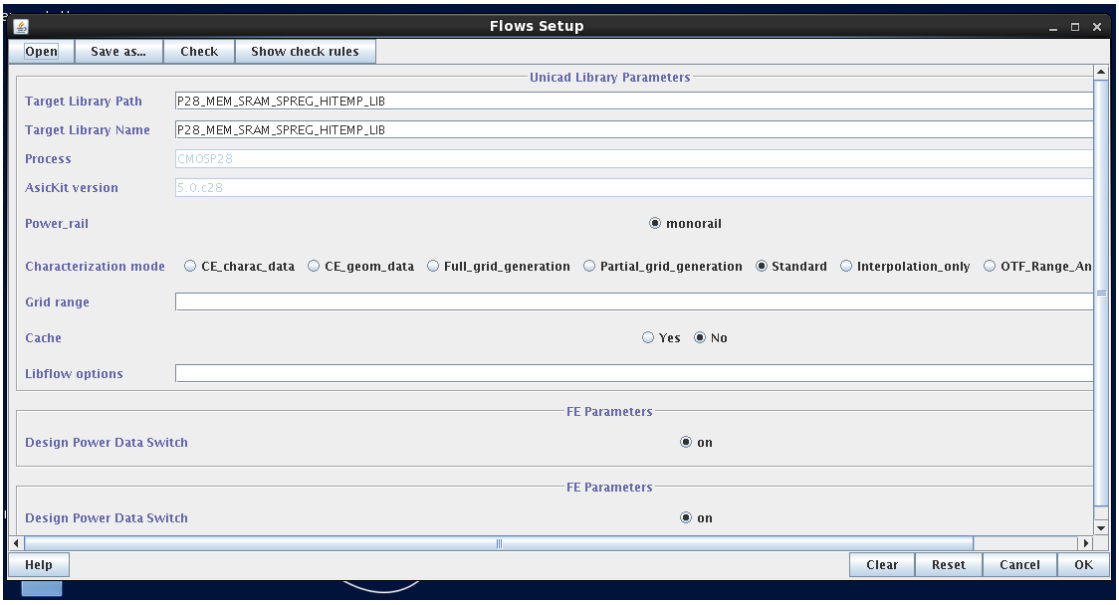

Figure 5.4: Flows Setup

- **Step 3**: Now we can select any operating conditions from following pop-up GUI, which we get after clicking on operating conditions, according to requirement.

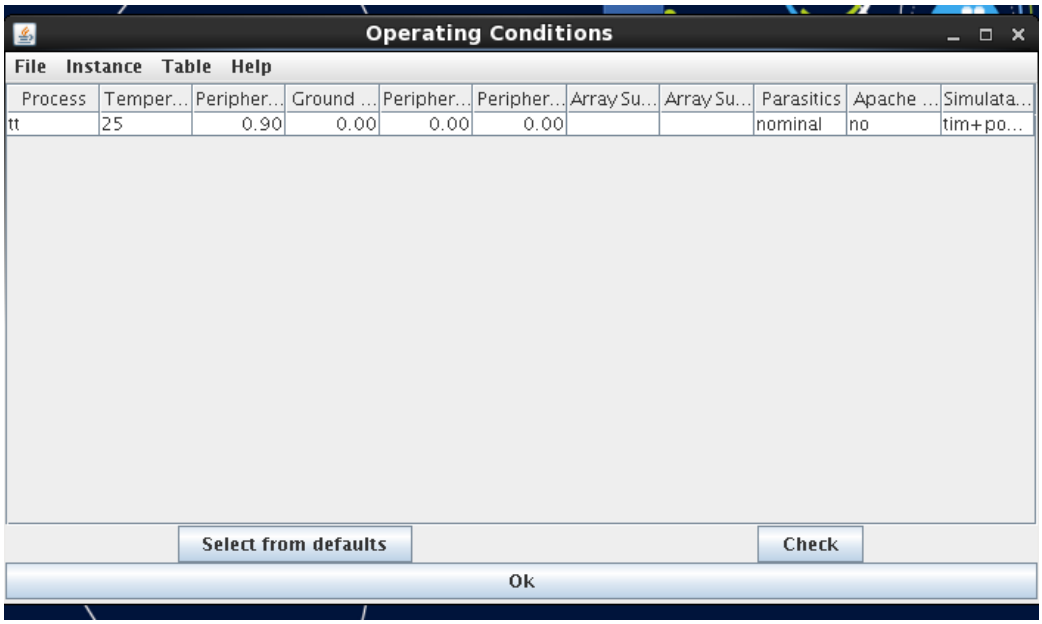

Figure 5.5: Operating Conditions

- **Step 4**: After selecting the operating condition now from the opening Generator Parameters user can open a pop-up and fill cut configurations as per his or her wish as show in figure.

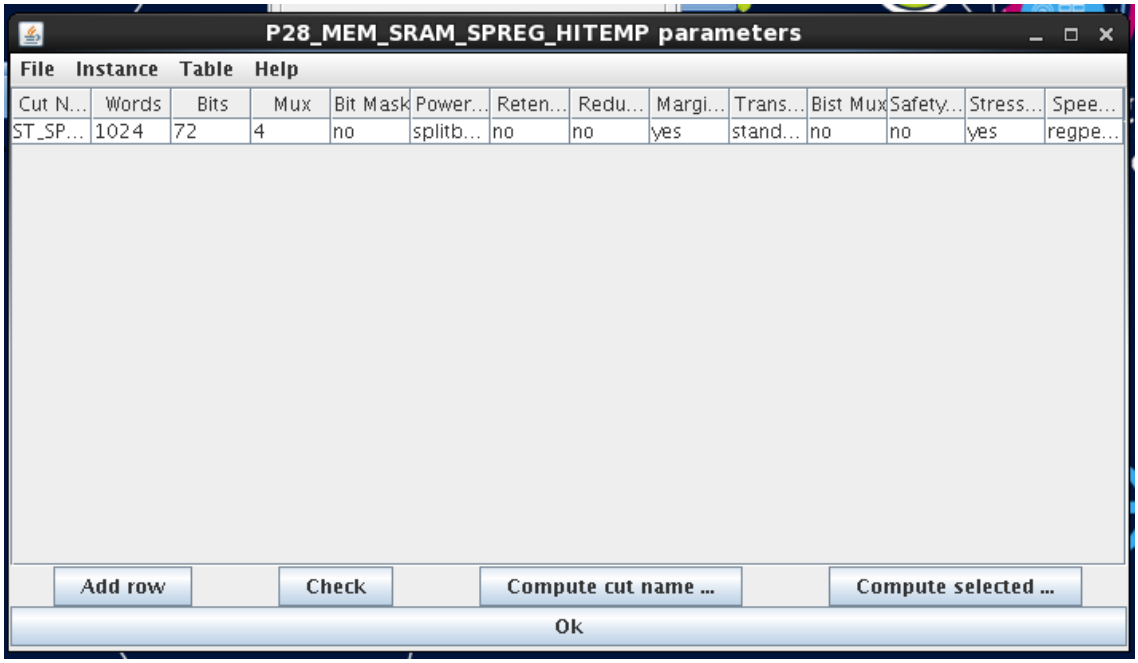

Figure 5.6: Generator Parameters (Cut Configuration)

Here we can select words, bits, mux and all other options based on this our memory will generate from our code.

Compute cut name will change name according to all the option selected. Add Row will allow you to add another cut.

- **Step 5**: In last step before starting execution for cut generation, user is provided for different views from which user can select view(s) for cut generation. After selection the view, he or she click on the ok Button then on the RUN button and the required view gets generated.

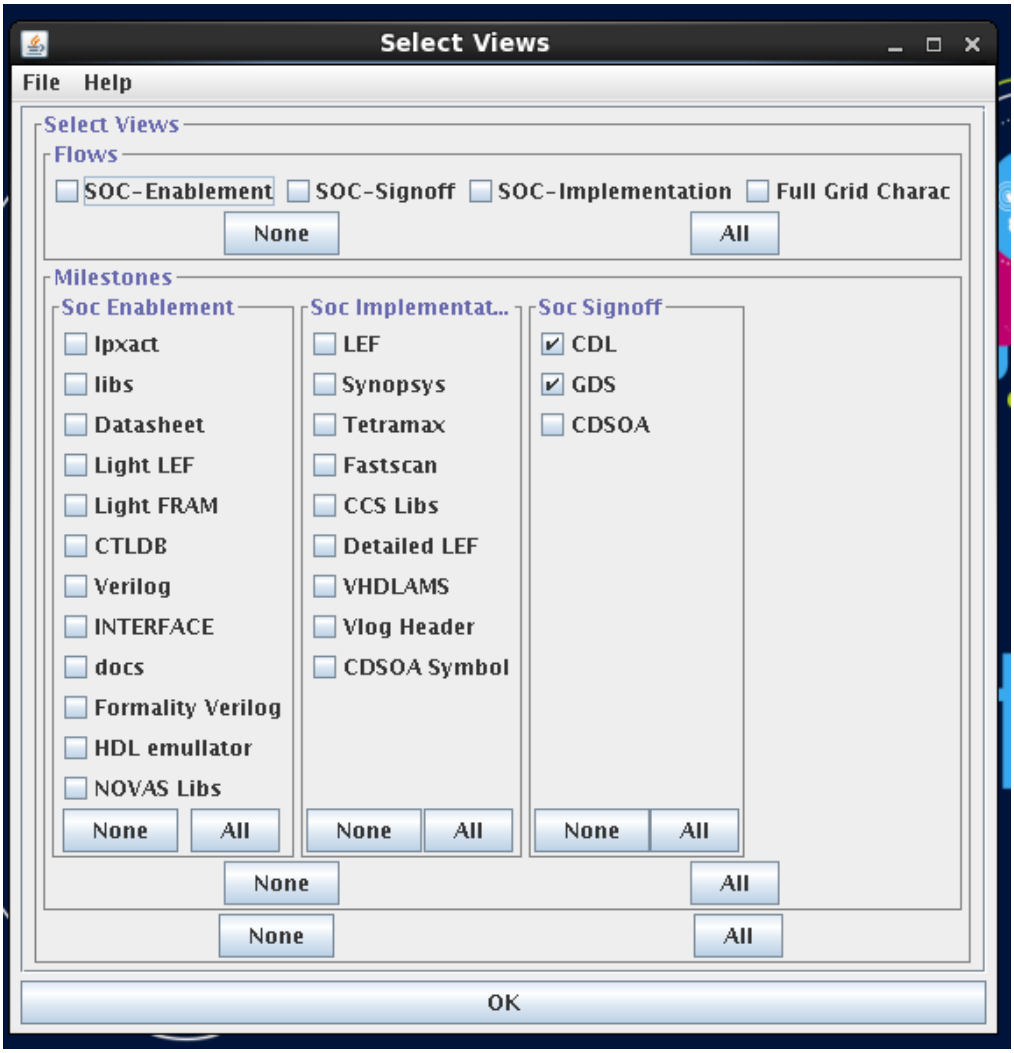

Figure 5.7: Select Views Window

Chapter 6

Conclusion and Future Scope

5.1 Conclusion

As we know quality time is more important in microelectronics. So it is require to make the thing automate and as possible as optimize. By making a unique framework, it gives platform to develop the compiler for all technology and architecture supporting combination of all configuration parameter. Moreover optimization in Layout code and input file saves quality time. Change in single parameter doesn't effect others thing. Hence it reduces the time and effort to develop one mature compiler.

Here we develop compiler for different memory. So for all memory we need to design compiler. But we can also develop a single code in which all memory will be developed, but again this will be very complex and need to be accurate. This will save more cost and efforts.

5.2 Future Scope

- **UniBe:** Here we develop compiler for different memory, so for all different memory we need to design compiler. But now we are also developing a single code in which all memory will be developed through it, but again this is very complex and need to be accurate. This will save more cost and efforts.
- **Checker:** It is one of the important module of this project as it check template files of a technology for invalid value, so that maximum number of errors can be which can occur during the run time can be traced and corrected. Checker is developed with keeping in mind that changes will keep on occurring in the templatefor supporting different technologies, thus a Checker config file is maintained for making changes in process flow according to the alterations in template files. Therefore, it is developed in such a manner that it take least efforts to maintain it.

References

[1] STMicroelectronics, “St internal documents on memories and training manuals,” pp. 314 – 317, October 2015.

[2] ST proprietary document for Product Making.

[3] STMicroelectronics, “St Documents on cut generation,” pp. 1 – 17, October 2015.

[4] STMicroelectronics, “St Tools,” pp. 1 – 8, October 2015.

[5] ST UniBE Document.

[6] http://www.vogella.com/tutorials/JavaRegularExpressions/article.html

[7] "Detailed description of GDSII (Graphic Data System) format[online]," Available: http://boolean.klaasholwerda.nl/interface/bnf/ gdsformat.html.

[8] Product-Virtuso http://www.cadence.com

[9] "MOS and BiCMOS circuit design process" Basic VLSI Design, Douglas A Pucknell, Kamran Eshraghian, 3rd Edition,Chapter 3., pp 60-70

[10] Ender Yılmaz, Günhan Dündar, “New Layout Generator For Analog CMOS Circuits”, 2007, IEEE.

[11] O.A.Marvik, “A Method for IC Layout Verification”, 2006, IEEE.

[12] P. Daglio, D. Iezzi, D. Rimondi, “Building the hierarchy from a flat netlist for a fast and accurate post-layout simulation with parasitic components”, 2004, IEEE.

[13] I. Issenin, E. Brockmeyer, B. Durinck, “Multiprocessor system-on-chip data reuse analysis for exploring customized memory hierarchies”, September 2006, IEEE.

[14] H.L. Lung, M. BrightSky, W. C. Chien, “Towards the integration of both ROM and RAM functions phase change memory cells on a single die for system-on-chip (SOC) applications”, September 2014, IEEE.

[15] G. Van Wauwe, E. Huyskens, “A cost effective way to test embedded RAM and ROM”, August 2002, IEEE.

[16] G. Martin, H. Chang, L. Cooke, A. de Oliveira, "Introduction to Platform-Based Design of Systems on Chip" in IP/SoC 2001 Tutorial, Santa Clara, California:, March 2001.

[17] G. Martin, H. Chang, “System-on-Chip design”, August 2002, IEEE.