SCISIP – Program for Switched Circuit Analysis in Matlab

Tomas Lukl
Department of Telecommunications
Brno University of Technology
Purkynova 118, 612 00 BRNO
CZECH REPUBLIC
+42054114 9217
thomlu@eabrno.cz

Jaroslav Vrana
Department of Telecommunications
Brno University of Technology
Purkynova 118, 612 00 BRNO
CZECH REPUBLIC
+42054114 9217
xvrana02@stud.feec.vutbr.cz

Jiri Misurec
Department of Telecommunications
Brno University of Technology
Purkynova 118, 612 00 BRNO
CZECH REPUBLIC
+42054114 9200
misurec@feec.vutbr.cz

ABSTRACT
The article deals with the SCISIP program, which is a program for analysis of switched as well as general type electronic circuits. SCISIP is being developed in mathematical and programming tool Matlab. The reader can find the description of an open and modular structure of the program as well as the algorithms used in the simulation core. In the end one can find an example of the analysis of SI lossless integrator and its graphical results.

1. INTRODUCTION
General switched circuits are circuits with quantized signal processing. The switch is the most frequently used electrical element in these circuits. In real world, switches are usually realized by transistors in the CMOS technology (as IC in most cases). This is the case of externally controlled switches (electronic filters) while in internally controlled switches (Switch Mode Power Supplies) diodes and thyristors are the most frequently used elements for switching. There is also a third group of switched circuits – mixed controlled switches (internally as well as externally controlled switches), which are used in Σ-Δ converters.

During the testing stage of the development of a new switched circuit we want to analyze it using a computer. We can use Spice-like simulators and transistor-level circuit models. But in this case, the simulator spends a lot of computing time to calculate a circuit response in the switching instants (which we are not so interested in) because of the nonlinear character of switch models. Using linear or even ideal models of switches is not so time consuming and not so inaccurate. But in this case, most Spice-like simulators have problems using ideal switch models. This leads to the need to develop a new tool for the analysis of switched circuits. A lot of these tools have been developed in the past, e.g. MALINSC, MATSC, COCOSC, WATSCAD, CPPSIM and SWANN. But these tools are either old (or thus no longer supported) or very specific (for a closed group of switched circuits, e.g. switched capacitors – SC or just linear SC).

In the previous paragraph the main reason why to start the development of a new switched circuit analysis tool was presented. The following most important properties were declared:

- ability to analyze general switched circuits,
- ability to analyze even general electronic circuits,
- open and modular structure,
- easy-to-use command line and graphical environment with simple text input and output structures (files),
- availability to the universities.

Its name is SCISIP – Switched CIrcuit SImulation Program.

2. PROGRAM STRUCTURE
2.1 Matlab environment
SCISIP is still being developed in the Matlab environment (at this moment, Matlab r14sp1), because Matlab is widely used at universities, so students and professors know how to work with it. The decision to use Matlab as the development environment was based on the following pros and cons:

- C++ or Java programming language similarity,
- many predefined mathematical functions,
- vector-and-matrix-based Matlab core (useful for methods for numerical analysis, e.g.: modified nodal analysis – MNA, Newton-Raphson iteration method – NR or the other methods for numerical integration),
- effective creation of graphical user interface (GUI),
- classical OOP technique available.

- installed and running Matlab is needed,
- complicated (or even impossible) Matlab code compilation of big projects,
- limited group of GUI objects,
- problems with OpenGL graphical rendering mode (very effective otherwise) and large amount of GUI objects,
- very high demand on hardware for new versions of Matlab,
- mutual incompatibility of previous major releases of Matlab (especially between versions 4 and 5) - this can be quite a big problem in large projects.

2.2 Modularity

The most important requirement was the modular and open structure. Each module represents logically separated parts of whole system. Open structure represents the fact that a new module can be easily "plugged" into the existing system so its functionality can be extended. Additionally, the source code of the program will be (after completion) available (except the source code of the system core).

The main part of system structure is the system core. Its main purpose is to accept events from all modules and perform a requested action. A schema of the system core is shown in Figure 1. There are four function blocks in the system core:

EHS – Event Handler System. A module calls (using a public function `invokeEvent`) this block and passes an event ID. EHS looks for this ID in the event table and performs the requested operation (e.g. startup of another module). This structure can be explained by the following example: the simulation module finishes the analysis and sends an event to inform that the analysis is complete. EHS looks at the event table and calls a module, which displays the results in a graph). Event ID is a number of class "uint16" – 16bit unsigned integer.

DSS – Data Storage System. The big issue is how to store data for next calls of a Matlab function. For this purpose we can use either the `globally declared` or the `persistent` variable. Persistent variable is the only acceptable possibility because globally declared variable is simultaneously a public variable and thus accessible from the Matlab command line. There is also a possibility to use functions `guidata` or `setappdata` (standard Matlab functions) but these functions are intended to store GUI data, so we would always have to have a graphical object (e.g. a window) created. DSS provides data storage for all modules during the entire time SCISIP is running. EHS event table or EMHS (error and message handler system) data are stored using DSS. This block also simplifies access to HDD files, especially text files.

SE – SCISIP Exec provides a Matlab command line interface for executing SCISIP functions. SE is the first point where users enter the SCISIP system. They can either execute the GUI or the analysis procedure using SE.

MIR – Module InstalleR provides a user-friendly tool for upgrading an existing module or installing a new one.

Modules themselves are another part of program structure. There are naturally two types of modules. Internal – predefined modules, which cannot be removed (they can be only updated) and external (user defined) modules. The simulation core, the already mentioned EMHS module, LSM (language support module) and GHM (GUI Handler Module) are internal modules.

EMHS module provides a uniform way of displaying error, warning and other messages based on events (see EHS). The text and description of these messages are saved in a text file, which comes with each module.

LSM module provides language support for GUI and EMHS through language definition files (the default language of SCISIP is English and another language definition file comes with the Czech language).

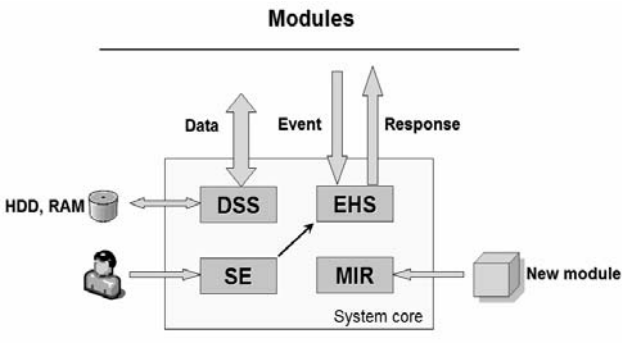

Figure 1. Schema of the designed system structure.

There is no strictly defined or required structure of user modules. They just have to offer necessary functions for accessing public module data (e.g. in the case of simulation core a data structure with analysis results).

2.3 Input Files Definition

There are two main input files: netlist file and model file. Both these files are saved in clear text format which has simple and intuitive syntax (structure). In the following two paragraphs you can see a description of a new simple modeling language. And that was the idea – simple and intuitive utilization. Due to the modular structure of the SCISIP program, one can program a module to handle widely used complex modeling languages such as Spice language or VHDL-AMS.

2.3.1 Netlist

Netlist is a structure that contains all the necessary data for analysis. Netlist has two main parts: the analysis definition block and the circuit definition block.

Analysis definition block tells the Simulation core "what to do". It contains several *key words* with their parameters. You can see the list of several keywords in Table 1. There are very simple syntax rules for writing an analysis definition block:

- Each key word, introduced with character `$`, must be on a separate line. If there is more than one keyword used, you can enclose all keywords between two `$` characters.
- Unknown keywords are ignored (skipped).
- The character `%` introduces a comment text when used wherever on the line.

There are a few possibilities how to write a keyword and its parameters, so the documented syntax need not be strictly observed. Syntax errors, which occur during parsing the netlist file, are skipped (except errors that could cause incorrect parsing or syntax errors in required keywords). For a deeper insight into the netlist syntax, see an example in Table 2.

Table 1: The list of netlist keywords.

Keyword name	Description
`analysis` *param*	Defines the type of analysis. The *bias* or *tran* strings can be used as parameters. Both parameters can be written at the same time

	(delimited by white space).
`algorithm` *param*	Tells the simulation core which algorithm to use.
`epsilonr_i` *param* `epsilonr_u` *param* `epsilonr_q` *param* `epsilonr_f` *param*	Defines the relative error of solution for current variables (i), voltage variables (u), charge variables (q) and magnetic flux variables (f).
`deltamax` *param*	Defines the maximum error of the solution.
`maxiter` *param*	Defines the maximum number of iterations (for numerical methods).
`variablestep` *param*	Boolean parameter, which tells the simulation core whether to use the variable-step algorithm or not (numerical integration).
`initstep` *param*	Sets the initial step length.
`maxstep param`	Sets the maximum step length.
`minstep param`	Sets the minimum step length.
`tspan param`	Defines the time range of transient analysis plus user-or-system-defined time instants within this time range.
`timemetering`	Enables or disables the measurement of analysis time.
`output param`	Defines output of analysis results. The following strings can be used as parameter: *file* (saves the results into a file), *graph* (uses a module to display analysis results), *structure*.
`dsource param`	Defines the digital clock for driving externally driven switches. The parameters can be: *d* – delay; *r* – ratio; ***T*** – period or ***f*** – frequency; ***name*** – a name of a "clock wire". Parameters written in bold are required. More than one digital source can be defined.

Circuit definition block defines how electrical parts are interconnected, how they are modeled, and it can provide their (user-defined) parameters. Each line which begins with an English letter is parsed as a circuit definition line. Each circuit definition line has the following structure:

`ref nodes model [params]`,

where

`ref` is the reference name of electrical element. It can contain English letters and numbers, but must begin with letters.

`nodes` are numbers of nodes, delimited by white space, where electrical elements are connected to.

`model` is the name of a model. The character "!" in the model name introduces an internal model.

`[params]` is an optional parameter. One can redefine one or more model parameters. This parameter consists of pairs in the form `parameter_name = parameter_value`.

The following rules apply to numerical values, used wherever in a Netlist file:

- Floating-point double precision (IEEE 754 standard) format can be used.
- Characters "." or "," stand for decimal point.
- Basic mathematical operators (+,-,*,/) can be used.
- Commonly used prefixes can be used as shown in the following list: f = 1e-15, p = 1e-12, n = 1e-9, u = 1e-6, m = 1e-3, k = 1e3, M = 1e6, T = 1e9. Only the prefixes *mili* (m) and *Mega* (M) are case-sensitive.

An example of a netlist file, which describes a circuit in Figure 2, follows.

```
$
analysis bias
 algorithm AL
 maxiter 100
 deltamax 1u
analysis tran
 algorithm TR
 maxiter 20
 tspan 0 3m 'Dig'
dsource name='Dig' T=40u
$
U 1 0 U [U=5]
R 1 3 R [R=1k]
I1 1 3 I [I=400u]
I2 1 2 I [I=200u]
I3 1 6 isin [Ia = 130u;f = 1k]
T1 5 2 0 !nmos [Cgs=20p;L=2u;W=20u]
T2 4 2 0 !nmos [Cgs=20p;L=2u;W=20u]
T3 4 3 0 !nmos [Cgs =20p;L=2u;W=40u]
S1 6 2 sw [Ron=1u;Roff=0.1G;clk='~dig']
S2 2 5 sw [Ron=1u;Roff=0.1G;clk='dig']
S3 2 4 sw [Ron=1u;Roff=0.1G;clk='~dig']
```

2.3.2 Model

The model of an electrical element mathematically describes its properties. There are a few types of models used in SCISIP:

- Ideal models are defined by a matrix. There are many models, defined in SCISIP: resistor, capacitor, inductor, OpAmp, CCs, CFA, etc.
- Nonlinear models are defined by a mathematical expression. Models of a diode, bipolar and unipolar transistor and thyristor are predefined.

- Macro models are in fact circuits in a circuit. Their definition is similar to the definition of a circuit. Real-world electrical elements are modeled as macro models, e.g. nonlinear dynamic model of a diode.
- "Plot models" are models defined by a table of measured values. The simulation core then uses this table to automatically generate an expression, which describes a given element.

In the model file we can use the following keywords (they are used without the character $):

`model` *name* – each model file must contain this keyword as an identifier of model name.

`modeltype` *name* defines the type of model. The following strings for parameter *name* are acceptable (see the list of model types above): *stamp*, *matx*, *math*, *plot*, *macro*.

`modfuntype` *type* sets the type of model function (for *math* model type only). Acceptable types are *lin* and *nlin*.

`pindef` *param* defines internal names of element pins (English letter or a number).

`inputs` *param* defines input variables (port voltages, pin currents).

`outputs` *param* defines output variables and their derivatives with respect to input variables (if applicable).

`moddef`:: introduces a block which defines the given model itself. This block then must end in the keyword `$enddef`. There is a different format of this block for each type of model, but its description is beyond the scope of this contribution.

In the example below, you can see the static nonlinear model of an NMOS transistor.

```
model nmos
modeltype math
modfuntype nlin
pindef G D S
inputs uDS uGS
outputs iD,iD_uDS,iD_uGS
moddef::
  Ut = 0;
  W = 20*1e-6;
  L = 2*1e-6;
  kn = unCox/2 * W/L;

  if uGS<Ut
        iD = 0; iD_uDS = 0; iD_uGS = 0;
  elseif uDS <= (uGS-Ut)
        iD_uGS = 2*kn*uDS;
        iD_uDS = 2*kn*(uGS-Ut)-2*kn*uDS;
        iD = kn * (2*(uGS-Ut)*uDS - uDS^2);
  else
        [iD_uDS iD_uGS] = [0 2*kn*(uGS-Ut)];
        iD = kn * (uGS - Ut)^2;
  end
$enddef
```

3. UTILIZATION OF ANALYSIS METHODS

The simulation core contains a mathematical apparatus which can solve the system of nonlinear differential equations which mathematically describes the circuit analyzed. The commonly used method for forming circuit equations is called Modified Nodal Analysis (MNA). Using this method one can write the following system of equations [1], which describes the general type of a circuit

$$\begin{aligned}\mathbf{A}_{\mathrm{I}}\mathbf{f}_{\mathrm{I}}\left(\mathbf{u}_n, \frac{d\mathbf{q}_{\mathrm{I}}(\mathbf{u}_n)}{dt}, \mathbf{i}_{\mathrm{II}}, \frac{d\boldsymbol{\psi}_{\mathrm{I}}(\mathbf{i}_{\mathrm{II}})}{dt}, \mathbf{s}(t)\right) + \mathbf{A}_{\mathrm{II}}\mathbf{i}_{\mathrm{II}} &= \mathbf{0},\\ \mathbf{f}_{\mathrm{II}}\left(\mathbf{u}_n, \frac{d\mathbf{q}_{\mathrm{II}}(\mathbf{u}_n)}{dt}, \mathbf{i}_{\mathrm{II}}, \frac{d\boldsymbol{\psi}_{\mathrm{II}}(\mathbf{i}_{\mathrm{II}})}{dt}, \mathbf{s}(t)\right) &= \mathbf{0}.\end{aligned} \tag{3.1}$$

where **A** is the incidence matrix, **f** is the vector of functions which describe each electrical element, **u** is the vector of nodal voltages, **i** is the vector of currents, **s** is the vector of sources, **q** is the vector of charges and **ψ** is the vector of magnetic fluxes. Indexes I and II denote current- and voltage-defined branches, respectively.

System (3.1) is generally nonlinear and differential. Numerical methods for solving this type of equations are used.

3.1 Numerical integration

Numerical integration is the first step while solving system (3.1). During numerical integration (solving differential equations) we convert a system of differential equations into a system of nonlinear algebraic equations. Most of integration methods substitute all the derivatives in the system by the following general formulae

$$\dot{x}_n = \gamma_0 (x_n - d_x), \tag{3.2}$$

$$x_n = d_x. \tag{3.3}$$

In equations (3.2) and (3.3) the coefficients γ_0 and d_x determine the numerical integration method. A method defined by eq. (3.2) is called implicit, while substitution by (3.3) is called explicit.

In the Simulation Core, the backward Euler method, trapezoidal method or BDF methods are used (see (3.4), (3.5) and (3.6) respectively). For a more detailed description see [1].

$$x_n = x_{n-1} + h_n \dot{x}_n. \tag{3.4}$$

$$x_n = x_{n-1} + \frac{h_n}{2}(\dot{x}_{n-1} + \dot{x}_n). \tag{3.5}$$

$$\dot{x}_n = \sum_{k=0}^{l} \gamma_k^{(l)} x_{n-k}. \tag{3.6}$$

3.2 Nonlinear equations

From the previous step we have a system of nonlinear algebraic equations, generally in the form

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}. \tag{3.7}$$

In the Simulation core, two methods are used to solve system (3.7): the Newton Raphson method (NR) and the Homotopy method.

The NR method is the most common method used. It is described by the system of equations

$$\mathbf{J}\left(\mathbf{x}^{(k)}\right)\cdot\Delta\mathbf{x}^{(k+1)} = -\mathbf{F}\left(\mathbf{x}^{(k)}\right),$$
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k+1)}. \tag{3.8}$$

This form of the NR method is called incremental. It has a few advantages over the standard form. The biggest disadvantages of the NR method are local convergence and ability to find just one root of system (3.7).

The Homotopy method is a less known method, which is described by the equation [4]

$$\mathbf{H}\left(\mathbf{x}(s),\lambda(s)\right) = \mathbf{0},$$
$$\dot{\mathbf{x}}^{t}\dot{\mathbf{x}} + \dot{\lambda}^{2} - 1 = 0, \tag{3.9}$$

where the point over vector **x** means the derivative of **x** with respect to variable s (the same for variable λ). The Homotopy function **H** is a function such that $\mathbf{H}(\mathbf{x},0) = 0$ is simply solvable and the root of $\mathbf{H}(\mathbf{x},1) = 0$ is also the root of (3.7). There are a few methods how to solve homotopies. Eq. (3.9) is called the Arc length method and its big advantages are global convergence and ability to find multiple roots of (3.7). In the Simulation core, this method is used in the case when the NR method fails.

After this step we have a system of linear algebraic equations, which is solvable very simply, using known methods such as LU decomposition.

3.3 Methods for switched networks analysis

It is simpler to analyze externally controlled networks than internally controlled ones. Inconsistent initial conditions are a big problem during switched networks analysis. In the case of linear, externally controlled switched circuits, the backward Laplace transform is used for analysis. During an analysis of nonlinear switched networks, the following procedures are used according to the way the switches are controlled [8], [7].

Externally controlled switches

The situation is here a little bit simpler because we know the exact time instants, when the switches change their state. The analysis of nonlinear switched networks with ideal switches consists of two steps:

- Determining the initial conditions just after the switching instant.
- Determining the presence of the Dirac impulses.

The numerical integration method used must be auto-starting at the times the switches are changing their state. In the Simulation Core, 1st order BDF method or the Trapezoid method is used.

Internally controlled switches

Because internally controlled switches can change their state at each time instant (they are controlled by the value of a circuit variable) so after each integration step an algorithm must check whether to change the state of a switch or not. For this checking, the sign of control functions $p_i(t)$ is used. If the sign changes, the algorithm has to find this time instant t_s precisely (using the NR or the Homotopy method). The algorithm then computes the response of the circuit in time t_s+h_{min} and checks the state of the switches again. If a switch has changed its state, the response values of the circuit at the instant t_s+h_{min} are canceled, the circuit topology is changed, and a new response is calculated. This loop is repeated until all the switches are in steady state. The initial conditions and the area of the Dirac impulse are computed by two-step backward integration. After that the numerical integration can proceed.

Prof. Vlach has defined 6 types of internally controlled switches [9]:

- An ideal diode,
- an ideal thyristor,
- a voltage-controlled switch,
- a current-controlled switch,
- a switch controlled by a diode,
- an inverse switch controlled by a diode.

4. EXAMPLES

You can see a schematic diagram of SI lossless integrator (which is a block frequently used in SI filters) in Figure 2. In this figure you can also see a window of the Schematic Editor tool. This is an external (user defined) module, which contains all the necessary functions, methods and graphical object to draw and edit an electrical schematic diagram. Individual parameters of models of used electrical elements can be changed directly in the Schematic Editor. The output of this module is a netlist file or structure (see chap. 2.3.1).

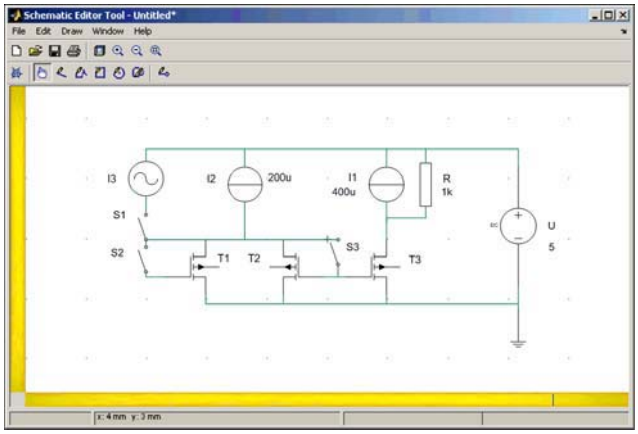

Figure 2: Schematic diagram of the SI (switched currents) lossless integrator drawn in the Schematic Editor tool.

When the simulation is complete, we need to display its results somehow. There are a few possibilities according to parameter `output` (see Netlist parameters in chap. 2.3.1). To plot a graph is the most user-friendly way how to display the results. There is a module called Presentation Module in SCISIP. It contains all the necessary functions, methods and graphic objects for displaying the time response in a graph or the bias point values of the

analyzed circuit. You can see the time response of the circuit from Figure 2 in Figure 3. The green curve is input current i_{i3} and the red one is output current i_R, which flows through resistor R.

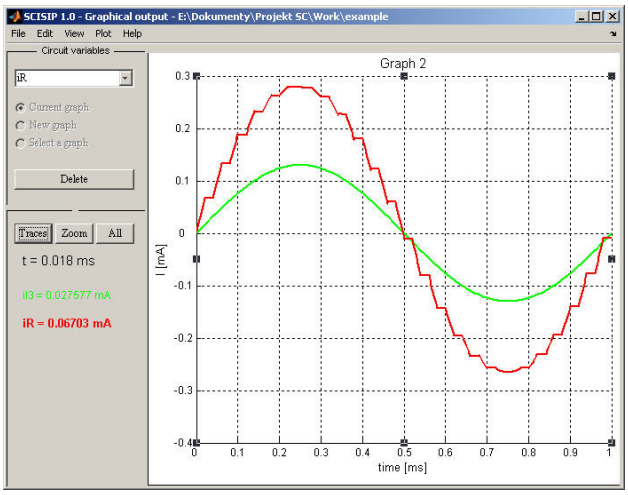

Figure 3: Results of the analysis of SI integrator (see Figure 2) in the Presentation Module window – i_{I3} (green) and i_R (red).

5. CONCLUSION

A new simulation tool for switched circuits in Matlab was presented in the paper. The tool is still being developed, so the results presented are the main functional fragments of the whole program. After SCISIP is completed, it can be used in some theoretical classes (like Circuit Theory, Analogue Circuits, Electronic Filters, etc.) and their computer practices even during switched circuit design. It will be available and utilizable at universities, because Matlab is a widely used mathematical and programming environment there.

6. ACKNOWLEDGMENTS

This paper was worked out with the support of FRVŠ 3248/2006/G1 project.

7. REFERENCES

[1] Ogrodzki, J. *Circuit Simulation Methods and Algorithms*. CRC Press 1994, ISBN 0-8493-7894-X.

[2] Lukl, T., Novotny, V., Misurec, J. Computer-Aided Circuit Analysis with Respect to Switched Circuits. *WSEAS Transactions on Electronics*, ISSN 1109-9445, 2005, vol. 2, p. 139 - 143.

[3] Vlach, J., Bedrosian, D. Analysis of Switched Networks. *International Journal of Circuit Theory and Applications.* vol. 20, p. 728.1-728.17, 1992.

[4] Lukl, T. *Utilization of homotopy methods for analysis of not only switched circuits; in Czech - Elektrorevue – Internet magazine*. http://www.elektrorevue.cz, ISSN 1213-1539, 2005.

[5] Vlach, J., Singhal, K. *Computer Methods for Circuit Analysis and Design. Second Edition*. New York: International Thomson Publishing Co., 1994. 712 pages. ISBN 0-442-01194-6.

[6] Toumazou, C., Lidgey, F.J., Haigh, D.G. *Analogue IC design: the Current-mode approach*. Peter Peregrinus Ltd., England, 1998. ISBN 0-86341-297-1.

[7] Novotny, V., Lukl, T. Algorithms for Non-linear Switched Network Simulation in *Telecommunications and Signal Processing TSP-2003. Intrn.Conference TSP 2003 – Telecommunications and Signal Processing*. Brno, CR: FEKT VUT Brno, 2003, p. 205 - 208, ISBN 80-214-2433-8.

[8] Vlach, J., Wojciechowski, J.M. Analysis of Nonlinear Networks with Inconsistent Initial Conditions. *In IEEE Transaction on Circuits and Systems – I: Fundamental Theory and Applications*. vol. 42, no. 4, p. 195-200, 1995.

[9] Vlach, J., Bedrosian, D. Time-Domain Analysis of Networks with Internally Controlled Switches. *IEEE Transaction on Circuits and Systems – I: Fundamental Theory and Applications*, vol. 39, no. 0, p. 1-14, 1992.