Study of Simulation Based Software for Electrical Circuits

A PROJECT REPORT

Submitted in partial fulfillment of the
requirements for the award of the degrees

of
BACHELOR OF TECHNOLOGY
in
ELECTRICAL ENGINEERING

Submitted by:
RUPANK PAHUJA

Guided by:
Dr. Santosh Kumar Vishwakarma, Associate Professor
Nanoscale Devices, VLSI Circuit and System Design Research Group
Discipline of Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2018

Declaration of Authorship

I hereby declare that the project entitled "Study of Simulation Based Software for Electrical Circuits" submitted in partial fulfilment for the award of the degree of Bachelor of Technology in 'Electrical Engineering' completed under the supervision of Dr. Santosh Kumar Vishwakarma, Associate Professor, Electrical Engineering, IIT Indore is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Signed:

Rupank Pahuja

_____

Certificate

This is to certify that the B.Tech Project entitled, *"Study of Simulation based Software for Electrical Circuits"* and submitted by Rupank Pahuja in partial fulfillment of the requirements of B.Tech Project embodies the work done by them under my supervision.

Supervisor

Dr. SANTOSH KUMAR VISHWAKARMA
Associate Professor,
Indian Institute of Technology Indore

Acknowledgements

It is my privilege to express my gratitude to several persons who helped me directly or indirectly to conduct this project work. I express my heart full indebtedness to my BTP guide **Dr. Santosh Kumar Vishwarkma** for his sincere guidance and inspiration in completing this Project.

I also thank my friends who have more or less contributed to the making of this project.

This study has indeed helped me to explore more knowledgeable avenues related to this topic and I am sure it will help me in future.

INDIAN INSTITUTE OF TECHNOLOGY INDORE

Abstract

Department of Electrical Engineering

Bachelor of Technology

Study of Simulation based Software for Electrical Circuits

This project studies the fundamentals of SPICE based simulation software for electrical circuits. It was done as a part of six months internship at Mentor Graphics, an EDA Tool Design Comapany. A detailed study of core algorithm involved in SPICE based software such as one on which I worked AFS(Analog FastSPICE Simulator) is included in the report. Description of other tasks such as Testing, Verification and Automation which are an important part of delivering products to client is also included in the report.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Electronic Circuit Simulation

Electronic circuit simulation uses mathematical models to replicate the behavior of an actual electronic device or circuit. Simulation software allows for modeling of circuit operation and is an invaluable analysis tool. Due to its highly accurate modeling capability, many colleges and universities use this type of software for the teaching of electronics technician and electronics engineering programs. Simulating a circuit's behavior before actually building it can greatly improve design efficiency by making faulty designs known as such, and providing insight into the behavior of electronics circuit designs. In particular, for integrated circuits, the tooling is expensive, breadboards are impractical, and probing the behavior of internal signals is extremely difficult. Therefore, almost all IC design relies heavily on simulation.

The most well known analog simulator is SPICE. Probably the best known digital simulators are those based on Verilog and VHDL. Some electronics simulators integrate a schematic editor, a simulation engine, and on-screen waveform display, allowing designers to rapidly modify a simulated circuit and see what effect the changes have on the output. They also typically contain extensive model and device libraries. These models typically include IC specific transistor models such as BSIM, generic components such as resistors, capacitors, inductors and transformers, user defined models (such as controlled current and voltage sources, or models in Verilog-A or VHDL-AMS). Printed circuit board (PCB) design requires specific models as well, such as transmission lines for the traces and IBIS models for driving and receiving electronics.

1.2 About types of Simulation Based Software

While there are strictly analog electronics circuit simulators, popular simulators often include both analog and event-driven digital simulation capabilities, and are known as

mixed-mode simulators. This means that any simulation may contain components that are analog, event driven (digital or sampled-data), or a combination of both. An entire mixed signal analysis can be driven from one integrated schematic. All the digital models in mixed-mode simulators provide accurate specification of propagation time and rise/fall time delays.

Mixed-mode simulation is handled on three levels:

- With primitive digital elements that use timing models and the built-in 12 or 16 state digital logic simulator.
- With sub-circuit models that use the actual transistor topology of the integrated circuit.
- With In-line Boolean logic expressions.

1.3 About the company and Product

- My internship started at Mentor Graphics on 15th May 2018 and was a six months long internship. Mentor Graphics is EDA (Electronic Design Automation) multinational corporation for Electrical Engineering and Electronics. Calibre is the flagship tool by mentor well-known for its accuracy among the industry giants in the same field Apart from Calibre, simulation tools like, AFS(Analog FastSPICE), Questa-Sim, Model-Sim are reputed tools and widely used by Industry and Academia as well. Mentor Graphics acquired many different EDA companies with applications in different field which diversified its market and customer-base. Finally Mentor Graphics was itself acquired in April 2017 by Siemens and styled it as "Mentor, a Siemens Business".
- “Mentor, a Siemens Business” then “Mentor Graphics” acquired BDA (Berkeley Design Automation) an EDA company with expertise in nanometer analog, mixed-signal, and RF circuit verification in 2014. With that accusation also came in AFS(Analog FastSPICE) which is a world’s fastest nanometer circuit verification platform for analog, RF, mixed-signal, and custom digital circuits. One of the biggest competitors in AMS(Analog/Mixed Signal) automation is Spectre by Cadence.

Chapter 2

SPICE and Netlists

SPICE (Simulation Program with Integrated Circuit Emphasis) is an open source circuit simulation software tool that was developed in the early 1970's at the University of California, Berkeley. SPICE is intended to simulate the behavior of signal generators, measurement equipment (e.g. multimeters, oscilloscopes, etc.), passive elements (e.g. resistors, capacitors, inductors), and active devices (e.g. diodes, transistors, etc.) in a circuit to give designers a cost-effective means of confirming a circuit's intended operation before investing time and money into physically fabricating the circuit. Although there are other computer-aided circuit simulators in the market, SPICE is the most widely used and has become the industry standard on which most commercial circuit simulators are based.

When it was first developed, SPICE was limited to mainframe computers due to its processor intensive calculations. However, with the advancement of personal computer systems in the 1980's, many versions of SPICE became available that allow for SPICE circuit simulations on an average desktop computer. In this research, a commercial version of SPICE from Synopsys®, called HSpice, was used for all circuit simulations because of its high reputation for accurate simulations and ability to incorporate custom models of power devices though the Verilog-A modeling interface. SPICE reads in circuit descriptions, analysis descriptions, and output requests from a text file called a netlist. The netlist describes a circuit by listing each component with its respective value, connection nodes, and any additional required parameters. For example:

$$\begin{array}{llll} \mathcal{V}1 & 0001 & 0 & 10V \\ \mathcal{R}1 & 0001 & 2 & 10kOhm \\ \mathcal{C}1 & 0002 & 0 & 50uF \end{array}$$

describes a 10 V DC source connected across a 10 kOhm resistor in series with a 50 uF capacitor where 0001 and 0002 are connection nodes and 0 is the circuit ground node

2.1 Understanding SPICE

SPICE has the ability to simulate components ranging from the most basic passive elements such as resistors and capacitors to sophisticated semiconductor devices such as MESFETs and MOSFETs. Using these intrinsic components as the basic building blocks for larger models, designers and chip manufacturers have been able to define a truly vast and diverse number of SPICE models. Most commercially available simulators include more than 15,000 different components.

The quality of SPICE models can vary, and not all SPICE models are applicable to every application. It is important to consider this when using the models supplied with a SPICE simulation package. Using a SPICE model inappropriately can lead to inaccurate results, or even generate an error in some circumstances. One of the most common errors made by even seasoned engineers is confusing a SPICE model with a PSPICE model. PSPICE is a commercially available program that uses proprietary languages to define components and models.

A circuit must be presented to SPICE in the form of a netlist. The netlist is a text description of all circuit elements such as transistors and capacitors, and their corresponding connections. Modern schematic capture and simulation tools such as Multisim allow users to draw circuit schematics in a user-friendly environment, and automatically translate the circuit diagrams into netlists. Consider as an example the simple voltage divider circuit below. We include both netlist and corresponding circuit schematic.

Voltage Divider Netlist

```
* Any text after the asterisk '*' is ignored by SPICE

* Voltage Divider
```

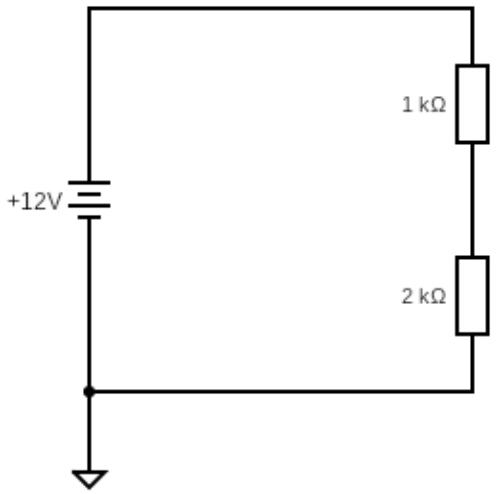
$$\begin{array}{llll} \mathcal{V}1 & 1 & 0 & 12 \\ \mathcal{R}1 & 1 & 2 & 1000 \\ \mathcal{R}2 & 2 & 0 & 2000 \end{array}$$

```
.OP * perform a DC operating point analysis
```

```
.END
```

Voltage Divider Schematic

FIGURE 2.1: Voltage divider circuit



2.2 Trade off Between Speed and Accuracy

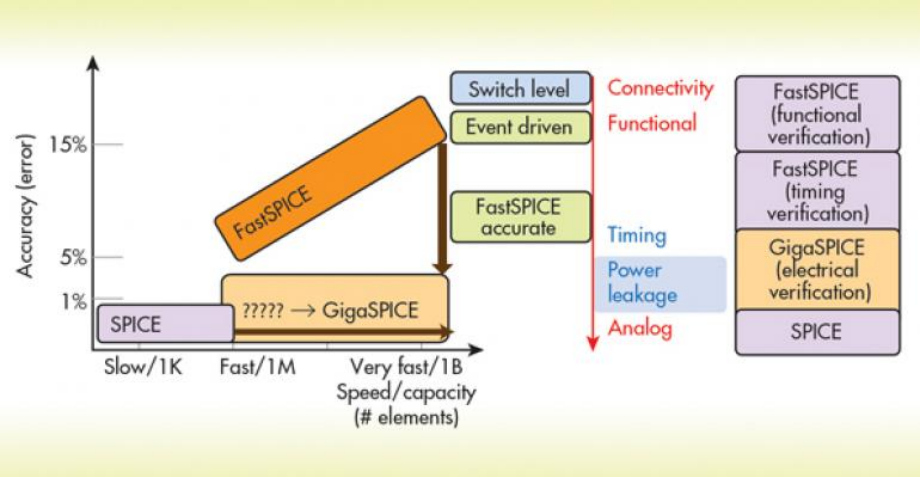
Although the SPICE models used in a SPICE simulation can greatly affect the accuracy of the results, simulation settings also contribute to varying degrees of accuracy. SPICE simulation options generally allow the user to gain more accuracy in the results at the cost of the speed of the simulation.

To understand the trade off between speed and accuracy in SPICE simulation one must consider a number of factors. SPICE simulation was created over 30 years go and around that time a typical computer had less power than the average microwave oven did thirty years later. Computing power was very expensive. The simulation of a circuit to the highest degree of accuracy could have taken longer and cost more money than building the actual circuit to see the results. Also, consider that the broad purpose of circuit simulation is to augment basic hand calculations and predict general circuit behavior. With these considerations in mind, the designers of SPICE created a program

that could produce reasonably accurate results in a cost-effective manner. They also included many options to allow engineers to customize the accuracy of a simulation.

As computing power has increased exponentially over the years, so have the complexity of circuit designs being simulated. Speed and accuracy are still important factors to consider when simulating circuits.

FIGURE 2.2: Speed Accuracy Comparison)



Chapter 3

DC Analysis

3.1 Modified Nodal Analysis

Many different kinds of network element are encountered in network analysis. For circuit analysis it is necessary to formulate equations for circuits containing as many different types of network elements as possible. There are various methods for equation formulation for a circuit. These are based on three types of equations found in circuit theory.

- equations based on Kirchhoff's voltage law (KVL)
- equations based on Kirchhoff's current law (KCL)
- branch constitutive equations

The equations have to be formulated (represented in a computer program) automatically in a simple, comprehensive manner. Once formulated, the system of equations has to be solved. There are two main aspects to be considered when choosing algorithms for this purpose: accuracy and speed. The MNA, briefly for Modified Nodal Analysis, has been proved to accomplish these tasks. MNA applied to a circuit with passive elements, independent current and voltage sources and active elements results in a matrix equation of the form:

$$\left[\ A\ \right] . \left[x\right] = \left[\ z\ \right]$$

For a circuit with N nodes and M independent voltage sources:

- The A matrix
  - is (N+M)(N+M) in size, and consists only of known quantities
  - the NN part of the matrix in the upper left:

    - has only passive elements
    - elements connected to ground appear only on the diagonal
    - elements not connected to ground are both on the diagonal and off-diagonal terms
  - the rest of the A matrix (not included in the NN upper left part) contains only 1, -1 and 0 (other values are possible if there are dependent current and voltage sources)

- The x matrix
  - is an (N+M)1 vector that holds the unknown quantities (node voltages and the currents through the independent voltage sources)
  - the top N elements are the n node voltages
  - the bottom M elements represent the currents through the M independent voltage sources in the circuit

- The z matrix
  - is an (N+M)1 vector that holds only known quantities
  - the top N elements are either zero or the sum and difference of independent current sources in the circuit
  - the bottom M elements represent the M independent voltage sources in the circuit

The circuit is solved by a simple matrix manipulation:

$$\left[\ x\ \right] = \left[A\right]^{-1} . \left[\ z\ \right]$$

3.1.1 Producing the MNA matrix

This describes the algorithmic approach to the Modified Nodal Analysis. The three matrix that we need to generate are the A matrix, the x matrix and the z matrix. They will be created by combining various different matrices.

3.1.2 The A Matrix

The A matrix will be developed as the combination of 4 smaller matrices, G, B, C, and D.

$$\left[\ A\ \right] = \begin{bmatrix} G & B \\ C & D \end{bmatrix}$$

The A matrix is (M+N)(M+N) (N is the number of nodes, and M is the number of independent voltage sources) and:

- the G matrix is NN and is determined by the interconnections between the circuit elements
- the B matrix is NM and is determined by the connection of the voltage sources
- the C matrix is MN and is determined by the connection of the voltage sources (B and C are closely related, particularly when only independent sources are considered)
- the D matrix is MM and is zero if only independent sources are considered

Rules for making the G matrix

The G matrix is an NxN matrix formed in two steps.

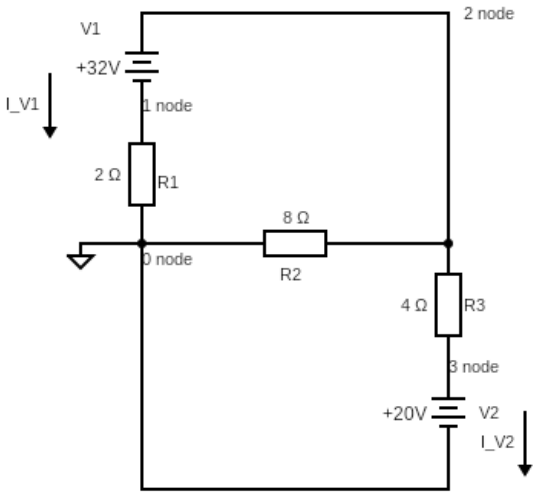
1. Elements in diagonal matrix equals to the sum of the conductance of each element connected to corresponding node.
2. The off diagonal elements are the negative conductance of the element connected to the pair of corresponding node.

If an element is grounded, it will only have contribute to one entry in the G matrix – at the appropriate location on the diagonal. If it is not grounded it will contribute to four entries in the matrix – two diagonal entries (corresponding to the two nodes) and two off-diagonal entries.

$$\left[\ G\ \right] = \begin{bmatrix} \frac{1}{R1} & 0 & 0 \\ 0 & \frac{1}{R2} + \frac{1}{R3} & -\frac{1}{R2} \\ 0 & -\frac{1}{R2} & \frac{1}{R2} \end{bmatrix}$$

$$\left[\ G\ \right] = \begin{bmatrix} \frac{1}{R1} + \frac{1}{R2} & -\frac{1}{R2} \\ -\frac{1}{R2} & \frac{1}{R2} + \frac{1}{R3} \end{bmatrix}$$

FIGURE 3.1: Example 1 of making a G Matrix



Rules for making the B matrix

The B matrix is an NM matrix with only 0, 1 and -1 elements. Each location in the matrix corresponds to a particular voltage source (first dimension) or a node (second dimension). If the positive terminal of the ith voltage source is connected to node k, then the element (k,i) in the B matrix is a 1. If the negative terminal of the ith voltage source is connected to node k, then the element (k,i) in the B matrix is a -1. Otherwise, elements of the B matrix are zero.

If a voltage source is ungrounded, it will have two elements in the B matrix (a 1 and a -1 in the same column). If it is grounded it will only have one element in the matrix.

Case 1:

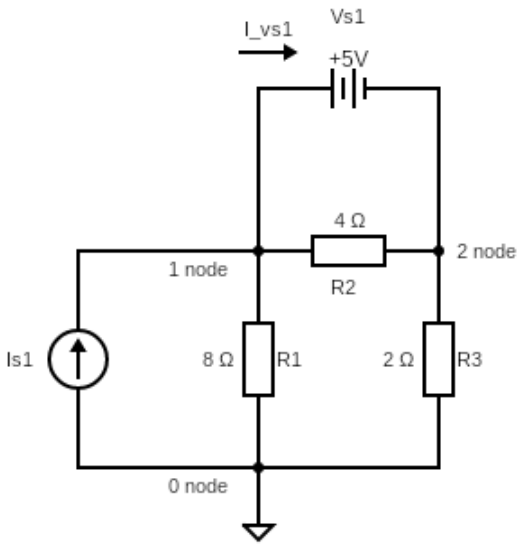
$$\left[\ B\ \right] = \begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Case 2:

$$\left[\ B\ \right] = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Rules for making C matrix The C matrix is an MN matrix with only 0, 1 and -1 elements. Each location in the matrix corresponds to a particular node (first dimension) or voltage source (second dimension). If the positive terminal of the ith voltage source is

FIGURE 3.2: Example 2 of making a G Matrix



connected to node k, then the element (i,k) in the C matrix is a 1. If the negative terminal of the ith voltage source is connected to node k, then the element (i,k) in the C matrix is a -1. Otherwise, elements of the C matrix are zero. In other words, the C matrix is the transpose of the B matrix. This is not the case when dependent sources are present.

Rules for making the D matrix The D matrix is an MM matrix that is composed entirely of zeros. It can be non-zero if dependent sources are considered.

3.1.3 The x matrix

The x matrix holds our unknown quantities and will be developed as the combination of 2 smaller matrices v and j. It is considerably easier to define than the A matrix.

$$\begin{bmatrix} x \end{bmatrix} = \begin{bmatrix} v \\ j \end{bmatrix}$$

The x matrix is (NxM)x1 (N is the number of nodes, and M is the number of independent voltage sources) and:

- the v matrix is Nx1 and hold the unknown voltages
- the j matrix is Mx1 and holds the unknown currents through the voltage sources

Rules for making the v matrix The v matrix is an 1N matrix formed of the node voltages. Each element in v corresponds to the voltage at the equivalent node in the

circuit (there is no entry for ground – node 0). For a circuit with N nodes we get:

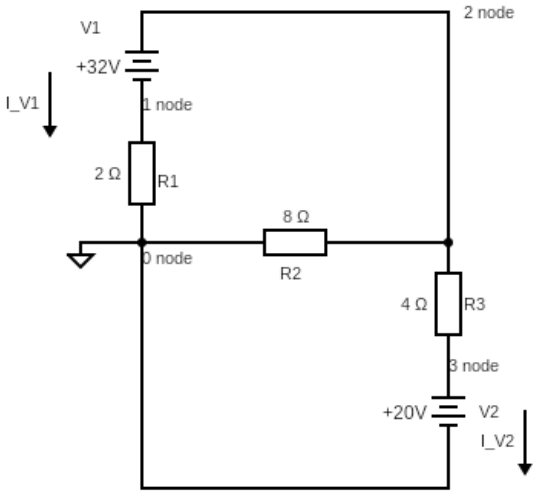
$$\left[v\right] = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_N \end{bmatrix}$$

Rules for making the j matrix The j matrix is an 1M matrix, with one entry for the current through each voltage source. So if there are M voltage sources V1, V2 through VM, the j matrix will be:

$$\left[j\right] = \begin{bmatrix} iv_1 \\ iv_2 \\ \dots \\ iv_M \end{bmatrix}$$

Case1:

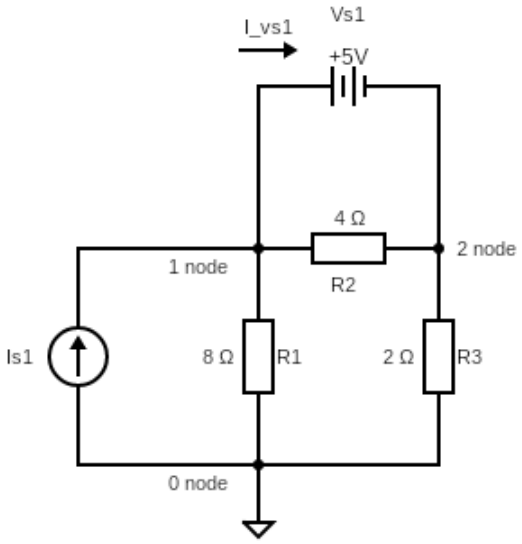
FIGURE 3.3: Example 1 making a x Matrix



$$\left[v\right] = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}, \left[j\right] = \begin{bmatrix} iv_1 \\ iv_2 \end{bmatrix}, \left[x\right] = \begin{bmatrix} v \\ j \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ i_v1 \\ i_v2 \end{bmatrix}$$

Case2:

FIGURE 3.4: Example 2 making a x Matrix



$$\left[v\right] = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}, \left[j\right] = \left[iv_1\right], \left[x\right] = \begin{bmatrix} v \\ j \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ i_v 1 \end{bmatrix}$$

3.1.4 The Z matrix

The **z** matrix holds our independent voltage and current sources and will be developed as a combination of 2 smaller matrices i and e. It is quite easy to formulate.

$$\left[z\right] = \begin{bmatrix} i \\ e \end{bmatrix}$$

The z matrix is 1(M+N) (N is the number of nodes, and M is the number of independent voltage sources) and:

- the i matrix is 1N and contains the sum of the currents through the passive elements into the corresponding node (either zero, or the sum of independent current sources)
- the e matrix is 1M and holds the values of the independent voltage sources

Rules for making i matrix

The i matrix is an Nx1 matrix with each element of the matrix corresponding to a particular node. The value of each element of i is determined by the sum of current sources into the corresponding node. If there are no current sources connected to the node, the value is zero.

Rules for making e matrix

The e matrix is Mx1 matrix with each element of the matrix equal in value to the corresponding independent voltage source.

Case 1:

$$\left[i\right] = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \left[e\right] = \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}, \left[z\right] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ v_1 \\ v_2 \end{bmatrix}$$

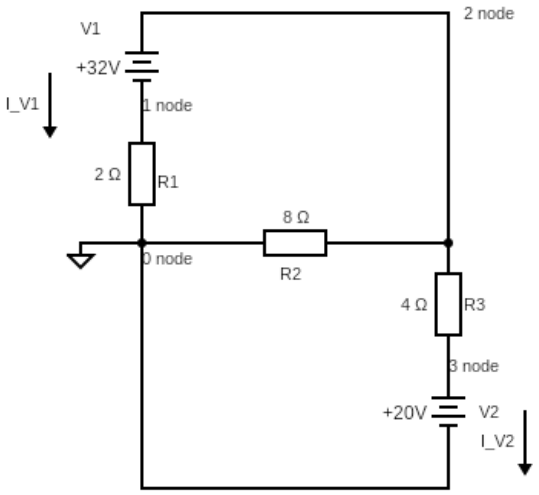
Case 2:

$$\left[i\right] = \begin{bmatrix} Is1 \\ 0 \end{bmatrix}, \left[e\right] = \left[Vs1\right], \left[z\right] = \begin{bmatrix} Is1 \\ 0 \\ Vs1 \end{bmatrix}$$

3.1.5 Putting it together

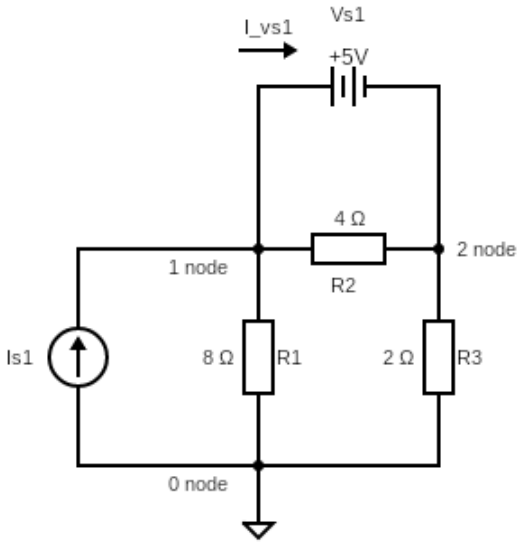
We can write out the full matrix solution for both cases that we will be developing.

FIGURE 3.5: Example 1 making a complete Matrix



$$\begin{bmatrix} \frac{1}{R_1} & 0 & 0 & -1 & 0 \\ 0 & \frac{1}{R_2}+\frac{1}{R_3} & -\frac{1}{R_2} & 1 & 0 \\ 0 & -\frac{1}{R_2} & \frac{1}{R_2} & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} . \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ i_V 1 \\ i_V 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ V1 \\ V2 \end{bmatrix}$$

FIGURE 3.6: Example 2 making a complete Matrix



$$\begin{bmatrix} \frac{1}{R_1}+\frac{1}{R_2} & -\frac{1}{R_2} & 1 \\ -\frac{1}{R_2} & \frac{1}{R_2}+\frac{1}{R_3} & -1 \\ 1 & -1 & 0 \end{bmatrix} . \begin{bmatrix} v_1 \\ v_2 \\ i_V s1 \end{bmatrix} = \begin{bmatrix} Is1 \\ 0 \\ Vs1 \end{bmatrix}$$

3.2 Verification of Modified Nodal Analysis

Expanding the matrix representation above to a set of equations denotes the following equation system consisting of 3 of them.

Case 1:

$$v_1 . \frac{1}{R_1} - iv1 = 0$$

$$(\frac{1}{R_2}+\frac{1}{R_3}).v2 + (-\frac{1}{R_2}).v3 + iv1 = 0$$

$$(-\frac{1}{R_2}).v2 + \frac{1}{R_2}.v3 + iv2 = 0$$

The above three equations are KVL equations on the case 1 and the rest two equations will just affirm

$$0 = 0$$

which has infinite solutions. We will get the solution to variables from above equation.

Case 2:

$$(\frac{1}{R_1} + \frac{1}{R_2}).v1 (-\frac{1}{R_2}).v2 + ivs1 = Is1$$

$$(-\frac{1}{R_2}).v1 + (\frac{1}{R_2} + \frac{1}{R_3}).v2 + ivs1 = 0$$

$$v1 - v2 = Vs1$$

The first two equations are KVL and the last equation is a equation which affirms the situation present in the circuit. Hence MNA is verified.

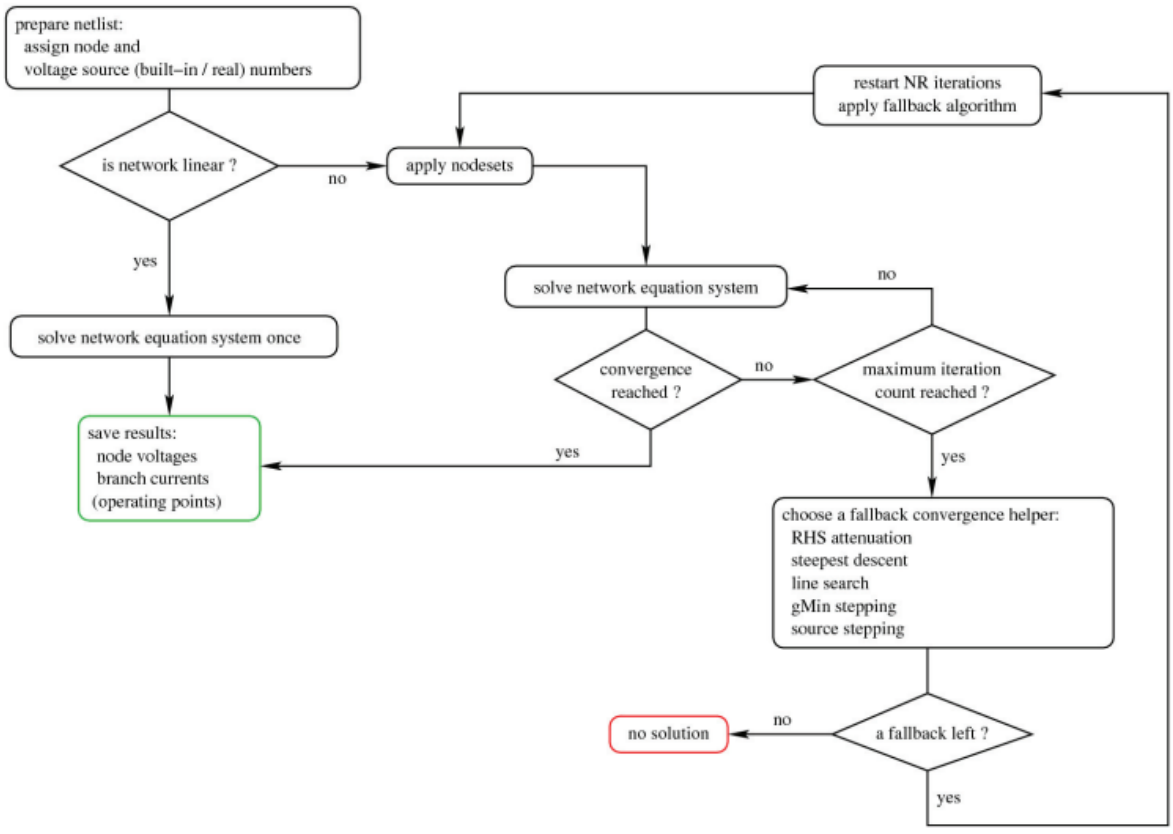
Chapter 4

Flow of SPICE based software

4.1 Overall solution algorithm for DC Analysis

In this section an overall solution algorithm for a DC analysis for linear as well as non-linear networks is given. With non-linear network elements at hand the Newton-Raphson (NR) algorithm is applied.

FIGURE 4.1: DC solution Algorithm flow chart



The algorithm shown in fig. 4.1 has been proved to be able to find DC solutions for a large variety of networks. It must be said that the application of any of the fall-back convergence helpers indicates a nearly or definitely singular equation system (e.g.

floating nodes or over determining sources). The convergence problems are either due to an apparently “wrong” network topology or to the model implementation of nonlinear components.

In some cases it may even occur that tiny numerical inaccuracies lead to non-convergences whereas the choice of a more accurate (but probably slower) equation system solver can help. With network topologies having more than a single stable solution (e.g. bi-stable flip-flops) it is recommended to apply node sets, i.e. forcing the Newton-Raphson iteration into a certain direction by initial values.

When having problems to get a circuit have its DC solution the following actions can be taken to solve these problems.

- check circuit topology (e.g. floating nodes or over determining sources)
- check model parameters of non-linear components
- apply nodesets
- choose a more accurate equation system solver
- relax the convergence tolerances if possible
- increase the maximum iteration count
- choose the prefered fallback algorithm

Chapter 5

Other Contribution

5.1 Some other software related work

There were other enabling tasks which I performed there at my time in the company.The first part included me getting acquainted with the software and how its managed here. I learned the basic code of conduct for contributing you code in the main repositories. In the second part I built upon the knowledge gained by me in first part and I tried to find runtime-errors and memory leaks in various patches which were contributing to the master branch and degrading the performance. The other part of internship was a completely independent project which aimed to compress a regressoion repository.

- This was my first introduction to a variant of SPICE simulation software. We had the experience to use Spectre by Cadence which is also a simulation software during the 6th semester to design 8-bit multiplier. The fact that I had already used the software as a designer, it gave me a great perspective of what I was getting into while working on it as a developer.

- After gaining the enough and required knowledge to start working, I was assigned the task to automate the process of finding patches which had runtime-errors and memory leaks in them. Some background- The soft wares to which developers commit code from different time zones all the time it's were hard to check any commit is affecting a different commit before checking in the code. During the maintenance of such large code-base it's necessary that code is run every day and all the the test-cases are run to identify if some developer's code is causing some problem. With every commit, GIT generates a hash which helps identify every change done by developer to the code-base. I was given a set of test-cases and the date they started failing, my task was to point to a particular patch which might have caused the problem, additionally the task entailed me to make a bash script which automates the given process and optimize it.

  To find the particular patch I first tried to nail down the date around which the

patch might exist for which I used the daily-builds which were already present on the servers. The script takes test-case file, start date, end-date as input and collects the data of all the runs which gives the approximate idea of where the faulty patch might be hiding. This task that I performed was very critical for the development as well Q/A testing team.

After figuring out the dates, next step is to checkout the patches between those dates and find the one which takes the most time.

- Some background **Regression testing** is re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change. Changes that may require regression testing include bug fixes, software enhancements, configuration changes, and even substitution of electronic component.As regression test suites tend to grow with each found defect, test automation is frequently involved. Sometimes a change impact analysis is performed to determine an appropriate subset of tests. Now considering the fact that for each bug-fixed there is test-case which has to be added to regression testing repository to check if that fail is not happening again the size of regression repository takes up huge space. One of the reason contributing to this increased size is the duplicate include library files for various test cases. The library files are the files provided by **foundry companies** which specifies how the electrical circuit components behave(the ones which we simulating) under different conditions referred as different corners in industry language. The task at my hand was to parse all the test-cases and find the include files, see if any of them are duplicates and keep the single copy of unique file and remove the other files. With this I also had to soft-link all the test-cases to the library files, now present in the new location. Other than these three important works I also helped configure code-review environment which potentially could make the process a lot efficient.

Chapter 6

Future Work

I can definitely expand on the two parts of the internship to make the work more versatile and add more features for developers to use it. The second part of the project which includes finding the guilty patch, the steps which I follow is first find out the probable days between which the faulty patch might exist and then checkout and compile all the patches between those dates. Instead of doing the task manually I aim to automate the script to find the probable dates between which the failure might have occur ed automatically without human intervention.

The third part of the project is coded very specifically to alter particular type of test files which includes particular type of lib files. The project can be extended to include all the test cases and all the different types of library files included in them. One modification which can be done easily is to take all these information about the test files and library files as inputs by the user user to make the project more dynamic and well-equipped.