Improving an Analog Circuit Simulator based on Homotopy Methods

Jatin Vikram Singh
Indian Institute of Technology, Kanpur, India
jatinvs@iitk.ac.in

Faculty Advisor: Prof. Ljiljana Trajkovic
School of Engineering Science
Simon Fraser University, Burnaby
British Columbia, Canada

1 Introduction

DC operating points, also known as the bias points or the quiescent points, are the values of the voltages and currents established in the circuit with DC sources. Devices, such as transistors, behave differently when biased with DC or AC sources. They are usually combined with resistors that ensure their proper functioning. The values of currents and voltages determined with DC voltage sources, under a certain configuration of resistors and other components, are the operating points. The bias point depends on the values of resistances and other components used along with the transistors. To find the DC operating points of a transistor circuit, we need to solve a system of nonlinear algebraic equations that describe the DC behavior of the circuit.

A common method used to find DC operating points is the Newton-Raphson algorithm. It requires an initial point close enough to the solution, which is sometimes difficult to provide. Therefore alternative methods to solve nonlinear system of equations, such as homotopies, are often applied to find the DC operating points of various circuits. In this project, we have used a software implementation of homotopy algorithm. It is composed of two programs: a parser, developed by Edward Chan [2] improved by Andrea Dyess [3], and then recently further improved by Joao Eric Melo [5]. It generates a system of equations derived from a circuit description file. A Matlab script written by Heath Hoffman [4] implements the homotopy method to solve the system of equations. The output generated by the parser was incomplete and needed modifications that were in past done manually. The main goal of the project is to identify and correct the problems.

2 Homotopy Methods

Homotopy methods [1], [3], also called parameter embedding or continuation methods, involve the introduction of a new parameter λ into the system of nonlinear equations. Consider the system of nodal or modified nodal equations:

$$\mathcal{F}(x) = 0,$$

where the vector x represents the unknown node voltages and/or currents. *F(x)* is a system of n equations with n unknowns. Hence:

$$\mathcal{F} : \mathcal{R}^n \rightarrow \mathcal{R}^n.$$

We then create another function:

$$\mathcal{H} : \mathcal{R}^{n+1} \rightarrow \mathcal{R}^n$$

by introducing an additional parameter. The goal is to choose an appropriate homotopy equation

$$\mathcal{H}(x, \lambda) = 0$$

so that the solution to $\mathcal{F}(x) = 0$ can be derived from the solution of homotopy equation. We used the following homotopy mapping:

$$\mathcal{H}(x, \lambda) = (1 - \lambda)(x - a) + \lambda\mathcal{F}(x).$$

This function is chosen so that:

$$\mathcal{H}(x, 0) = (x - a) \ , \ \mathcal{H}(x, 1) = \mathcal{F}(x).$$

The values of λ form a path with the starting point **a**. The solution(s) are the point(s) where the path crosses $\lambda = 1$.

3 The Parser

Implementation of the homotopy methods requires that the set of equations that describe the circuit be specified. For some circuits, these equations can be written by hand. However, this is not possible for more complicated circuits. A C++ program written by Edward Chan [2] generates nodal equations or modified nodal equations for the circuit to be solved. The parser takes the SPICE netlist file as the input that describes the elements of the circuit and how they are connected. It then generates the circuit equations for either nodal or modified nodal analysis and their Jacobians.

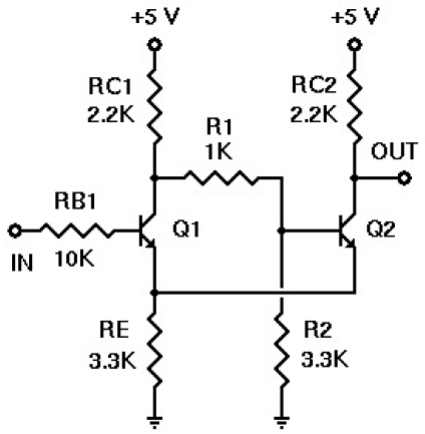

Figure 1: Schmidt trigger: Spice circuit file.

```
Rc1   1 2 2.2K
R1    2 3 1K
Rc2   1 4 2.2K
Q1    2 5 6 Q2N2222A
Q2    4 3 6 Q2N2222A
Vin   5 0 5.0
RE    6 0 3.3K
R2    3 0 3.3K

.model Q2N2222A NPN BF=150 IS=1E-16 BR=7.5
```

Figure 2: Schmidt trigger: Spice netlist file.

4 Modified Nodal Equations

Although the node voltage and loop current method are the most widely used, modified nodal analysis (MNA) is another powerful method. MNA often results in larger systems of equations than the other methods. However, it is easier to be implemented, which is a considerable advantage for automated solution. To use modified nodal analysis, one equation for each node not attached to a voltage source (as in standard nodal analysis) is written. These equations are augmented with an equation corresponding to each voltage source. In the figure 3, the first six equations are standard nodal analysis equations. The last two are the current equations for the sources that are not connected to a resistor. This is done to balance the number of unknowns with the number of equations.

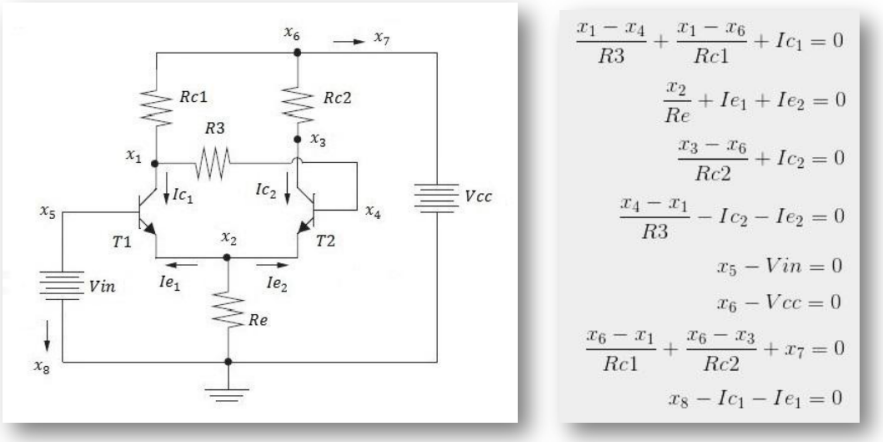

Figure 3: Schmidt trigger circuit: MNA equations.

5 Finding DC Operating Points

The output from the Parser consists of the set of equations and Jacobians required by the Matlab code to employ the homotopy method. The Matlab script then calculates all operating points using iterative techniques. The advantage of using homotopy methods is that the output contains all possible operating points as compared to the SPICE output that returns only one result depending on the initial point provided.

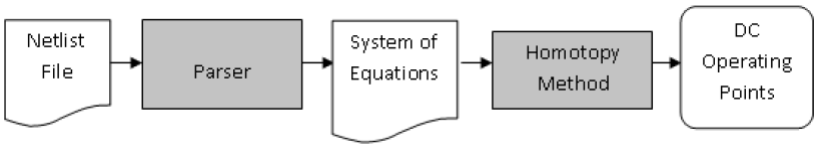

Figure 4: Algorithmic flow for DC operating point calculation [5].

6 Platform Change

The platform for the code has been changed from the command prompt to Visual Studio. The version used in this project is Microsoft Visual Studio Ultimate 2013. This software is freely available on the official website of Microsoft [6]. Visual Studio offers many benefits in terms of user interface. Debugging is easier by using the code flow charts and by tracking the local variables. This has been further elaborated in the Section 7.

7 Parser Modifications

The parser, developed by Edward Chang and modified by Joao Eric Melo, was difficult to debug because of the platform used, command prompt, and notepad++ editor. Thus, the first step was to identify a better alternative. The code was ported to Microsoft Visual Studio Ultimate 2013. The options of using code flow maps facilitated and breakpoints the debugging process. The programmer has the choice of seeing all the local variables in a separate window and monitoring how the variables change. The changes are identified by change in colour by Visual Studio.

The output files for previous version and the new version of the Parser have been posted on this web page for references and further modifications.

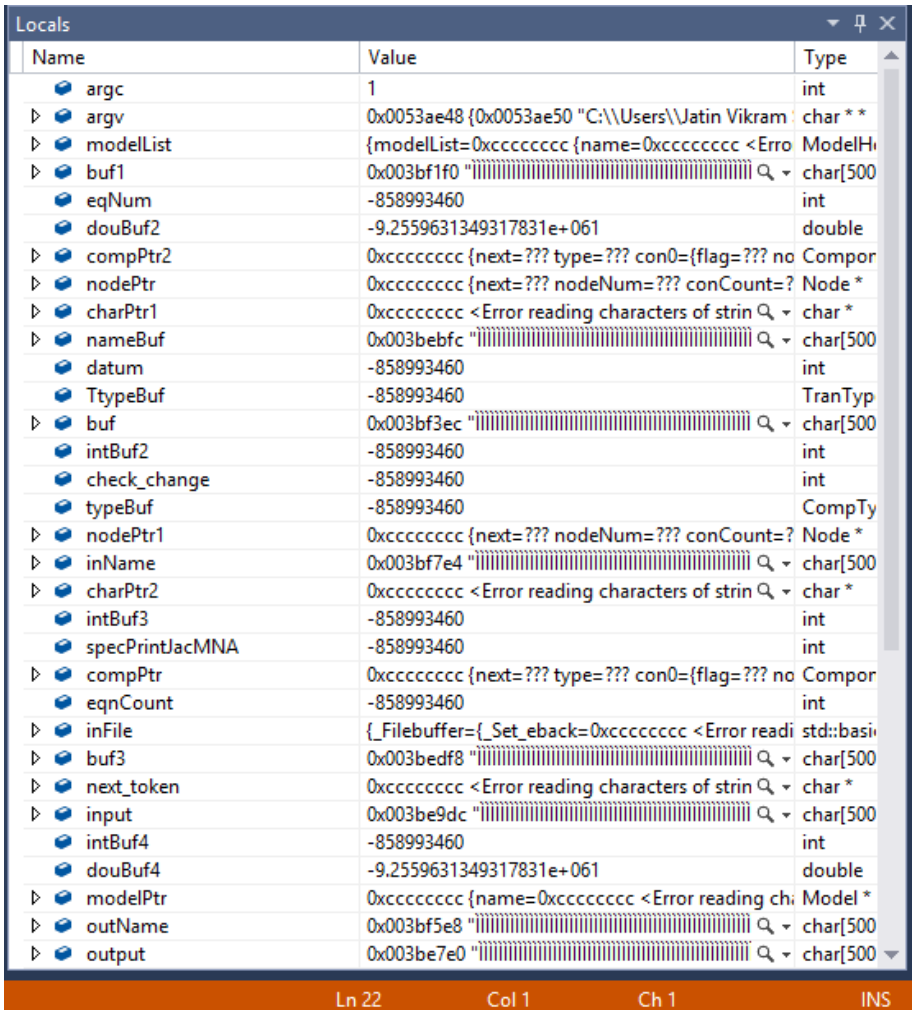

Figure 5: The Locals window showing variables and their values.

7.1 The Datum Node

The original Parser consisted of a datum node. It is usually a node with maximum number of connections and is used as a reference node. Equations for the datum node is omitted. The older version of the parser calculated the datum node. If the ground node is not the datum node, it's equation is printed. When solving circuit equations, we usually chose the ground node as the datum node. In the Parser we have chosen the ground node to be a datum node by default. This would involve further changes to the code. Making the ground node as the datum node provided user with the equations mentioned in literature. This made it easy to compare the equations generated by the Parser with equations available in the literature .

In further improvements to the Parser, user may be given the choice of the datum node.

7.2 Nodal and Modified Nodal Equations

The parser generates nodal and modified nodal equations as per the selection of the user. Since the ground node was made as the universal datum node, some of the equations had missing node voltages. The source of these error was unclear. These were rectified by using the flow map and Locals window. Flow maps depicted which function was responsible for generating the equations. The Locals window gave an idea of how the variables changed. Using the above tools, the appropriate changes have been made and errors have been rectified.

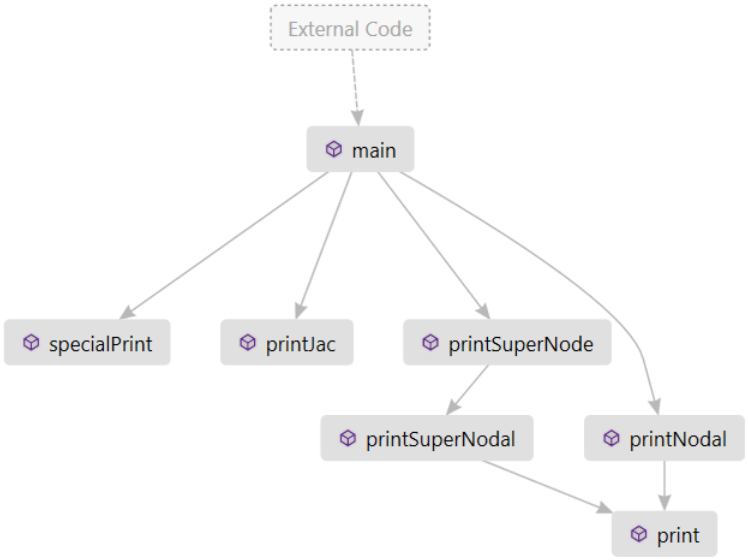

Figure 6: The code flow map for printing equations and Jacobians

7.3 Equation Numbering

The equations in the older version of the parser were being numbered corresponding to the node they belonged. Modified nodal analysis has all the nodal analysis equations and current equations at all the sources. Since the source node voltage equations had already been written, making current equations at the same node gave two different equations same number. This also caused errors in jacobian calculation as there were two values of jacobian at the same indices.

So a new method has been used to number the equations. They have been consecutively numbered. The unknowns, that is, the voltages and currents are still numbered corresponding to the node they belong to. The equations have to be equated to zero, so numbering them consecutively does not make any difference. Similarly, the Jacobians have been calculated with respect to the new reference for the equations.

```
%                    Circuit Equations:

F(1) =  (q1IS )*(exp(-q1N*(X(2)-X(5))) -1) + (-q1IS /q1BR )*(1+q1BR )*(exp(-q1N*(X(1)-X(5))) -1) +
(X(1)-X(6))/rc1 + (X(1)-X(4))/r3 ;

F(2) =  (-q1IS /q1BF )*(1+q1BF )*(exp(-q1N*(X(2)-X(5))) -1) - (- q1IS )*(exp(-q1N*(X(1)-X(5))) -1) +
(-q2IS /q2BF )*(1+q2BF )*(exp(-q2N*(X(2)-X(4))) -1) - (- q2IS )*(exp(-q2N*(X(3)-X(4))) -1) + (X
(2))/re ;

F(3) =  (q2IS )*(exp(-q2N*(X(2)-X(4))) -1) + (-q2IS /q2BR )*(1+q2BR )*(exp(-q2N*(X(3)-X(4))) -1) +
(X(3)-X(6))/rc2 ;

F(4) =  (-q2IS )*(exp(-q2N*(X(2)-X(4))) -1) + (q2IS /q2BR )*(1+q2BR )*(exp(-q2N*(X(3)-X(4))) -1) +
(q2IS /q2BF )*(1+q2BF )*(exp(-q2N*(X(2)-X(4))) -1) - ( q2IS )*(exp(-q2N*(X(3)-X(4))) -1) + (X(4)-X
(1))/r3 ;

F(6) =  (X(6)) -10;

F(5) =  (X(5)) -1.5;

F(5)= (-q1IS )*(exp(-q1N*(X(2)-X(5))) -1) + (q1IS /q1BR )*(1+q1BR )*(exp(-q1N*(X(1)-X(5))) -1) +
(q1IS /q1BF )*(1+q1BF )*(exp(-q1N*(X(2)-X(5))) -1) - ( q1IS )*(exp(-q1N*(X(1)-X(5))) -1) + X(8);

F(6)= (X(6)-X(1))/rc1 + (X(6)-X(3))/rc2 + X(7);
```

Figure 7(a): Older version equations.

```
%*******************************************************************************

%                     Circuit Equations:

F(1) =  (q1IS )*(exp(-q1N*(X(2)-X(5))) -1) + (-q1IS /q1BR )*(1+q1BR )*(exp(-q1N*(X(1)-X(5))) -1) +
(X(1)-X(6))/rc1 + (X(1)-X(4))/r3 ;

F(2) =  (-q1IS /q1BF )*(1+q1BF )*(exp(-q1N*(X(2)-X(5))) -1) - (- q1IS )*(exp(-q1N*(X(1)-X(5))) -1) +
(-q2IS /q2BF )*(1+q2BF )*(exp(-q2N*(X(2)-X(4))) -1) - (- q2IS )*(exp(-q2N*(X(3)-X(4))) -1) + (X
(2))/re ;

F(3) =  (q2IS )*(exp(-q2N*(X(2)-X(4))) -1) + (-q2IS /q2BR )*(1+q2BR )*(exp(-q2N*(X(3)-X(4))) -1) +
(X(3)-X(6))/rc2 ;

F(4) =  (-q2IS )*(exp(-q2N*(X(2)-X(4))) -1) + (q2IS /q2BR )*(1+q2BR )*(exp(-q2N*(X(3)-X(4))) -1) +
(q2IS /q2BF )*(1+q2BF )*(exp(-q2N*(X(2)-X(4))) -1) - ( q2IS )*(exp(-q2N*(X(3)-X(4))) -1) + (X(4)-X
(1))/r3 ;

F(5) =  (X(6)) -10;

F(6) =  (X(5)) -1.5;

F(7)= (-q1IS )*(exp(-q1N*(X(2)-X(5))) -1) + (q1IS /q1BR )*(1+q1BR )*(exp(-q1N*(X(1)-X(5))) -1) +
(q1IS /q1BF )*(1+q1BF )*(exp(-q1N*(X(2)-X(5))) -1) - ( q1IS )*(exp(-q1N*(X(1)-X(5))) -1) + X(8);

F(8)= (X(6)-X(1))/rc1 + (X(6)-X(3))/rc2 + X(7);
```

Figure 7(b): Modified version equations.

7.4 Jacobian

The Jacobians are required by the Matlab script to solve for the DC operating points. There were errors in calculation of the Jacobian values. There were two or more different values of Jacobians for the same indices because there were two or more equations with the same number. This problem was rectified when the consecutive naming of equations was implemented. In addition to this issue, the Jacobians were not being printed for a few nodal equations. The code has been appropriately modified to get the output. One other issue included repetition of Jacobian for some equations. The value of Jacobian at some nodes were being repeated because they were being visited more than once. The values of the repeated Jacobian were not faulty but to maintain the output uniformity, they have been fixed to be printed only once.

```
JAC(5, 2) = 0;

JAC(5, 3) = 0;

JAC(5, 4) = 0;

JAC(5, 6) = 0;

JAC(8, 1) =  ( - q1IS * q1N / q1BR)*(1 + q1BR)*(exp(-q1N*(X(1) - X(5)))) + (q1IS * q1N)*(exp(-q1N*
(X(1) - X(5)))) + 0 ;

JAC(8, 5) =  ( - q1IS * q1N)*(exp(-q1N*(X(2) - X(5)))) + (q1IS * q1N / q1BR )*(1 + q1BR )*(exp(-
q1N*(X(1)-X(5)))) + (q1IS * q1N / q1BF )*(1 + q1BF )*(exp(-q1N*(X(2) - X(5)))) - (q1IS * q1N)*(exp
(-q1N*(X(1) - X(5)))) + 0 ;

JAC(8, 8) = 1;

JAC(8, 2) =  (q1IS * q1N)*(exp(-q1N*(X(2) - X(5)))) - (q1IS * q1N / q1BF)*(1 + q1BF)*(exp(-q1N*(X(2)
- X(5)))) + 0 ;

JAC(8, 3) =  0 + 0 ;

JAC(8, 4) =  0 + 0 ;

JAC(8, 6) =  0 + 0 ;

JAC(8, 7) = 0;

JAC(7, 1) =  (-1/rc1) + 0 + 0 ;

JAC(7, 5) =  0 + 0 + 0 ;
```

Figure 8(a): Older Version Jacobians

```
JAC(6, 8) = 0;

JAC(6, 5) = 1;

JAC(6, 2) = 0;

JAC(6, 3) = 0;

JAC(6, 4) = 0;

JAC(6, 6) = 0;

JAC(7, 1) =  ( - q1IS * q1N / q1BR)*(1 + q1BR)*(exp(-q1N*(X(1) - X(5)))) + (q1IS * q1N)*(exp(-
q1N*(X(1) - X(5)))) + 0 ;

JAC(7, 5) =  ( - q1IS * q1N)*(exp(-q1N*(X(2) - X(5)))) + (q1IS * q1N / q1BR )*(1 + q1BR )*(exp
(-q1N*(X(1)-X(5)))) + (q1IS * q1N / q1BF )*(1 + q1BF )*(exp(-q1N*(X(2) - X(5)))) - (q1IS *
q1N)*(exp(-q1N*(X(1) - X(5)))) + 0 ;

JAC(7, 8) = 1;

JAC(7, 2) =  (q1IS * q1N)*(exp(-q1N*(X(2) - X(5)))) - (q1IS * q1N / q1BF)*(1 + q1BF)*(exp(-q1N*
(X(2) - X(5)))) + 0 ;

JAC(7, 3) =  0 + 0 ;

JAC(7, 4) =  0 + 0 ;

JAC(7, 6) =  0 + 0 ;

JAC(7, 7) = 0;

JAC(8, 1) =  (-1/rc1) + 0 + 0 ;
```

Figure 8(b): Modified Version Jacobians

8 Conclusion

The parser has been improved in terms of quality of output. The equations are in the format required by the Matlab code. The equations, nodal as well as modified nodal equations, have been corrected. The Jacobians that posed a lot of issues have also been corrected to produce the result that can be used to employ homotopy. Further improvements can be made to incorporate more devices like current sources, MOSFET's into the parser.

References

[1] L. Trajkovic, "DC Operating points of transistor circuits," *Nonlinear Theory and Its Applications, IEICE,* vol. 3, no. 3, pp. 287-300, July 2012.

[2] E. Chan, Documentation on the parser program, Technical Report, UC Berkeley, 1996.

[3] A. Dyess, Finding dc operating points of chua's circuit using homotopy methods, Technical report, University of Alabama, 1997.

[4] A. Dyess, E. Chan, H. Hofmann, W. Horia, and L. Trajkovic, "Simple implementations of homotopy algorithms for finding dc solutions of nonlinear circuits," *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems,* volume 6, pages 290–293 vol.6, Jul 1999.

[5] J. Eric Melo and L. Trajkovic, Improving an electronic circuit simulator based on homotopy methods, Technical report, SFU Burnaby, 2014.

[6] https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx