fREEDA$^{\mathrm{TM}}$ User's Manual

Version 1.3

June 29, 2007

Compiled on July 25, 2007.

Michael B. Steer,
Carlos E. Christoffersen,
Mark Basel,
Joseph N. Hall

July 25, 2007

Contents

Chapter 1

Introduction

1.1 Overview of fREEDA

fREEDA$^{\text{TM}}$ is a multiphysics simulator that is based on the use of compact models as used in circuit simulators. fREEDA^{TM} carries a Gnu Public License (GPL) and is available at http://www.freeda.org.

ifREEDA$^{\text{TM}}$ is a companion interactive GUI based on the QUCS schematic capture engine see http://qucs.sourceforge.net/ . ifREEDA$^{\text{TM}}$ uses the Qt® is a registered trademark of Trolltech AS in Norway, The United States of America, and other countries worldwide. See http://trolltech.com/copyright for their copyright restriction and http://trolltech.com/products/qt/lice for licensing information. ifREEDA$^{\text{TM}}$ uses the Open Source Edition of Qt®.

fREEDA$^{\text{TM}}$ and ifREEDA$^{\text{TM}}$ are released under the GPL license and so is open source software. fREEDA$^{\text{TM}}$ and ifREEDA$^{\text{TM}}$ can be sold but developments that are commercialized must be made available as open source software.

The community is welcome to extend the capabilities of *f*REEDA™ particularly model development. *f*REEDA™currently supports the following analyses: transient (many types including high dynamic range capability), wavelet transient, harmonic balance, large signal noise analysis including phase noise, AC, DC. There are a large number of models available (there are at least 107 but the number grows all the time). There is an extensive support for true electro-thermal modeling capturing long tail thermal effects. We need to develop application notes to show how to use all the capabilities. Most of the developments have been reported in Master;s theses and PhD dissertations at North Carolina State University, University of Arizona, Queen's University. Contact Michael Steer at m.b.steer@ieee.org if you are interested in modifying *f*REEDA™ .

1.2 A Multi-Physics Simulator

A number of concepts are supported that enable multiphysics modeling. The most important of these are

1. Local reference terminals supplanting universal ground [S13, S14]. (Different physics can have their own reference. High speed circuits and microwave circuits do not posses a global ground and so the distributed nature of high-speed circuits is naturally captured.

2. Energy norm. fREEDA^{TM} uses an energy norm in computing solutions [S12, S15]. Each terminal has two quantities potential and flux. This contrasts with the conventional circuit simulator paradigm of solving Kirchoff's Current Law which only addresses flux conservation. This solution does not work when there is no flux.

3. fREEDA^{TM} uses state-variables and parameterization [S16, S1] so that homotopy techniques do not need to be used to arrive at a solution. For example fREEDA can arrive at a solution starting from a zero initial guess. The modeling scope is far beyond the modeling capabilities of SPICE-like simulators. The modeling scope is defined by

$$y(t) = F\left[\begin{array}{l} x_1(t),\ldots,x_n(t),\frac{dx_1(t)}{dt},\ldots,\frac{dx_n(t)}{dt}, \\ \frac{d^2x_1(t)}{dt^2},\ldots,\frac{d^2x_n(t)}{dt^2},\frac{d^3x_1(t)}{dt^3},\ldots,\frac{d^3x_n(t)}{dt^3}, \\ x_1(t-\tau_1),\ldots,x_n(t-\tau_1) \end{array}\right]$$

where $F[]$ can be nonlinear. Note that rue time-delays are supported.

1. fREEDA^{TM} supports distributed circuits using UIUC developed technology to use frequency-domain characterizations in transient analysis. [S6]

2. Electro-thermal modeling is supported and has been implemented and verified for microwave integrated circuits. Electro-thermal elements are based on a unique theory that captures thermal nonlinearities [S7], [S8]

3. It is very easy to write a model in fREEDA. For example. The ekv FET model was added to fREEDA^{TM} in May 2007. This took approximately four weeks with most of the time spent reverse engineering the ekv model to discover the smoothing algorithms used (this information was only available with an NDA). Turning the electrical model into a complete electro-thermal model took six hours. Making changes to the model takes an almost insignificant amount of time as manual derivatives do not need to be developed (uses automatic differentiation technology). Model code in fREEDA^{TM} uses is typically 5 to 10% of the code required to implement the model in SPICE. About 90% of . fREEDA^{TM} code is off-the-shelf numerical libraries. For example, ADOLC was modified to suit . fREEDA^{TM} as at the time, in 2005, . fREEDA^{TM} was regarded as the most sophisticated implementation of ADOLC.

4. fREEDA^{TM} currently supports 150 models, some of the more exotic are fully-physical models for molecular electronics [S4] and models that capture high-order fileters in transient analysis [S3].

5. fREEDA^{TM} supports many types of analyses including four main transient analyses each with particular attributes. For example, one has a dynamic range exceeding 140 dB and particularly useful in modeling RFICs. Wavelet transient analysis [S9] uses wavelet technology albeit somewhat disappointing because of matrix ill-conditioning. In time this will be addressed and further support multi-physics modeling.

fREEDA^{mathrmTM} has an input format that is similar to the SPICE input format with extensions for variables, sweeps, user defined models, and repetitive simulation. The program provides a variety of output data and plots. The netlist format (including output commands) is discussed in chapter 2.

The program supports several types of analyses, subcircuit instances, and local reference nodes. The harmonic balance approach in fREEDA is discussed in chapter **??**. The convolution transient is described in chapter **??**.

fREEDA™ supports two schematic and layout capture engines. One of these is iFREEDA™ which is based on the QUCS engine and is intricately connected to fREEDA™. iFREEDA™ uses the same code tree as fREEDA™. The other schematic engine is Electric (Editor) of the Electric VLSI Design System, http://www.staticfreesoft.com fREEDA™ also provides an interactive interface (GUI) called iFREEDA. Both Electric and iFREEDA™ support local reference terminals

fREEDA™ and iFREEDA™ are available from http://www.freeda.org

fREEDA^{mathrmTM} provides a graphical user interface which includes a netlist editor, a run manager and a output viewer. Details are given in chapter 6. While this is no longer distributed with fREEDA it still exists.

fREEDA^{mathrmTM} allows the addition or removal of new circuit elements in a very simple way. It is designed so that new circuit elements can be coded and incorporated into the program without modification to the high-level simulator and editor. It is also quite simple to add a new analysis type. Some insight into the program architecture is given in chapter **??**.

1.3 Supported Platforms

fREEDA^{mathrmTM} has been developed using the GNU compilers. All platforms supported by these compilers should be able to run fREEDA^{mathrmTM}. Most of the program is written in ANSI C++ using object-oriented techniques, but it also contains off-the-shelf libraries and routines written in C and FORTRAN. The user's interface is written in Java.

fREEDA^{mathrmTM} has been compiled and run successfully in Solaris, and HPUX, Linux, Windows/cygwin with virtually no alteration. The main development environment is linux but cygwin should work just as well.

1.4 Command Line Options

The syntax for the simulator engine invocation is

`freeda` <*netlist file*> [<*output file*>]
or (on Cygwin) `freeda.exe` <*netlist file*> [<*output file*>]

where <*netlist file*> is the input file name (normally ended with `.net`) and <*output file*> is the output file name (normally ended with `.out`). If the output file name is omitted, the default output name is formed by replacing `.net` with `.out` in the input file name, or by just adding `.out` if the input file name does not ends with `.net`.

When not running a netlist file, fREEDA^{mathrmTM} accepts the following command line flags:

Flag	Description
`-a`	: generate analysis catalog files.
`-c, --catalog`	: generate element catalog files.
`-c` *elementName*	: generate catalog files for teh element *elementName* (must be lower case).
`-h, --help`	: get help (this message).
`-l, --licence`	: show license information.
`-v, --version`	: print program version.

fREEDA is self-documenting for analyses and elements. Data is taken from the data structures for the elements, authors of analyses and elements often provide more extensive documentations.

1.5 Release Notes

1.5.1 Installation Notes

The installation instructions are located in the file `README.install` in the `src/` directory.

1.5.2 Directory Structure

The simulator assumes the directory structure

- $USER/freeda
- $USER/library (where the libraries reside and not overwritten by new releases)
- $USER/freeda/projects (where the projects reside and not overwritten by new releases)
- $USER/freeda/freeda-x.x (the release)
- ln -s freeda-x.x freeda (soft link to create $USER/freeda/freeda)
- $USER/freeda/freeda/bin
- $USER/freeda/freeda/doc (documentation for this release)
- $USER/freeda/freeda/simulator (top of source code tree for fREEDA and ifREEDA)
- $USER/freeda/freeda/elements (top of source code tree for elements, used in generating element documentation)

These defaults can be overwritten by environment variables as discussed in Section 1.5.5

1.5.3 Setting up the Cygwin Environment

1.5.4 Setting Up .bash_profile

It is important that users set up the .bash_profile correctly. Here is a suitable `.bash_profile` script. Gets around the problem of spaces in the directory path. This should be added to the file .bash_profile in the login directory. This is where you will go when you launch cygwin.

```
USER="mbs"
USERNoSpaces="mbs"
export HOME="/$USER"
mount -f -s -b "C:/Documents and Settings/$USER" "/$USERNoSpaces"
export PATH="/$USERNoSpaces/freeda/bin:$PATH"
export HOMEPATH="/$USERNoSpaces"
$FREEDA_HOME="/$USERNoSpaces/freeda"
```

1.5.5 Environment Variables

Generally the defaults will be fine for freeda and ifreeda users. Environment variables are available to adapt to the local environment. These environment variables can be set in the .bash_profile file, see Section 1.5.4.

Environment Variable	Internal **Variable**	**Default**
FREEDA_HOME	env_freeda_home	$USER/freeda
FREEDA_LIBRARY	env_freeda_library	$FREEDA_HOME/library
FREEDA_PROJECTS	env_freeda_projects	$FREEDA_HOME/projects
FREEDA_PATH	env_freeda_path	$FREEDA_HOME/freeda
FREEDA_BIN	env_freeda_bin	$FREEDA_PATH/bin
FREEDA_SIMULATOR	env_freeda_simulator	$FREEDA_PATH/simulator
FREEDA_ELEMENTS	env_freeda_elements	$FREEDA_SIMULATOR/elements
FREEDA_DOCUMENTATION	env_freeda_documentation Documentation developers should set the variable to $FREEDA_PATH/doc	/tmp
FREEDA_WEB_DOCUMENTATION	env_freeda_web_documentation Documentation developers should set the variable to $FREEDA_PATH/doc	http://www.freeda.org/doc
FREEDA_BROWSER	env_freeda_browser default for CygWin, cygstart is not a browser but works this way in fREEDA as it uses the registry table to open the appropriate application for a file.	cygstart
FREEDA_BROWSER	env_freeda_browser default if not CygWin environment.	firefox

1.5.6 Known Bugs

A list of known simulator bugs is located in the file `README.bugs` in the `src/` directory. Known element model bugs are provided in the element documentation.

1.6 Help

If you need help contact one of the developers or send email to m.b.steer@ieee.org . Several groups use *f*REEDA™but these are early days for *f*REEDA™ so you may have issues that have not been addressed.

Chapter 2

Netlist Format

The netlist input of fREEDA is almost compatible to Spice. There are a number of additional features and these are documented below. The focus of the additions is to facilitate the addition of new models, allow variables, and support hierarchical descriptions of coupling in a network.

2.1 Structure of fREEDA^{TM}'s Netlist

The fREEDA netlist mainly consists in a title, an analysis specification, a list of connected elements[1], and a list of output commands.

2.1.1 Lexical

fREEDAs grammatical rules are very similar to those of spice:

whitespace blank
  a newline followed immediately by a + sign. a tab
  a vertical tab
  a newpage

identifier A character sequence beginning with an Alphabetic character

$$A - Za - z$$

variables A variable must begin with an alphabetic character or a $ followed by alphanumeric characters or '_' or '.'

  Example:

```
HEIGHT
\$height
```

[1]element: a model of a physical component of a network.

```
height.1_1
```

Note that `HEIGHT` and `height` are identical as case is not preserved.

strings Either as an identifier (a continuous sequence of alphanumeric characters or enclosed within double quotes.
The following special escaped characters are allowed in strings defined within double quotes.

`"`To include a double quote in a string.

`n`To indicate a newline

Examples:

```
gate
"VOLTAGE WAVEFORM"
```

Note: Strings may continue across lines using the Spice continuation syntax:

```
"VOLTAGE
+ WAVEFORM"
```

or simply by continuing across a line as in

```
"VOLTAGE
 WAVEFORM"
```

numbers "E" or "e" to indicate exponent.

dotted command A "." folowed by alphabetic characters

lf A line feed or cariage return.

Capitalization

The case of identifiers and keywords is ignored in fREEDA^{TM} netlists. The significance of case is retained only within quoted strings, and in that case it is always retained. Internally characters are mapped to lower case.

2.1.2 Continuation of Line

A line beginning with a plus sign is considered to be the continuation of the previous one.

2.1.3 Title Line

`*** Unit Cell Folded Slot Antenna ***`

As in Spice, the first line of the netlist file is the title and does not contain commands.

2.1.4 Comments

`* Local reference nodes`

As in Spice, comment lines begin with an asterisk.

2.1.5 .options

Used to set up quantities similar to spice syntax. The general syntax is

`.options` <*identifier*> `=` <*value*>

All identifiers set in a .options card are treated as a variable. *value* may be an number or a previously defined variable.

Preset Options

Some variables are preset:

variable	**default**	**value**
defl	OPTIONS_DEFAULT_DEFL	100×10^{-6}
defw	OPTIONS_DEFAULT_DEFW	100×10^{-6}
defad	OPTIONS_DEFAULT_DEFAD	0
defas	OPTIONS_DEFAULT_DEFAS	0
tnom	OPTIONS_DEFAULT_TNOM	27
numdgt	OPTIONS_DEFAULT_NUMDGT	4
cptime	OPTIONS_DEFAULT_CPTIME	1×10^{6}
limpts	OPTIONS_DEFAULT_LIMPTS	201
itl1	OPTIONS_DEFAULT_ITL1	40
itl2	OPTIONS_DEFAULT_ITL2	20
itl4	OPTIONS_DEFAULT_ITL4	10
itl5	OPTIONS_DEFAULT_ITL5	5000
reltol	OPTIONS_DEFAULT_RELTOL	0.001
trtol	OPTIONS_DEFAULT_TRTOL	7.0
abstol	OPTIONS_DEFAULT_ABSTOL	1×10^{-12}
chgtol	OPTIONS_DEFAULT_CHGTOL	0.01×10^{-12}
vntol	OPTIONS_DEFAULT_VNTOL	1×10^{-6}
pivrel	OPTIONS_DEFAULT_PIVREL	1×10^{-13}
gmin	OPTIONS_DEFAULT_GMIN	1×10^{-12}

(For the developer: the defaults are defined in spice.h)

Control Options

Under Construction.

2.1.6 .model

```
.model c_line tlinp4 ( z0mag=75.00 k=7 fscale=1.e10
+ alpha = 59.9 )
```

Each `.model` is a statement that associates a name (<*model name*>) with a list of parameter values (<*param-list*>). The parameter names given must be the names of parameters of the element specified after the model keyword. Thus, `alpha` and `z0mag` must be parameters of `tlinp4` in the above example.

Further, the values assigned to parameters must be of an appropriate type. The parser goes to some lengths to coerce types where the result is sensible (i.e., if you give an integer value "1" to a parameter of float type, the parameter will be assigned the floating-point value "1.0"). However, you can't assign string values to float parameters, or vice-versa.

The `.model` statements define the default characteristics of the different physical elements ("models") in a network.

The syntax is as for spice

`.model` <*model name*> <*type name*> `([`<*parameter name*> `=` <*value*>`]*)`

Here, <*model name* > is an identifier by which the model is referred to. <*type name*> is the physical element name that the model refers to. the *parameter name* must be a valid parameter for the element (indicated by <*type name*>) referred to.

Any number of models may be specified for a single element.

2.1.7 Analysis Specification

```
.ac start = 3.6GHz stop = 4.8GHz n_freqs = 7
```

This line consists in a dot followed by the analysis name and a list of parameters. See the analysis catalog for a list of analysis and the description of the paramenters.

Note that 4.8GHz is equivalent to 4.8e9 or 4.8g. This is the same as in Spice.

2.1.8 Element Specification

```
nport:cpw_2 10 20 100 200 filename = "unitcell.yp"
```

Elements are specified with the element type name (`nport` in this example) followed by a colon and the element instance name. Then a list of nodes (or terminals) to which the element is connected followed by a paramenter list. See the element catalog for a list of available elements and the description of the paramenters.

The terminals can be named using integer numbers or strings. When using strings, they must be enclosed in quotes.

Regular passive elements (R, L and C) also support the standard Spice syntax with the following additions common to all elements in fREEDA^{TM}:

1. A `.model` specification is allowed for all elements.

2. Anything that can appear in a `.model` specification can be included in the specification of the element.

3. If a parameter is not specified either through an element specification or a `.model` specification then the default parameters for that model will apply to this element.

2.1.9 End of Netlist

Every netlist must finish with a `.end` control card.

2.1.10 Subcircuits

The subcircuit definition and instantiation is pretty much as in Spice. The definition may appear after or before the instatiation in the netlist. Node names inside the subcircuit are local to the subcircuit. The following is an example for a three-terminal subcircuit.

```
...

.subckt era6 1 5 "gnd1"

... (subcircuit definition)

.ends

...

xamp1 10 50 0 era6

...
```

The name of the subcircuit instance must begin with `x`.

2.2 Output Control

fREEDA$^{\mathrm{TM}}$ has an interpretive output language which uses a reverse polish notation syntax. The operators operate on a stack and as an operation is performed zero or more arguements are consumed by an opertor. This is an extremely powerful way of controlling output.

Output Commands

`.out write ( (`<*qualifier*> <*value*>`* )*` <*operator*> `)* in` <*filename*>

or

.out plot ((<*qualifier*> <*value*>*)* <*operator*>)* [[<*gnuplotPostambleScript*>] <*gnuplotPreambleScript*>] in <*filename*>

or

.out system <*string*>

2.2.1 Writing

.out write ((<*qualifier*> <*value*>*)* <*operator*>)* in <*filename*>

The `write` command writes what is left on the stack into the file *filename*.

Example

```
.out write term 4 vt in "4v.out"
```

This writes the time domain voltage at terminal 4 using the file 4v.out as an output file.

2.2.2 Plotting

.out plot ((<*qualifier*> <*value*>*)* <*operator*>)* [[<*gnuplotPostambleScript*>] <*gnuplotPreambleScript*>] in <*filename*>

The `plot` command writes what is left on the stack into the file *filename* and initiates a plot. The file can be plotted interactively using the fREEDA™ Output Viewer. Also, a file named <*filename*>`.cmd` is created. This file is a gnuplot [24] script file that plots the desired data. The Scripts are optional strings and are used to send additional commands to the gnuplot program.

<*gnuplotPreambleScript*> is a string of semicolon delineated gnuplot commands prior to the plot command which is automatically issued.

<*gnuplotPostambleScript*> is a string of semicolon delineated gnuplot commands after the plot command.

If the option `gnuplot` is present in the `.options` card, the gnuplot program will be called automatically by fREEDA™. Note that this is generally not needed when using the Output Viewer.

Example

```
.out plot term 4 vt in "4v.out"
```

There are no script commands here. This plots the time domain voltage at terminal 4 using the file 4v.out as an output file. This functions as both a write and a plot.

2.2.3 Running a System Command

.out system <*string*>

Use this to run the string as a command to the operating system.

Example

```
.out system "echo Hello"
```

Prints “Hello” on the screen.

2.2.4 Nomenclature

The following nomenclature is used in describing the output operators.

type	**description**
scalar numeric types	
i	integer
f	floating-point
r	real (integer or floating-point)
c	complex
s	scalar (integer, floating-point or complex)
scalar and mixed numeric types	
fv	floating-point vector
cv	complex vector
v	floating-point or complex vector
fsv	floating-point scalar or vector
csv	complex scalar or vector
sv	scalar or vector (any)
prom	an appropriately-promoted numeric type
-x	(suffix to vector types) x data required
other types	
any	any type
string	character string
var	variable name
file	data file
func	function pointer

2.2.5 Qualifiers

type	**description**
qualifiers (network types)	
term	terminal (or node)
element	circuit element

2.2.6 Operators

General Operators

operator	function	argument(s)	result
`dup`	duplicate object	*any*	*same*
`get`	get element of vector	*arg:v* *index:i*	*s*
`put`	modify element of vector	*arg:v* *index:i* *val:s*	*v*
`stripx`	remove x data	*vx*	*v*
`pack`	concatenates last *vx*'s on stack	variable number of *vx*	*m*
`system`	execute shell command	*string*	*none*

2.2.7 Network Operators

`vf`	complex freq. domain voltage vector at a terminal	*term*	*cv*
`if`	complex freq. domain current vector at a terminal	*term*	*cv*
`xf`	complex freq. domain state variable vector at a terminal	*term*	*cv*
`vt`	time domain voltage vector at a terminal	*term*	*fv*
`it`	time domain current vector at a terminal	*term*	*fv*
`ut`	time domain voltage vector at an element port	*elem*	*fv*

RPN Arithmetic Operators

Arithmetic Operators for reverse polish notation e.g. 3 4 add = 7

`add`	addition	*sv*	*prom*
		sv	
`sub`	subtraction	*sv*	*prom*
		sv	
`mult`	multiplication	*sv*	*prom*
		sv	
`div`	division	*sv*	*prom*
		sv	
`real`	real part	*csv*	*fsv*
`imag`	imaginary part	*csv*	*fsv*
`mag`	magnitude	*csv*	*fsv*
`abs`	absolute value or magnitude	*sv*	*fsv*
`contphase`	continuous phase	*csv*	*fsv*
`prinphase`	principal value phase	*csv*	*fsv*
`conj`	complex conjugate	*csv*	*csv*
`neg`	additive inverse (negative)	*sv*	*sv*
`recip`	reciprocal	*sv*	*sv*

Mathematical Operators

`db`	dB ($20\log_{10}$)	*sv*	*fsv*
`db10`	dB applied to power ($10\log_{10}$)	*sv*	*fsv*
`rad2deg`	convert radians to degrees	*fsv*	*fsv*
`deg2rad`	convert degrees to radians	*fsv*	*fsv*
`minlmt`	limit the minimum value	*arg:fsv*	*fsv*
		min:f	
`maxlmt`	limit the maximum value	*arg:fsv*	*fsv*
		max:f	
`diff`	differences	*fsv*	*fsv*
`deriv`	derivative	*fsv*	*fsv*
`sum`	sums	*fsv*	*fsv*
`integ`	integral	*fsv*	*fsv*

Signal Processing Operators

`smpltime`	current analysis timebase as x *and* y of result	*none*	*fv*
`sweepfrq`	current analysis sweep frequencies as x *and* y of result	*none*	*fv*
`smplcvt`	interpolate *signal1* over timebase of *signal2*	*signal1:v* *signal2:vx*	*vx*
`sweepcvt`	interpolate *frq1* over sweep frequencies of *frq2*	*frq1:v* *frq2:vx*	*vx*
`maketime`	create timebase starting at $t = 0$ in x *and* y of result	*tmax:r* *pts:i*	*vx*
`makesweep`	create sweep frequencies starting at $f = 0$ in x *and* y of result	*fmax:r* *pts:i*	*vx*
`fft`	FFT (argument should have 2^k points)	*timedata:fv*	*cv*
`invfft`	inverse FFT (argument should have $2^k - 1$ points)	*frqdata:cv*	*fv*
`cconv`	real circular (FFT) convolution with zero padding	*signal1:fv* *signal2:fv*	*fv*
`upcconv`	unpadded real circular (FFT) convolution	*signal1:fv* *signal2:fv*	*fv*
`sconv`	slow (time-domain) real convolution	*signal1:fv* *signal2:fv*	*fv*
`fconv`	fast (approximate) real convolution	*signal1:fv* *signal2:fv*	*fv*
`lpbwfrq`	lowpass Butterworth filter frequency response	*frqvec:vx* *corner:f* *order:i*	*cvx*

2.3 Example: Simulation of a Folded Slot Antenna

The netlist format is illustrated using an example. This example uses local reference nodes. For a discussion of the local reference node concept see chapter ??. fREEDA™ provides the local references as a convenience tool, but it is still possible to treat circuits in a conventional way using the node "0" or "`gnd`" as a global reference.

As a component of a spatial power combining circuit the CPW folded-slot antenna [37], see Fig. 2.1, with polarizers transmits an amplified version of an incident propagating field. In Fig. 2.1 the two orthogonal folded-slots are connected to each other by a CPW with an inserted MMIC amplifier.The system is modeled using the circuit of Fig. 2.2 where electromagnetic modeling of this structure is discussed in [38, 39, 40]. Note that three different local reference nodes are required. EM modeling yields a port-based y parameters of the antennas at each frequency of interest. The transfer of data between the EM and circuit simulators (typically a file) includes a header with port grouping information (a port grouping includes terminals associated with a specific local reference node). This is required by

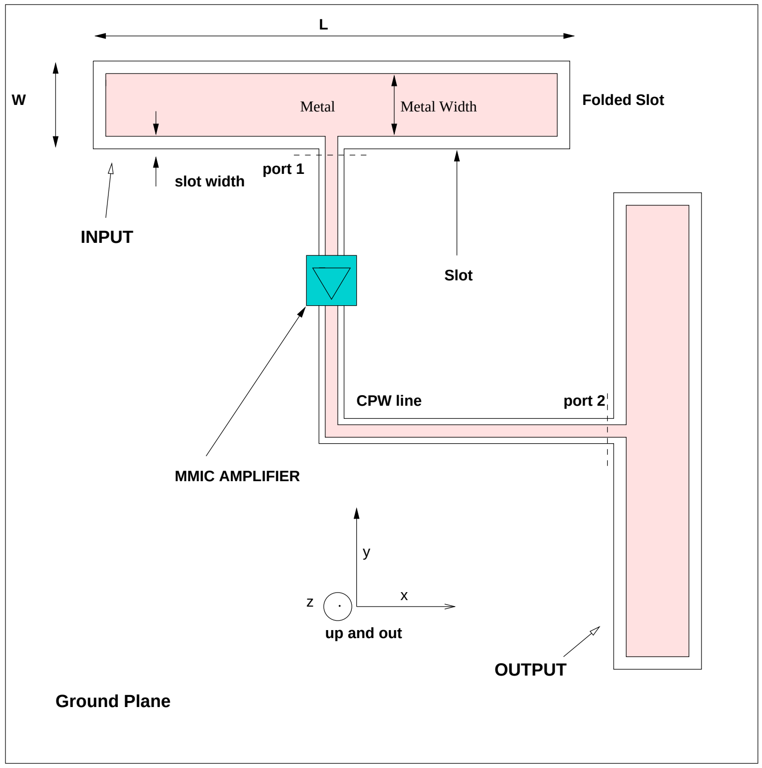

Figure 2.1: Unit cell of the CPW antenna array.

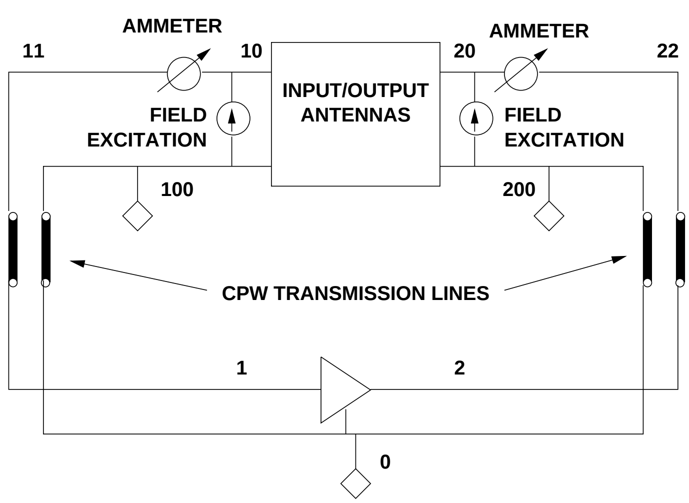

Figure 2.2: Circuit model of the unit cell. The diamond symbol indicates a local reference node.

the circuit simulator in order to expand the port-based matrix into nodal form and also to check the connectivity of the spatially-distributed circuit.

Below is shown the data file for this example. Each port belong to a different group, so the element has four terminals. After reading the header the circuit simulator knows the number of elements of the matrix and the port number and local reference corresponding to each row and column of the matrix.

```
# port:group
1:1
1:2
# GHZ Y RI R 50
4
0.00355603      -0.0233196
-0.00121905     -0.00496212
-0.00121905     -0.00496212
0.00355603      -0.0233196
...
```

The rest of the file consist in a list of frequencies and the associated matrix elements in complex form.

The parser provides several facilities such as the `.model` statement support for any element type and a complete reverse polish notation calculator for the output data. The corresponding netlist is shown below.

```
*** Unit Cell Folded Slot Antenna ***

.ac start = 3.6GHz stop = 4.8GHz n_freqs = 7

* Local reference nodes
.ref 100
.ref 200

* CPW structure
nport:cpw_2 10 20 100 200 filename = "unitcell.yp"

* Transistor small signal model
nport:amplifier 1 2 0 filename = "feedback_ne3210s1.yp"

* Field excitation
gridex:iin 10 100 20 200 ifilename = "unitcell.i"
+ efilename = "dummy.e"

* current meters
vsource:amp1 10 11
vsource:amp2 20 22
```

```
* CPW Transmission lines
.model fsa1 cpw (s=.369m w=1m t=10u er=10.8 tand=.001)
cpw:t1 11 100 1 0 model="fsa1" length=8.5m
cpw:t2 22 200 2 0 model="fsa1" length=17.5m

.out write element "vsource:amp1" 0 if in "i1.out"
.out write element "vsource:amp2" 0 if in "i2.out"

* Plot magnitude of current gain
.out plot element "vsource:amp2" 0 if element
+ "vsource:amp1" 0 if div mag db in "igain.out"

* Plot magnitude of voltage gain
.out plot term 20 vf term 10 vf div mag db in "gain.out"

.end
```

Chapter 3

Algebraic Expressions

This page is still under development so there may be problems. The big one is that expressions do not work at all now.

Most places where a numeric value is normally used an expression (within braces { ... }) can be used instead. An expression can contain any supported mathematical operation, constant numeric values, parenthesees "(...)" to indicate precedence, commas "," to separate arguments of a function, or parameters. Valid parameters must begin with an alphabetic character. Places where expressions cannot be used are

- As polynomial coefficients.
- The values of the transmission line device parameters `NL` and `F`.
- The values of the piece-wise linear characteristic in the `PWL` form of the independent voltage (`V`) and current (`I`) sources.
- The values of the resistor device parameter `TC`.
- As node numbers.

and

- Values of most statements (such as .TEMP, .AC, .TRAN etc.)

Specifically included are

- The values of all other device parameters.
- The values in `.IC` and `.NODESET` statements.
- The values in `.SUBCKT` statements.

and

- The values of all model parameters.

Table 3.1: Expression operators.

Operator Name	**Precedence Index**	**Syntax**	**Description**
Arithmetic Operators			
UNARY_PLUS	10	+x	unary plus
UNARY_MINUS	10	-x	unary minus
POWER	9	x^y or x**y	raise to a power, x^y
MULTIPLY	8	x*y	multiply
DIVIDE	8	y/x	divide
PLUS	7	x+y	plus
MINUS	7	x-y	minus
Logical Operators			
NOT	10	!x	NOT
GREATER_OR_EQUAL	6	$x >= y$	greater than or equal
LESS_OR_EQUAL	6	$x <= y$	less than or equal
GREATER_THAN	6	$x > y$	greater than
LESS_THAN	6	$x < y$	less than
EQUAL	5	$x == y$	equality
NOT_EQUAL	5	$x! = y$	no equal
AND	4	x&y	logical and
OR	3	x|y	logical or
XOR	2	$x\ \ y$	exclusive or

Operators that can be used in expressions are listed in Table 3.1. Here x and y maybe numbers, parameters or sub-expressions. The result of the logical operations is either 0 or 1. Operands are treated as 1 if they are not exactly zero. The precedence of the operators is also given in Table 3.1 and follows normal practice. A higher precedence number indicates higher prcedence and operators of the same precedence are evaluated from left to right. For example

`idss* ( vgs + vgs^2 )`	is evaluated as $\texttt{idss} * (\texttt{vgs} + (\texttt{vgs}^2))$ where `idss` and `vgs` are parameters defined elsewhere.
`3 + 5 -- 4||(3^-4)>=23`	is evaluated as $(((3+5)-(-4))\|\|((3^{(}-4)) >= 23))$

Functions that can be used in expressions are listed in Table 3.2.

Note:

1. The table look up function returns a y value given an x value and a set of (x,y) points defining piece-wise linear. The number of $x - y$ pairs in the table function is limited approximately to 100. A further limit is imposed by the amounjt of information thatmustbe retained during expression evalurion. To obtain the full 100 print capability in a complicated expression it may be necessary to use an intermediate variable. expre

Table 3.2: Expression functions.

Operator	**Syntax**	**Description**
SIN	`sin`(x)	sine, argument in radians
COS	`cos`(x)	cosine, argument in radians
TAN	`tan`(x)	tangent, argument in radians
ASIN	`asin`(x) or `arcsin`(x)	arcsine, result in radians
ACOS	`acos`(x) or `arcsin`(x)	arccosine, result in radians
ATAN	`atan`(x) or `arcsin`(x)	arctangent, result in radians
SINH	`sinh`(x)	hyperbolic sine
COSH	`cosh`(x)	hyperbolic cosine
TANH	`tanh`(x)	hyperbolic tangent
EXP	`exp`(x)	exponentiation, e^x
ASINH	`asinh`(x) or `arcsinh`(x)	arc-hyberbolic sine
ACOSH	`acosh`(x) or `arccosh`(x)	arc-hyberbolic cosine
ATANH	`atanh`(x) or `arctanh`(x)	arc-hyberbolic tangent
ABS	`abs`(x)	absolute, $\|x\|$
SQRT	`sqrt`(x)	square root, $\sqrt{x}$
LN	`ln`(x)	log to base e of x
LOG	`log`(x) or log10(x)	log to base 10 of x
SIGN	`sign`(x)	sign of $x = \begin{cases} 1 & \text{if } x >= 0 \\ -1 & \text{if } x < 0 \end{cases}$
DB	`db`(x)	decibell = 10 log(x)
LIMIT	`limit`(x, min, max)	limit $= \begin{cases} min & \text{if } x < min \\ max & \text{if } x < max \\ x & \text{otherwise} \end{cases}$
TABLE	`table`($x, x_1, y_1, ..., x_n, y_n$)	table lookup, see note 1
DUPLICATE	`dup`(x)	duplicates x, see note 2
IF	`if`(x, y, z)	conditional, $= \begin{cases} y & \text{if } x \text{ not zero} \\ z & \text{if } x \text{ is zero} \end{cases}$

1. The dup() function duplicates an operand. It provides a means to use a sub-expression twice while only evaluating it once. For example

Operation	Expansion	
(dup(x)+)	$\longrightarrow$	$x + x$
(dup(dup(x))+*)	$\longrightarrow$	$(x + x) * x$
limit(dup(x),max)	$\longrightarrow$	$\begin{cases} x & \text{if } x < \text{max} \\ \text{max} & \text{if } x > \text{max} \end{cases}$
if(dup(dup(x))¿0, *2,*3)	$\longrightarrow$	$\begin{cases} 2x & \text{if } x > 0 \\ 3x & \text{if } x < 0 \end{cases}$

It is good practice to enforce precedence by using parentheses. That is, use (`dup(`x`)`+) rather than `dup(`x`)`+ .

Chapter 4

fREEDA Commands

4.1 .inc Include Statement

The `.inc` statement is an efficient way to include subcircuits and common netlist code.
Form

`.lib` [*FileName*]

Filename is the name of the file to be read.

Note

1. It must contain only `.model` statements, subcircuit definitions (between `.subckt` and `.ends` statements), and `.lib` statements.

2. The include file *Filename* is searched in the current directory.

   `.INC` and the `.LIB` function similarly with the exception that .LIB searches for the file in a specific directory while .INC searches for the file in current directory.

4.2 .lib Library Statement

The `.lib` statement is an efficient way to include `.model` statements and subcircuits.
Form

`.lib` [*FileName*]

Filename is the name of the library file.

Note

1. It must contain only `.model` statements, subcircuit definitions (between `.subckt` and `.ends` statements), and `.lib` statements.
2. The library file *Filename* is searched in the directory pointed to by the environment variable specified by the environment variable FREEDA_LIBRARY .

Libraries could be included using either the `.INC` statement or by the `.LIB` statement. `.INC` and the `.LIB` function similarly with the exception that .LIB searches for the file in a specific directory while .INC searches for the file in current directory.

4.3 .locate Identify Location of Terminals

Form:
`.locate ⟨ terminal name⟩ ⟨ X⟩ ⟨ Y⟩`

Description:
.LOCATE is used to identify the position of a terminal.

Credits:

Name	Affiliation	Date	Links
Michael Steer	NC State University	Sept 2000	NC STATE UNIVERSITY
m.b.steer@ieee.org			www.ncsu.edu

4.4 .plot Plot Specification

NOT FULLY FUNCTIONAL as of *f*REEDA™1.3

The plot specification controls the information that is plotted as the result of various analyses.

Form: Form

`.PLOT TRAN` *OutputSpecification* [*PlotLimits*]
`+` [*OutputSpecification* [*PlotLimits*] ...]

`.PLOT AC` *OutputSpecification* [*PlotLimits*]
`+` [*OutputSpecification* [*PlotLimits*] ...]

`.PLOT DC` *OutputSpecification* [*PlotLimits*]
`+` [*OutputSpecification* [*PlotLimits*] ...]

`.PLOT NOISE` *NoiseOutputSpecification* [`(`*DistortionReportType*`)`]
[*PlotLimits*]
`+` [*NoiseOutputSpecification* [`(`*DistortionReportType*`)`] [*PlotLimits*]

`.PLOT DISTO` *DistortionOutputSpecification* [`(`*DistortionReportType*`)`]
[*PlotLimits*]
`+` [*DistortionOutputSpecification* [`(`*DistortionReportType*`)`] [*PlotLimits*] ...] [`(`*DistortionReportType*`)`] [*PlotLimits*]

`tran` is the keyword specifying that this `.plot` statement controls the reporting of results of a transient analysis initiated by the `.TRAN` statement.

`ac` is the keyword specifying that this `.plot` statement controls the reporting of results of a small-signal `AC` analysis initiated by the `.ac` statement.

`dc` is the keyword specifying that this `.plot` statement controls the reporting of results of a `DC` analysis initiated by the `.dc` statement.

`noise` is the keyword specifying that this `.plot` statement controls the reporting of results of a noise analysis initiated by the `.noise` statement.

OutputSpecification specifies the voltage or current to be plotted against the sweep variable. The sweep variable is dependent on the type of analysis.

Voltages may be specified as an absolute voltage at a terminal: `V(`*TerminalName*`)` or the voltage at one terminal with respect to that at another terminal, e.g. `V(Terminal1Name,Terminal2Name)`.

For the reporting of the results of an `AC` analysis the following outputs can be specified by replacing the `V` as follows:

`VR`	-	real part
`VI`	-	imaginary part
`VM`	-	magnitude
`VP`	-	phase
`VDB`	-	$10 \log(10\, magnitude)$

In `AC` analysis the default is `VM` for magnitude.

<u>Currents</u> are specified by referencing the name of the voltage source through which the current is measured, e.g. `I(V`*VoltageSourceName*`)`.
For the reporting of the results of an `AC` analysis the following outputs can be specified by replacing the `I` as follows:

`IR`	-	real part
`II`	-	imaginary part
`IM`	-	magnitude
`IP`	-	phase
`IDB`	-	$10 \log(10\, magnitude)$

In `AC` analysis the default is `IM` for magnitude.

PlotLimits are optional and can be placed after any output specification. *PlotLimits* has the form `(`*LowerLimit*`,` *UpperLimit*`)` . All quantities will be plotted using the same *PlotLimits*. The default is to automatically scale the plot and perhaps use different scales for each of the quantities to be plotted.

NoiseOutputSpecification specifies the noise measure to be reported. The two options are `ONOISE` which reports the output noise and `INOISE` which reports the equivalent input noise. See the `.NOISE` statement for a detailed explanation.

It must be one of the following:

`ONOISE`	-	magnitude of the output noise
`DB(ONOISE)`	-	output noise in dB
`INOISE`	-	magnitude of the equivalent input noise
`DB(INOISE)`	-	equivalent input noise in dB
`GAIN`	-	voltage gain
`DB(GAIN)`	-	voltage gain in dB (= 20 log(`GAIN`)
`GT`	-	transducer gain
`DB(GT)`	-	transducer gain in dB (= 10 log(`GT`)
`NF`	-	spot noise factor
`DB(NF)`	-	spot noise figure (= 10 log(`NF`)
`SNR`	-	output signal-to-noise ratio
`DB(SNR)`	-	output signal-to-noise ratio in dB (= 20 log(`SNR`)
`TNOISE`	-	output noise temperature.

SParameterOutputSpecification specifies the S-parameter output variables that are to be printed. Each variable must have one of the following forms:

`S(i,j)`	-	Magnitude of S_{ij}
`SR(i,j)`	-	Real part of S_{ij}
`SI(i,j)`	-	Imaginary part of S_{ij}
`SP(i,j)`	-	Phase of S_{ij} in degrees
`SDB(i,j)`	-	Magnitude of S_{ij} in dB (= 20 log(`S(i,j)`))
`SG(i,j)`	-	Group delay of S_{ij}

The port numbers are i, j which are specified using the `PNR` keywor when the port ('P') element is specified.

DistortionOutputSpecification specifies the distortion component to be reported in a tabular format of up to 8 columns plus an initial column with the sweep variable. The *DistortionOutputSpecification* is one of the following:

`HD2`	-	the second harmonic distortion
`HD3`	-	the second harmonic distortion
`SIM2`	-	the sum frequency intermodulation component
`DIM2`	-	the difference frequency intermodulation component
`DIM3`	-	the third order intermodulation component

See the `.DISTO` statement for a description of these distortion components.

DistortionReportType specifies the format for reporting the distortion components. It must be one of the following:

R	-	real part
I	-	imaginary part
M	-	magnitude
P	-	phase
DB	-	$10\log(10\,magnitude)$

The default is `M` for magnitude.

Note

1. There can be any number of `.PLOT` statements.

2. All of the output quantities specified on a single `.PLOT` statement will be plotted on the same graph using ASCII characters. An overlap will be indicated by the letter `X`. The plot produced by the `.PLOT` statement is a line printer plot. While plotting is primitive it can be plotted on any printer and is incorporated in the output log file.

3. The plot output of the results of an `AC` analysis always have a logarithmic vertical scale.

4.5 .print Print Specification

NOT FULLY FUNCTIONAL as of *f*REEDA™1.3

Form:

The print specification controls the information that is reported as the result of various analyses.

Form

`.print TRAN` *OutputSpecification* [*OutputSpecification* ...]

`.print AC` *OutputSpecification* [*OutputSpecification* ...]

`.print DC` *OutputSpecification* [*OutputSpecification* ...]

`.print DISTO` *DistortionOutputSpecification* `(` *DistortionReportType*
`)`
`+` [*DistortionOutputSpecification* `(` *DistortionReportType* `)` ...]

`TRAN` is the keyword specifying that this `.print` statement controls the reporting of results of a transient analysis initiated by the `.TRAN` statement.

`AC` is the keyword specifying that this `.print` statement controls the reporting of results of a small-signal `AC` analysis initiated by the `.AC` statement.

`DC` is the keyword specifying that this `.print` statement controls the reporting of results of a `DC` analysis initiated by the `.DC` statement.

`NOISE` is the keyword specifying that this `.print` statement controls the reporting of results of a noise analysis initiated by the `.NOISE` statement.

`DISTO` is the keyword specifying that this `.print` statement controls the reporting of results of a small-signal `AC` distortion analysis initiated by the `.DISTO` statement.

OutputSpecification specifies the voltage or current to be reported in a tabular format of up to 8 columns plus an initial column with the sweep variable.

Voltages may be specified as an absolute voltage at a node: `V(`*TerminalName*`)` or the voltage at one node with respect to that at another node, e.g. `V(Terminal1Name,Terminal2Name)`.
For the reporting of the results of an `AC` analysis the following outputs can be specified by replacing the `V` as follows:

`VR`	-	real part
`VI`	-	imaginary part
`VM`	-	magnitude
`VP`	-	phase
`VDB`	-	$10 \log(10\, magnitude)$

In `AC` analysis the default is `VM` for magnitude.

Currents are specified by referencing the name of the voltage source through which the current is measured, e.g. `I(V`*VoltageSourceName*`)`.
For the reporting of the results of an `AC` analysis the following outputs can be specified by replacing the `I` as follows:

`IR`	-	real part
`II`	-	imaginary part
`IM`	-	magnitude
`IP`	-	phase
`IDB`	-	$10 \log(10\, magnitude)$

In `AC` analysis the default is `IM` for magnitude.

NoiseOutputSpecification specifies the noise measure to be reported. The two options are `ONOISE` which reports the output noise and `INOISE` which reports the equivalent input noise. See the `.NOISE` statement for a detailed explanation.
It must be one of the following:
`ONOISE` - RMS output noise voltage
`DB(ONOISE)` - output noise voltage in dB (= 20 log(`ONOISE`)
`INOISE` - RMS equivalent input noise voltage
`DB(INOISE)` - equivalent input noise voltage in dB (= 20 log(`INOISE`)
`GAIN` - voltage gain
`DB(GAIN)` - voltage gain in dB (= 20 log(`GAIN`)
`GT` - transducer gain
`DB(GT)` - transducer gain in dB (= 10 log(`GT`)
`NF` - spot noise factor
`DB(NF)` - spot noise figure (= 10 log(`NF`)
`SNR` - output signal-to-noise ratio
`DB(SNR)` - output signal-to-noise ratio in dB (= 20 log(`SNR`)
`TNOISE` - output noise temperature.

SParameterOutputSpecification specifies the S-parameter output variables that are to be printed. Each variable must have one of the following forms:
`S(i,j)` - Magnitude of S_{ij}
`SR(i,j)` - Real part of S_{ij}
`SI(i,j)` - Imaginary part of S_{ij}
`SP(i,j)` -Phase of S_{ij} in degrees
`SDB(i,j)` - Magnitude of S_{ij} in dB (= 20 log(`S(i,j)`))
`SG(i,j)` - Group delay of S_{ij}

The port numbers are i, j which are specified using the `PNR` keyword when the port element is specified.

DistortionOutputSpecification specifies the distortion component to be reported in a tabular format of up to 8 columns plus an initial column with the sweep variable. The *DistortionOutputSpecification* is one of the following:

`HD2`	-	the second harmonic distortion
`HD3`	-	the second harmonic distortion
`SIM2`	-	the sum frequency intermodulation component
`DIM2`	-	the difference frequency intermodulation component
`DIM3`	-	the third order intermodulation component

See the `.DISTO` statement for a description of these distortion components.

DistortionReportType specifies the format for reporting the distortion components. It must be one of the following:

R	-	real part
I	-	imaginary part
M	-	magnitude
P	-	phase
DB	-	$10 \log(10\, magnitude)$

The default is `M` for magnitude.

Note

1. There can be any number of `.print` statements.

2. The number of significant digits of the results reported is `NUMDGT` which is set in a `.options` statement.

`dc` and `tran` Reporting

The output specifications available for the `DC` sweep and transient analyses are

`I(`*DeviceName*`)` Current through a two terminal device (such as a resistor `R` element) or the output of a controlled voltage or current source. e.g. `I(R22)` is the current flowing through resistor `R22` from node N_1 to N_2 of `R22`.

`I` TerminalName`(`*DeviceName*`)` Current flowing into terminal named *TerminalName* (such as `B` for gate) from the device named *DeviceName* (such as `Q12`). e.g. `IB(Q12)`

`I` PortName`(`*TransmissionLineName*`)` Current at port named *PortName* (either `A` or `B`) of the transmission line device named *TransmissionLineName*

`V(`*TerminalName*`)` Voltage at a node of name *TerminalName*.

`V(`n_1, n_2`)` Voltage at node n_1 with respect to the voltage at node n_2.

`V(`*DeviceName*`)` Voltage across a two terminal device (such as a resistor `R` element) or at the output of a controlled voltage or current source.

`V` TerminalName`(`*DeviceName*`)` Voltage at terminal named *TerminalName* (such as `G` for gate) of the device named *DeviceName* (such as `M12`). e.g. `VG(M12)`

`V` TerminalName1 TerminalName2`(`*DeviceName*`)` Voltage at terminal named *TerminalName1* (such as `G` for gate) th respect to the terminal name *TerminalName2* (such as `S` for source) of the device named *DeviceName* (such as `M12`). e.g. `VGS(M12)`

`V` PortName`(`*TransmissionLineName*`)` Voltage at port named *PortName* (either `A` or `B`) of the transmission line device named *TransmissionLineName* (such as `T5`). e.g. `VA(M5)`

Two Terminal Device Types Supported for `DC`and Transient Analysis Reporting

The single character identifier for the following elements as well as the rest of the device name can be used as the *DeviceName* in the `I(`*DeviceName*`)` and `I(`*DeviceName*`)` output specifications.

Element Type	Description
`C`	capacitor
`D`	diode
`E`	voltage-controlled voltage source
`F`	current-controlled current source
`G`	voltage-controlled current source
`H`	current-controlled voltage source
`I`	independent current source
`L`	inductor
`R`	resistor
`V`	independent voltage source

Multi-Terminal Device Types Supported for `DC`and Transient Analysis Reporting

The single character identifier for the following elements as well as the rest of the device name can be used as the *DeviceName* in the `I` TerminalName`(`*DeviceName*`)`, `V` TerminalName`(`*DeviceName*`)` and `V` TerminalName1 TerminalName2`(`*DeviceName*`)` output specifications.

Element Type	Description
`B`	GaAs MESFET Terminals: D — drain G — gate S — source
`J`	JFET Terminals: D — drain G — gate S — source
`M`	MOSFET Terminals: B — bulk or substrate D — drain G — gate S — source
`Q`	BJT Terminals C — collector B — base E — emitter S — source

AC Reporting

The output specifications available for reporting the results of an AC frequency sweep analysis includes all of the specification formats discussed above for DC and transient analysis together with a number of possible suffixes:

DB	-	$10\log(10\,magnitude)$
M	-	magnitude
P	-	phase
R	-	real part
I	-	imaginary part
G	-	group delay $= \partial\phi/\partial f$ where ϕ is the phase of the quantity being reported and f is the analysis frequency.

In AC analysis the default suffix is M for magnitude.

Two-Terminal Device Types Supported for AC Reporting

The single character identifier for the following elements as well as the rest of the device name can be used as the *DeviceName* in the I(*DeviceName*) and I(*DeviceName*) output specifications.

Element Type	Description
C	capacitor
D	diode
I	independent current source
L	inductor
R	resistor
V	independent voltage source

Multi-Terminal Device Types Supported for DCand Transient Analysis Reporting

The single character identifier for the following elements as well as the rest of the device name can be used as the *DeviceName* in the I TerminalName(*DeviceName*), V TerminalName(*DeviceName*) and V TerminalName1 TerminalName2(*DeviceName*) output specifications.

Element Type	Description
`B`	GaAs MESFET Terminals: D — drain G — gate S — source
`J`	JFET Terminals: D — drain G — gate S — source
`M`	MOSFET Terminals: B — bulk or substrate D — drain G — gate S — source
`Q`	BJT Terminals C — collector B — base E — emitter S — source

Credits:

Name	Affiliation	Date	Links
Michael Steer	NC State University	Sept 2000	NC STATE UNIVERSITY
m.b.steer@ieee.org			www.ncsu.edu

4.6 Structure of a fREEDA Netlist

There are four types of elements used in TRANSIM: [1] nodes, edges, edge coupling groups (ECGs) and node coupling groups (NCGs). Within those broad classifications there are a wide variety of individual element types, for example, “mlin” (microstrip line), “coax” (coaxial cable), and “idealj” (ideal junction). “element” and “model” are used synonomously.

4.6.1 Lexical Rules

A lexical rule defines an identifiable object in the input file. That is it defines the equivalent of words. Words put togetehr in a particular order define a grammar. fREEDA recognizes many ”words” but the important one are as follows.

whitespace a blank
a newline followed immediately by a + sign.
a tab
a vertical tab
a newpage

identifier A character sequence beginning with an alphabetic character

$$A - Za - z$$

variables A variable must begin with an alphabetic character or a $ followed by alphanumeric characters or ‘_’ or ‘.’
Example:
`HEIGHT`
`$height`
`height.1_1`
Note that `HEIGHT` and `height` are identical as case is not preserved.

[1]element: a model of a physical component of a network.

strings Either as an identifier (a continuous sequence of alphanumeric characters) or enclosed within double quotes.

The following special escaped characters are allowed in strings defined within double quotes.

`"` To include a double quote in a string.

`\n` To indicate a newline

Examples:

```
gate
"VOLTAGE WAVEFORM"
```

Note: Strings may continue across lines using the continuation syntax:

```
"VOLTAGE
+ WAVEFORM"
```

or simply by continuing across a line as in

```
"VOLTAGE
WAVEFORM"
```

numbers] "E" or "e" to indicate exponent.

dotted command A "." followed by alphabetic characters at the beginning of a line.

lf A line feed or carriage return.

Capitalization

The case of identifiers and keywords is ignored in TRANSIM netlists. The significance of case is retained only within quoted strings, and in that case it is always retained. Internally characters are mapped to lower case.

4.7 SPICE Elements

All regular SPICE elements have the same syntax as in standard SPICE but with the following additions.

1. A `.model` specification is allowed for all elements.
2. Anything that can appear in a `.model` specification can be included in the specification of the element.
3. If a parameter is not specified either through an element specification or a `.model` specification then the default parameters for that model will apply to this element.

<term_id> is either an integer or a string in double quotes, and is the name of a terminal in the network. <term_id_list> is a list of one or more terminal id's separated by white space.

4.8 General File Comments

The first line of an input file is used as the identifier string and is associated with various output files to identify their origin. It is seen strictly as a text string and no processing is done on it. If a particular statement won't fit on a single line, it may be continued by placing a "+" at the beginning of each additional line. All comments are proceeded by an "*" (an asterisk) and there is no limit to the number of comment lines used in a file. A comment may begin anywhere on a line (such as after a statement) and any text after the asterisk is ignored by the parser.

4.9 Element Instance Syntax

Each instance of an element in TRANSIM netlist is declared in the same manner with each declaration existing on a separate line. The syntax is:

```
element:instance_id term1 term2....  model = "identifier"
```

The terms *element* and *identifier* are the same as those used in the description of the *.model* statement and *instance_id* is a unique string that identifies this instance of *identifier*. *term1*, *term2*, etc. are the terminal specifiers which maybe a string or numeric values.

4.10 Netlist Variables

Local variables for use inside a netlist may be set with the *.options* command using the same syntax as used to set system variables. For example

```
.options logic1 = 5.0
.options logic0 = 0.6
.options  vdiff = logic1 - logic2
```

These local variables do not need to be declared before being set but they must be set before being used. Local variables are designed so that common parameters (such as microstrip width) may be declared in each *.model* statement as a variable with the variables value set once at the top of the netlist. Changing width requires changing one variable rather than multiple declarations in different *.model* statements. The third *.options* statement above illustrates the use of mathematical operations on local variables, in this case the difference between **logic1** and **logic0** is assigned to **vdiff**. Various analyses rely on variables set in a

4.11 .couple — Couple Elements

NOT FUNCTIONAL in current version Form:
`.couple` ⟨ `element instance name`⟩ `...` ⟨ `element instance name`⟩ ⟨ `terminal name`⟩ ⟨ `terminal name`⟩

Description:
.couple is used to identify the elements that combine to create a coupled element.

This command is used to indicate which edges (or nodes) to be simulated as coupled lines. The syntax is:

```
.couple line_1 line_2 line_3....
```

where *line_1* etc. are the specific names given to each instance of a line (or node). Note that the type of model used for coupled edges or nodes must be able to handle coupling. In general, a single line or node that may also be coupled is just a subset of the coupled line case. In other words, if a coupled line model (such as **cmlin**) is specified as the line model and the *.couple* statement is not used, then the simulator will default to using the uncoupled model (in this case **mlin**). This is not a runtime option but is fixed inside the code modules for each model.

4.11.1 .locate — Identify Location of Terminals

This command is used to define the physical location or a terminal. These cartesian coordinates refer to the locations of the "logical" terminals of the device. The units are meters. The syntax is:

```
.locate term x y
.locate term x y z
```

where `term` is one of the terminals of a device in the netlist and `x`, `y` and `z` (if provided) are the coordinates of that terminal. Be default `z`=0.

Credits:

Name	Affiliation	Date	Links
Michael Steer	NC State University	Sept 2000	
m.b.steer@ieee.org			www.ncsu.edu

4.12 .model

Description:
.model is used to identify the elements that combine to create a coupled element.

The syntax of the *.model* statement is:

```
.model identifier element (par1 par2 ...)
```

- (*identifier*) is any character string name assigned by the user by which this particular model will be referred.
- (*model_type*) is the model name as defined in the .c file associated with this model and as declared in pd_physdef.c.
- (par1 par2...) is the parameter list.

The *.model* statement must be used before it is referred to in the netlist. All fREEDA elements can utilize a *.model* statement.

4.13 .options

Description:
.options is used to identify the elements that combine to create a coupled element.

This command allows various runtime options and user defined netlist variables to be set prior to execution. The various system options will be discussed later in this appendix but the general syntax is:

```
.options variable = value
.options variable = "string"
```

The first case is used for assigning a numeric value to a variable and the second is used to assign a string. Note that double quote marks ("...") must be used to surround the string. Not typecasting of numeric variables is performed in the .options command and thus no distinction is made between floating point and integer values. Therefore 2 is the same as 2.00 until the value is actually used in the simulator. Exponential notation is denoted by the "e" operator (i.e. 0.001 = 1.0e-3). Note that string variables may contain any symbols but must be continuous with no white space between characters (i.e. "V_high" not "V high").

4.14 .out

Form:
.out write ([[<*qualifier*>] <*value*>*] <*operator*>)* in <*filename*>
This write what is left on the stack into the file *filename* or
.out system ([[<*qualifier*>] <*value*>*] [<*operator*>])* This performs a system call of the string equivalent of whatever is left on the stack.

.out ⟨ terminal name⟩ ⟨ terminal name⟩ ⟨ terminal name⟩ ⟨ terminal name⟩

Description:
.out is used to identify the elements that combine to create a coupled element.

The *.out* command is used to process and output data resulting from a fREEDA run. The *.out* statement uses stacks and has a syntax much like a reverse polish notation calculator. It is a powerful output engine and can be utilized in its own right or in conjunction with the more usual *.print* and *.plot* statements although these provide much less functionality. A variety of signal processing functions including arithmetic operators may be used to manipulate the data prior to writing it to a file, plotting to the screen or piping it to a system call.

fREEDA has an interpretive output language which uses a reverse polish syntax. The operators operate on a stack and as an operation is performed zero or more arguments are consumed by an operator.

Details of the various options will be shown at the end of this section but for most situations and netlists, the voltages and currents at the various external ports are to be written to output files in standard ASCII format. An example is shown below:

```
.out write term 1 vt in "1v.out"
.out write term 2 it in "2i.out"
```

In the first example, the voltage at terminal 1 is written out to file “1v.out”. The second example writes the current going *into* terminal 2 to the file “2i.out”.

4.14.1 Qualifiers

type	**description**
qualifiers (network types)	
term	terminal reference
junct	junction (node) reference (NOT CURRENTLY AVAILABLE)
line	line (edge) reference (NOT CURRENTLY AVAILABLE)

4.14.2 Nomenclature

The following nomenclature is used in describing the output operators.

type	**description**
scalar numeric types	
i	integer
f	floating-point
r	real (integer or floating-point)
c	complex
s	scalar (integer, floating-point or complex)
scalar and mixed numeric types	
fv	floating-point vector
cv	complex vector
v	floating-point or complex vector
fsv	floating-point scalar or vector
csv	complex scalar or vector
sv	scalar or vector (any)
prom	an appropriately-promoted numeric type
-x	(suffix to vector types) x data required
other types	
any	any type
string	character string
var	variable name
file	data file
func	function pointer

4.14.3 Operators

General Operators

operator	function	argument(s)	result
`dup`	duplicate object	*any*	*same*
`get`	get element of vector	*arg:v* *index:i*	*s*
`put`	modify element of vector	*arg:v* *index:i* *val:s*	*v*
`stripx`	remove x data	*vx*	*v*
`shell`	execute shell command (UNIX ENVIRONMENT ONLY)	*string*	*none*

Network Operators

operator	function	argument(s)	result
`v`	complex voltage vector at a terminal	*term*	*cv*
`i`	complex current vector at a terminal	*term*	*cv*
`vt`	transient voltage vector at a terminal	*term*	*fv*
`it`	transient current vector at a terminal	*term*	*fv*
`zl`	load impedance at a terminal (NOT CURRENTLY SUPPORTED)	*term*	*cv*
`ymelem`	element of the y-parameter matrix of a junction (NOT CURRENTLY SUPPORTED)	*junct* *row:i* *col:i*	*cv*
`z0`	characteristic impedance of a line	*line* (NOT CURRENTLY SUPPORTED)	*cv*
`gamma`	complex attenuation of a line	*line* (NOT CURRENTLY SUPPORTED)	*cv*
`yp`	admittance parameter of two terminals	*term* (NOT CURRENTLY SUPPORTED)	*fv*

Arithmetic Operators

`add`	addition	*sv*	*prom*
		sv	
`+`	addition	*sv*	*prom*
		sv	
`sub`	subtraction	*sv*	*prom*
		sv	
`-`	subtraction	*sv*	*prom*
		sv	
`mult`	multiplication	*sv*	*prom*
		sv	
`*`	multiplication	*sv*	*prom*
		sv	
`div`	division	*sv*	*prom*
		sv	
`/`	division	*sv*	*prom*
		sv	
`real`	real part	*csv*	*fsv*
`imag`	imaginary part	*csv*	*fsv*
`mag`	magnitude	*csv*	*fsv*
`abs`	absolute value or magnitude	*sv*	*fsv*
`contphase`	continuous phase	*csv*	*fsv*
`prinphase`	principal value phase	*csv*	*fsv*
`conj`	complex conjugate	*csv*	*csv*
`neg`	additive inverse (negative)	*sv*	*sv*
`recip`	reciprocal	*sv*	*sv*

Mathematical Operators

`db`	dB ($20 \log_{10}$)	*sv*	*fsv*
`db10`	dB applied to power ($10 \log_{10}$)	*sv*	*fsv*
`rad2deg`	convert radians to degrees	*fsv*	*fsv*
`deg2rad`	convert degrees to radians	*fsv*	*fsv*
`minlmt`	limit the minimum value	*arg:fsv* *min:f*	*fsv*
`maxlmt`	limit the maximum value	*arg:fsv* *max:f*	*fsv*
`diff`	differences	*fsv*	*fsv*
`deriv`	derivative	*fsv*	*fsv*
`sum`	sums	*fsv*	*fsv*
`integ`	integral	*fsv*	*fsv*

Signal Processing Operators

`smpltime`	current analysis timebase as x *and* y of result	*none*	*fv*
`sweepfrq`	current analysis sweep frequencies as x *and* y of result	*none*	*fv*
`smplcvt`	interpolate *signal1* over timebase of *signal2*	*signal1:v* *signal2:vx*	*vx*
`sweepcvt`	interpolate *frq1* over sweep frequencies of *frq2*	*frq1:v* *frq2:vx*	*vx*
`maketime`	create timebase starting at $t = 0$ in x *and* y of result	*tmax:r* *pts:i*	*vx*
`makesweep`	create sweep frequencies starting at $f = 0$ in x *and* y of result	*fmax:r* *pts:i*	*vx*
`fft`	FFT (argument should have 2^k points)	*timedata:fv*	*cv*
`invfft`	inverse FFT (argument should have $2^k - 1$ points)	*frqdata:cv*	*fv*
`cconv`	real circular (FFT) convolution with zero padding	*signal1:fv* *signal2:fv*	*fv*
`upcconv`	unpadded real circular (FFT) convolution	*signal1:fv* *signal2:fv*	*fv*
`sconv`	slow (time-domain) real convolution	*signal1:fv* *signal2:fv*	*fv*
`fconv`	fast (approximate) real convolution	*signal1:fv* *signal2:fv*	*fv*
`lpbwfrq`	lowpass Butterworth filter frequency response	*frqvec:vx* *corner:f* *order:i*	*cvx*

Other Operators

`catalog`	produce catalog of elements	*none*	*func*

Example

```
.out write catalog in "list.txt"
```

Writes the catalog of the elements in the current fREEDA build and puts the catalog in the file 'list.txt'.

4.15 .tran

Similar to SPICE's *.tran* card with syntax:

```
.tran start stop delta
```

where `start` is the starting transient analysis time, `stop` is the ending time and `delta` is the time increment. If `delta` is zero, the finest time increment is used (determined by the highest frequency, `sfrq` and the number of frequency points `spts`).

4.16 .tran2

Under construction

4.17 .tran4

Under construction

4.18 .tran basel

NO LONGER SUPPORTED

But it will be. The analysis was used in an earlier version and performs convolution-based analysis.

This section defines the options used in Mark Basel's particular form of transient analysis. This analysis is not publicly available. The variables set in in a *.options* statement for this analysis are shown in Table 4.2

Table 4.2: fREEDA runtime options

Variable Name	Definition	Use
iterationdump	Debugging dump for each iteration of transient analysis	ON or OFF
dump	Debugging dump of various variables	ON or OFF
dumpnet	Debugging dump of network as interpreted by TRANSIM	ON or OFF
dcNormal	Switch for using threshold error correction	ON or OFF
spts	number of frequency points used in y(f)	int: power of 2
Z_m type	Matching network impedance form of analysis	float, ohms "transient" "hb"
sfrq	Maximum frequency	float: hz
LPFOrder	low pass filter order	int: 1,2 or 3
impulselength	fraction of impulse response to use in transient analysis	float: 0-1
impulsescale	scale factor for impulse responses	float: any
ytthresthru	relative threshold level for thru and self impulse response terms	float: 0-1
ytthrescross	relative threshold level for cross impulse response terms	float: 0-1
tolerance	stopping difference for successive values in Newton iteration	float: any
maxNoOfIterates	Maximum number of Newton iteration steps per analysis point	int: any
LPFCornerFrequency	corner frequency when using LP filter	float: hz

Chapter 5

Output Control

fREEDA™ has an interpretive output language which uses a reverse polish notation syntax. The operators operate on a stack and as an operation is performed zero or more arguements are consumed by an opertor. This is an extremely powerful way of controlling output.

5.1 Output Commands

`.out write ( (<qualifier> <value>* )* <operator> )* in <filename>`

or

`.out plot ( (<qualifier> <value>* )* <operator> )* [[<gnuplotPostambleScript>] <gnuplotPreambleScript>] in <filename>`

or

`.out system <string>`

5.1.1 Writing

`.out write ( (<qualifier> <value>* )* <operator> )* in <filename>`

The `write` command writes what is left on the stack into the file *filename*.

Example

`.out write term 4 vt in "4v.out"`

This writes the time domain voltage at terminal 4 using the file 4v.out as an output file.

5.1.2 Plotting

`.out plot ( (<qualifier> <value>* )* <operator> )* [[<gnuplotPostambleScript>] <gnuplotPreambleScript>] in <filename>`

The `plot` command writes what is left on the stack into the file *filename* and initiates a plot. The file can be plotted interactively using the fREEDA^{TM} Output Viewer. Also, a file named <*filename*>`.cmd` is created. This file is a gnuplot [24] script file that plots the desired data. The Scripts are optional strings and are used to send additional commands to the gnuplot program.

<*gnuplotPreambleScript*> is a string of semicolon delineated gnuplot commands prior to the plot command which is automatically issued.

<*gnuplotPostambleScript*> is a string of semicolon delineated gnuplot commands after the plot command.

If the option `gnuplot` is present in the `.options` card, the gnuplot program will be called automatically by fREEDA^{TM}. Note that this is generally not needed when using the Output Viewer.

Example

```
.out plot term 4 vt in "4v.out"
```

There are no script commands here. This plots the time domain voltage at terminal 4 using the file 4v.out as an output file. This functions as both a write and a plot.

5.1.3 Running a System Command

`.out system` <*string*>

Use this to run the string as a command to the operating system.

Example

```
.out system "echo Hello"
```

Prints “Hello” on the screen.

5.2 Nomenclature

The following nomenclature is used in describing the output operators.

type	**description**
scalar numeric types	
i	integer
f	floating-point
r	real (integer or floating-point)
c	complex
s	scalar (integer, floating-point or complex)
scalar and mixed numeric types	
fv	floating-point vector
cv	complex vector
v	floating-point or complex vector
fsv	floating-point scalar or vector
csv	complex scalar or vector
sv	scalar or vector (any)
prom	an appropriately-promoted numeric type
-x	(suffix to vector types) x data required
other types	
any	any type
string	character string
var	variable name
file	data file
func	function pointer

5.3 Qualifiers

type	**description**
qualifiers (network types)	
term	terminal (or node)
element	circuit element

5.4 Operators

5.4.1 General Operators

operator	function	argument(s)	result
`dup`	duplicate object	*any*	*same*
`get`	get element of vector	*arg:v* *index:i*	*s*
`put`	modify element of vector	*arg:v* *index:i* *val:s*	*v*
`stripx`	remove x data	*vx*	*v*
`pack`	concatenates last *vx*'s on stack	variable number of *vx*	*m*
`system`	execute shell command	*string*	*none*

5.4.2 Network Operators

`vf`	complex freq. domain voltage vector at a terminal	*term*	*cv*
`if`	complex freq. domain current vector at a terminal	*term*	*cv*
`xf`	complex freq. domain state variable vector at a terminal	*term*	*cv*
`vt`	time domain voltage vector at a terminal	*term*	*fv*
`it`	time domain current vector at a terminal	*term*	*fv*
`ut`	time domain voltage vector at an element port	*elem*	*fv*

5.4.3 RPN Arithmetic Operators

Arithmetic Operators for reverse polish notation e.g. 3 4 add = 7

`add`	addition	*sv*	*prom*
		sv	
`sub`	subtraction	*sv*	*prom*
		sv	
`mult`	multiplication	*sv*	*prom*
		sv	
`div`	division	*sv*	*prom*
		sv	
`real`	real part	*csv*	*fsv*
`imag`	imaginary part	*csv*	*fsv*
`mag`	magnitude	*csv*	*fsv*
`abs`	absolute value or magnitude	*sv*	*fsv*
`contphase`	continuous phase	*csv*	*fsv*
`prinphase`	principal value phase	*csv*	*fsv*
`conj`	complex conjugate	*csv*	*csv*
`neg`	additive inverse (negative)	*sv*	*sv*
`recip`	reciprocal	*sv*	*sv*

5.4.4 Mathematical Operators

`db`	dB ($20 \log_{10}$)	*sv*	*fsv*
`db10`	dB applied to power ($10 \log_{10}$)	*sv*	*fsv*
`rad2deg`	convert radians to degrees	*fsv*	*fsv*
`deg2rad`	convert degrees to radians	*fsv*	*fsv*
`minlmt`	limit the minimum value	*arg:fsv*	*fsv*
		min:f	
`maxlmt`	limit the maximum value	*arg:fsv*	*fsv*
		max:f	
`diff`	differences	*fsv*	*fsv*
`deriv`	derivative	*fsv*	*fsv*
`sum`	sums	*fsv*	*fsv*
`integ`	integral	*fsv*	*fsv*

5.4.5 Signal Processing Operators

`smpltime`	current analysis timebase as x *and* y of result	*none*	*fv*
`sweepfrq`	current analysis sweep frequencies as x *and* y of result	*none*	*fv*
`smplcvt`	interpolate *signal1* over timebase of *signal2*	*signal1:v* *signal2:vx*	*vx*
`sweepcvt`	interpolate *frq1* over sweep frequencies of *frq2*	*frq1:v* *frq2:vx*	*vx*
`maketime`	create timebase starting at $t = 0$ in x *and* y of result	*tmax:r* *pts:i*	*vx*
`makesweep`	create sweep frequencies starting at $f = 0$ in x *and* y of result	*fmax:r* *pts:i*	*vx*
`fft`	FFT (argument should have 2^k points)	*timedata:fv*	*cv*
`invfft`	inverse FFT (argument should have $2^k - 1$ points)	*frqdata:cv*	*fv*
`cconv`	real circular (FFT) convolution with zero padding	*signal1:fv* *signal2:fv*	*fv*
`upcconv`	unpadded real circular (FFT) convolution	*signal1:fv* *signal2:fv*	*fv*
`sconv`	slow (time-domain) real convolution	*signal1:fv* *signal2:fv*	*fv*
`fconv`	fast (approximate) real convolution	*signal1:fv* *signal2:fv*	*fv*
`lpbwfrq`	lowpass Butterworth filter frequency response	*frqvec:vx* *corner:f* *order:i*	*cvx*

Chapter 6

Graphical User Interface

6.1 Introduction

fREEDA supports three interactive front ends:

- iFREEDA — the preferred interface and part of the fREEDA distribution.
- Electric Editor — Very good for VLSI layout, not documented.
- fREEDA GUI — not currently distributed but described in this chapter.

The simulation engine in fREEDA$^{\text{TM}}$ can be used in the traditional way as a stand-alone program, for example in batch jobs. In this mode, the program reads an input netlist, process its contents and writes the requested output files.

fREEDA also provides a Graphical User Interface (GUI), which is more convenient for interactive use of the program. This GUI is written using the Java language, so it can be used in every system where Java is supported. In this chapter we describe the different components of the GUI. This has now been replaced by iFREEDA but this documentation is provided for completeness and the code is available.

6.2 The Netlist Editor

The netlist editor is a simple text editor combined with a simulation manager. The editor window is shown in Figure 6.1. Besides the normal editing commands, the editor provides buttons and keyboard shortcuts to analyze the netlist being edited and see the output of the simulation.

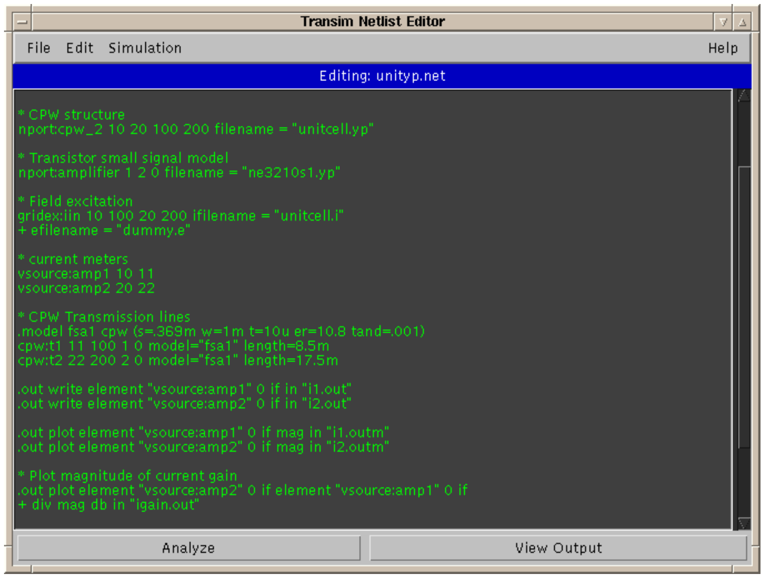

Figure 6.1: Netlist Editor window.

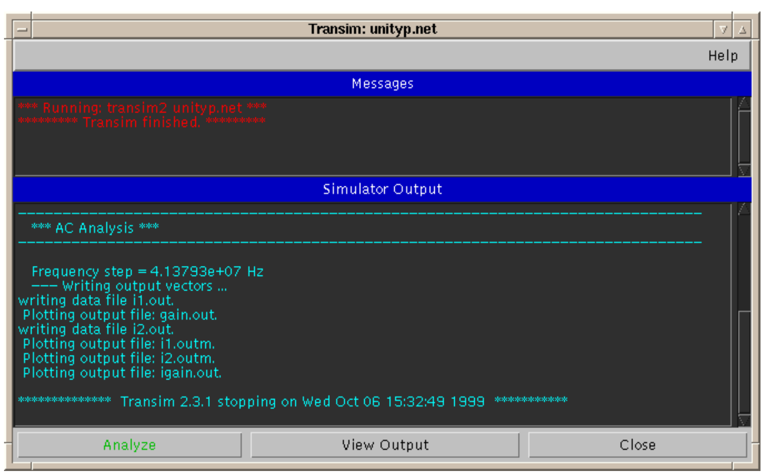

Figure 6.2: Analysis window.

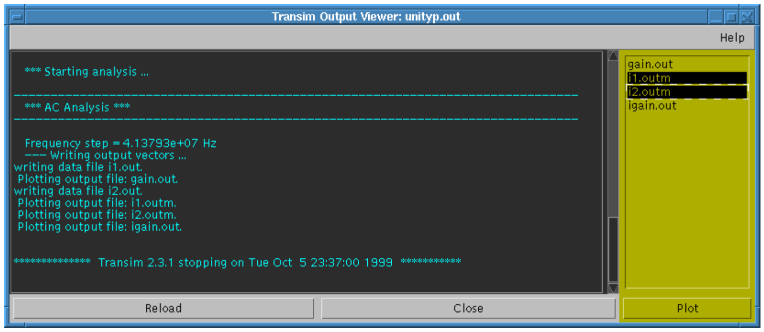

Figure 6.3: Output Viewer window.

The editor can edit several files and handle multiple simulations at once by spawning multiple windows.

6.3 The Analysis Window

The analysis window is used to show the progress of a simulation (Figure 6.2). The upper subwindow displays important messages such as when the program starts or stops, and also warnings and errors that may occur during the simulation. The lower subwindow shows the progress of the simulation.

The buttons are self-explaining. The "Analyze" button changes to "Stop" when the engine is running.

6.4 The Output Viewer Window

This window is perhaps the most useful of all. It is shown in Figure 6.3. The output file is displayed at the left. This file contains detailed information about the simulation.

At the right there is a list of files available for plotting. After selecting one or more of these files and depressing the "Plot" button, a plot window appears showing the desired data (see Figure 6.4). Any number of plots can be requested. Also, the plot data is kept in memory by the plot window, so it is possible to re-run a simulation with different parameters and compare the new and old graphs on the screen. An encapsulated postscript file can be generated pressing the corresponding button.

There are several features provided in the plot window. One of the most remarkable is that it is possible to zoom in or out the graph by dragging the left or right mouse button, respectively.

The plotting facility is provided by the ptplot [45] library.

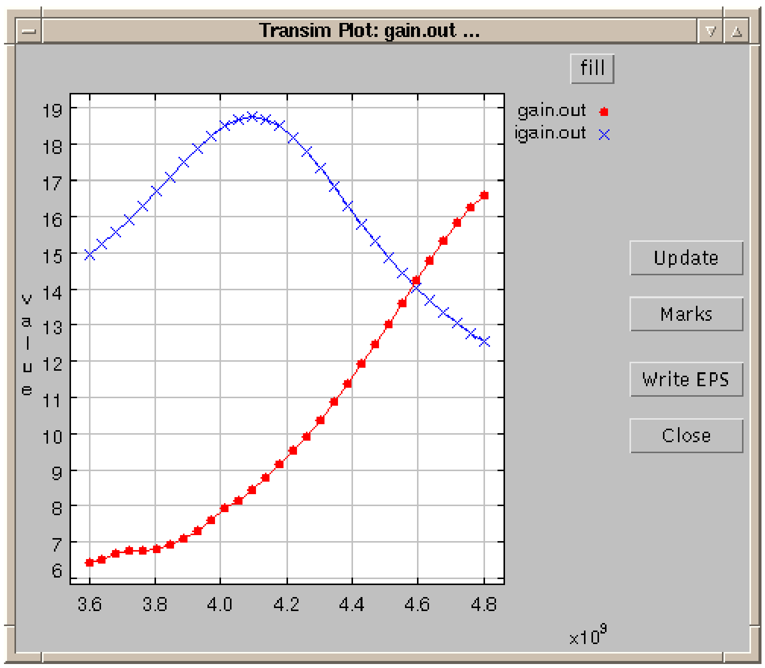

Figure 6.4: Plot window.

Bibliography

[1] S. M. S. Imtiaz and S. M. El-Ghazaly, “Global modeling of millimeter-wave circuits: electromagnetic simulation of amplifiers,” IEEE Trans. on Microwave Theory and Tech., vol 45, pp. 2208-2217. Dec. 1997.

[2] C.-N. Kuo, R.-B. Wu, B. Houshmand, and T. Itoh, Modeling of microwave active devices using the FDTD analysis based on the voltage-source approach, IEEE Microwave Guided Wave Lett., vol. 6, pp. 199-201, May 1996.

[3] E. Larique, S. Mons, D. Baillargeat, S. Verdeyme, M. Aubourg, P. Guillon, and R. Quere, “Electromagnetic analysis for microwave FET modeling,” IEEE microwave and guided wave letters Vol 8, pp. 41-43, Jan. 1998.

[4] T. W. Nuteson, H. Hwang, M. B. Steer, K. Naishadham, J.W.Mink, and J. Harvey, “Analysis of finite grid structures with lenses in quasi-optical systems,” IEEE Trans. Microwave Theory Techniques, pp. 666-672, May 1997.

[5] M. B. Steer, M. N. Abdullah, C. Christoffersen, M. Summers, S. Nakazawa, A. Khalil, and J. Harvey, “Integrated electro-magnetic and circuit modeling of large microwave and millimeter-wave structures,” Proc. 1998 IEEE Antennas and Propagation Symp., pp. 478-481, June 1998.

[6] J. Kunisch and I. Wolff, “Steady-state analysis of nonlinear forced and autonomous microwave circuits using the compression approach,” Int. J. of Microwave and Millimeter-Wave Computer-Aided Engineering, vol. 5, No. 4, pp. 241-225, 1995

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest *Introduction to Algorithms*, The MIT Press, McGraw-Hill Book Company, 1990.

[8] A. Eliëns, Principles of object-oriented software development, Adison-Wesley, 1995.

[9] R. C. Martin. “The dependency inversion principle,” C++ Report, May 1996.

[10] R. C. Martin, “The Open Closed Principle,” C++ Report, Jan. 1996.

[11] R. C. Martin, “The Liskov Substitution Principle,” C++ Report, March 1996.

[12] R. C. Martin, “The Interface Segregation Principle,” C++ Report, Aug 1996.

[13] R. C. Martin, “UML Tutorial: Part 1 — Class Diagrams,” Engineering Notebook Column, C++ Report, Aug. 1997.

[14] A. D. Robison, "C++ Gets Faster for Scientific Computing," Computers in Physics, vol. 10, pp. 458-462, 1996.

[15] J. R. Cary and S. G. Shasharina, "Comparison of C++ and Fortran 90 for Object-Oriented Scientific Programming," Available from Los Alamos National Laboratory as Report No. LA-UR-96-4064.

[16] The Object Oriented Numerics Page, http://oonumerics.org/.

[17] Silicon Graphics, Standard Template Library Programmer's Guide, http://www.sgi.com/Technology/STL/.

[18] T. Veldhuizen, Techniques for Scientific C++ - Version 0.3, Indiana University, Computer Science Department, 1999. (http://extreme.indiana.edu/ tveldhui/papers/techniques/)

[19] A. Griewank, D. Juedes, J. Utke, "Adol-C: A Package for the Automatic Differenciation of Algorithms Written in C/C++," ACM TOMS, vol. 22(2), pp. 131-167, June 1996.

[20] R. Pozo, MV++ v. 1.5a, Reference Guide, National Institute of Standards and Technology, 1997.

[21] M. Frigo and S. G. Johnson, FFTW User's Manual, Massachusetts Institute of Technology, September 1998.

[22] K. S. Kundert and A. Songiovanni-Vincentelli, Sparse user's guide - a sparse linear equation solver, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, Calif. 94720, Version 1.3a, Apr 1988.

[23] R. S. Bain, NNES user's manual, 1993.

[24] Gnuplot. Copyright(C) 1986 - 1993, 1998 Thomas Williams, Colin Kelley and many others.

[25] M. Valtonen and T. Veijola, "A microcomputer tool especially suited for microwave circuit design in frequency and time domain," Proc. URSI/IEEE National Convention on Radio Science, Espoo, Finland, 1986, p. 20,

[26] M. Valtonen, P. Heikkilä, A. Kankkunen, K. Mannersalo, R. Niutanen, P. Stenius, T. Veijola and J. Virtanen, "APLAC - A new approach to circuit simulation by object orientation," 10th European Conference on Circuit Theory and Design Dig., 1991.

[27] K. Mayaram and D. O. Pederson, "CODECS: an object-oriented mixed-level circuit and device simulator," 1987 IEEE Int. Symp. on Circuits and Systems Digest, 1987, pp 604-607.

[28] A. Davis, "An object-oriented approach to circuit simulation," 1996 IEEE Midwest Symp. on Circuits and Systems Dig., 1996, pp 313-316.

[29] B. Melville, P. Feldmann and S. Moinian, “A C++ environment for analog circuit simulation,” 1992 IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors.

[30] P. Carvalho, E. Ngoya, J. Rousset and J. Obregon, “Object-oriented design of microwave circuit simulators,” 1993 IEEE MTT-S Int. Microwave Symp. Digest, June 1993, pp 1491-1494.

[31] C. E. Christoffersen and M. B. Steer “Implementation of the local reference concept for spatially distributed circuits,” Int. J. of RF and Microwave Computer-Aided Eng., vol. 9, No. 5, 1999.

[32] A. I. Khalil and M. B. Steer “Circuit theory for spatially distributed microwave circuits,” IEEE Trans. on Microwave Theory and Techn., vol. 46, Oct. 1998, pp 1500-1503.

[33] C. E. Christoffersen, M. Ozkar, M. B. Steer, M. G. Case and M. Rodwell, “State variable-based transient analysis using convolution,” IEEE Transactions on Microwave Theory and Techniques, Vol. 47, June 1999, pp. 882-889.

[34] C. E. Christoffersen, M. B. Steer and M. A. Summers, “Harmonic balance analysis for systems with circuit-field interactions,” 1998 IEEE Int. Microwave Symp. Dig., June 1998, pp. 1131-1134.

[35] B. Speelpenning. “Compiling Fast Partial Derivatives of Functions Given by Algorithms,” Ph.D. thesis (Under the supervision of W. Gear), Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana-Champaign, Ill., January 1980.

[36] T. F. Coleman y G. F. Jonsson, “The Efficient Computation of Structured Gradients using Automatic Differentiation,” Cornell Theory Center Technical Report CTC97TR272, April 28, 1997

[37] H. S. Tsai, M. J. W. Rodwell and R. A. York, “Planar amplifier array with improved bandwidth using folded-slots,” IEEE Microwave and Guided Wave Letters, vol. 4, April 1994, pp. 112-114.

[38] M. B. Steer, M. N. Abdullah, C. Christoffersen, M. Summers, S. Nakazawa, A. Khalil, and J. Harvey, “Integrated electro-magnetic and circuit modeling of large microwave and millimeter-wave structures,” Proc. 1998 IEEE Antennas and Propagation Symp., pp. 478–481, June 1998.

[39] M. N. Abdulla, U.A. Mughal, and M B. Steer, “Network Charactarization for a Finite Array of Folded-Slot Antennas for Spatial Power Combining Application,” Proc. 1999 IEEE Antennas and Propagation Symp., July 1999.

[40] U. A. Mughal, “Hierarchical approach to global modeling of active antenna arrays,” M.S. Thesis, North Carolina State University, 1999.

[41] Rational Software, UML Resources, http://www.rational.com/.

[42] M. B. Steer, J. F. Harvey, J. W. Mink, M. N. Abdulla, C. E. Christoffersen, H. M. Gutierrez, P. L. Heron, C. W. Hicks, A. I. Khalil, U. A. Mughal, S. Nakazawa, T. W. Nuteson, J. Patwardhan, S. G. Skaggs, M. A. Summers, S. Wang, and A. B. Yakovlev, "Global modeling of spatially distributed microwave and millimeter-wave systems," IEEE Trans. Microwave Theory Techniques, June 1999, pp. 830-839.

[43] C. E. Christoffersen, S. Nakazawa, M. A. Summers, and M. B. Steer, "Transient analysis of a spatial power combining amplifier", 1999 IEEE MTT-S Int. Microwave Symp. Dig., June 1999, pp. 791-794.

[44] M. A. Summers, C. E. Christoffersen, A. I. Khalil, S. Nakazawa, T. W. Nuteson, M. B. Steer and J. W. Mink, "An integrated electromagnetic and nonlinear circuit simulation environment for spatial power combining systems," 1998 IEEE MTT-S Int. Microwave Symp. Dig., June 1998, pp. 1473-1476.

[45] Ptplot. http://ptolemy.eecs.berkeley.edu/java/ptplot

[46] V. Rizzoli, F. Mastri, F. Sgallari, G. Spaletta, *Harmonic-Balance Simulation of Strongly Nonlinear very Large-Size Microwave Circuits by Inexact Newton Methods*, IEEE MTT-S Digest, 1996.

[47] V. Rizzoli, A. Costanzo, and A. Lipparini, *An Electrothermal Functional Model of the Microwave FET Suitable for Nonlinear Simulation* International Journal of Microwave and Millimeter-Wave Computer-Aided Engineering, Vol. 5, No. 2, 104-121 (1995).

[48] V. Rizzoli, A. Lipparini, A. Costanzo, F. Mastri, C. Ceccetti, A. Neri and D. Masotti, *State-of-the-Art Harmonic-Balance Simulation of Forced Nonlinear Microwave Circuits by the Piecewise Technique*, IEEE Trans. on Microwave Theory and Techniques, Vol. 40, No. 1, Jan 1992.

[49] M. M. Gourary, S. G. Rusakov, S. L. Ulyanov, M. M. Zharov, K. K. Gullapalli, and B. J. Mulvaney, *Iterative Solution of Linear Systems in Harmonic Balance Analysis*, IEEE MTT-S Digest, 1997.

[50] I. Moret, *On the Convergence of Inexact Quasi-Newton Methods*, International J. of Computer Math., Vol. 28, pp. 117-137, 1987.

[51] M. S. Nakhla, J. Vlach, *A Piecewise Harmonic Balance Technique for Determination of Periodic Response of Nonlinear Systems*, IEEE Trans. on Circuits and Systems, Vol CAS-23, No. 2, Feb 1976.

[52] A. Materka and T. Kacprzak, *Computer Calculation of Large-Signal GaAs FET Amplifier Characteristics*, IEEE Trans. on Microwave Theory and Techniques, Vol MTT-33, No. 2, Feb 1985.

[53] M. B. Steer, *Transient and Steady-State Analysis of Nonlinear RF and Microwave Circuits*, ECE603 class notes, August 15, 1996.

[54] J. F. Sevic, M. B. Steer, and A. M. Pavio, *Nonlinear Analysis Methods for the Simulation of Digital Wireless Communication Systems*, International Journal of Microwave and Millimiter-Wave Computer-Aided Engineering, Vol. 6, No. 3, 197-216, 1996.

[55] J. Kunisch and I. Wolff, *Steady-State Analysis of Nonlinear Forced and Autonomous Microwave Circuits Using the Compression Approach*, International Journal of Microwave and Millimeter-Wave Computer-Aided Engineering, Vol. 5, No. 4, 241-255 (1995).

[56] E. Ngoya, A. S. R. Sommet and R. Quéré, *Steady State Analysis of Free or Forced Oscillators by Harmonic Balance and Stability Investigation of Periodic and Quasi-Periodic Regimes*, International Journal of Microwave and Millimeter-Wave Computer-Aided Engineering, Vol. 5, No. 3, 210-223 (1995).

[57] Compact Software, *Microwave Harmonica Elements Library*, (1994).

[58] M. J. D. Powell, *A hybrid method for nonlinear equations*, Numerical Methods for Nonlinear Algebraic Equations, P. Rabinowitz, Editor, Gordon and Breach, 1988.

[59] K. S. Kundert, J. K. White and A. Sangiovanni-Vincentelli, *Steady-state methods for simulating analog and microwave circuits*, Boston, Dordrecht, Kluwer Academic Publishers, 1990.

[60] M. B. Steer, C. Chang and G. W. Rhyne, *Computer-Aided Analysis of Nonlinear Microwave Circuits Using Frequency-Domain Nonlinear Analysis Techniques: The State of the Art*, International Journal of Microwave and Millimeter-Wave Computer-Aided Engineering, Vol. 1, No. 2, 181-200, 1991.

[61] R. J. Gilmore and M. B. Steer, *Nonlinear Circuit Analysis Using the Method of Harmonic Balance—A Review of the Art. II. Advanced Concepts*, International Journal of Microwave and Millimeter-Wave Computer-Aided Engineering, Vol. 1, No. 2, 159-180, 1991.

[62] C. R. Chang, *Computer-Aided Analysis of Nonlinear Microwave Analog Circuits Using Frequency-Domain Spectral Balance*, Ph.D. Thesis, Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC, 1990.

[63] D. D'Amore, P. Maffezzoni and M. Pillan, *A Newton-Powell Modification Algorithm for Harmonic Balance-Based Circuit Analysis*, IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications, Vol. 41, No. 2, February 1994.

[64] Y. Thodesen, K. Kundert, *Parametric harmonic balance*, IEEE MTT S. International Microwave Symposium Digest, Vol 3, 1996, IEEE, Piscataway, NJ, USA, pp. 1361-1364.

[65] A. Ushida and L. O. Chua. *Frequency-domain analysis of nonlinear circuits driven by multi-tone signals*, IEEE Transactions on Circuits and Systems, Vol. CAS-31, No. 9, September 1984, pp. 766-778.

[66] A. Ushida, L. O. Chua and T. Sugawara. *A substitution algorithm for solving nonlinear circuits with multi-frequency components*, International Journal on Circuit Theory and Application, Vol. 15, 1987, pp. 327-355.

[67] R. J. Gilmore and F. J. Rosenbaum, *Modelling of nonlinear distortion in GaAs MESFETs*, 1984 IEEE MTT-S International Microwave Symposium Digest, May 1984, pp. 430-431.

[68] G. P. Bava, S. Benedetto, E. Biglieri, F. Filicori, V. A. Monaco, C. Naldi, U. Pisani and V. Pozzolo, *Modelling and perfomance simulation Techniques of GaAs MESFETs for microwave power amplifiers*, ESA-ESTEC Report, Noordwijk, Holland, March 1982.

[69] H. Makino and H. Asai, *Relaxation-based circuit simulation techniques in the frequency domain*, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol E76-A, No. 4 Apr 1993, p 626-630.

[70] A. Brambilla, D. D'Amor, M. Pillan, *Convergence improvements of the harmonic balance method*, Proceedings IEEE International Symposium on Circuits and Systems, Vol. 4 1993, Publ. by IEEE, IEEE Service Center, Piscataway, NJ, USA. p 2482-2485.

[71] V. Rizzoli, A. Costanzo, P. R. Ghigi, F. Mastri, D. Masotti, C. Cecchetti, *Recent advances in harmonic-balance techniques for nonlinear microwave circuit simulation*, AEU Arch Elektron Uebertrag Electron Commun, Vol. 46, No. 4 Jul 1992, p 286-297.

[72] M. Celik, A. Atalar, M. A. Tan, *New method for the steady-state analysis of periodically excited nonlinear circuits*, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, Vol. 43, No. 12 Dec 1996, p 964-972.

[73] H. G. Brachtendorf, G. Welsch, R. Laur, *Fast simulation of the steady-state of circuits by the harmonic balance technique*, Proceedings IEEE International Symposium on Circuits and Systems, Vol. 2 1995, IEEE, Piscataway, NJ, USA, p 1388-1391.

[74] H. G. Brachtendorf, G. Welsch, R. Laur, *Simulation tool for the analysis and verification of the steady state of circuit designs*, International Journal of Circuit Theory and Applications, Vol. 23, No 4 Jul-Aug 1995, p 311-323.

[75] I. Barbancho Perez, I. Molina Fernandez, *Predictor strategies for continuation methods applied to nonlinear circuit analysis*, Industrial Applications in Power Systems, Computer Science and Telecommunications Proceedings of the Mediterranean Electrotechnical Conference MELECON, Vol. 3 1996, IEEE, Piscataway, NJ, USA, p 1419-1422.

[76] M. S. Basel, M. B. Steer and P. D. Franzon, "Simulation of high speed interconnects using a convolution-based hierarchical packaging simulator," *IEEE Trans. on Components, Packaging, and Manufacturing Techn.*, Vol. 18, February 1995, pp. 74-82.

[77] T. J. Brazil, "A new method for the transient simulation of causal linear systems described in the frequency domain," *1992 IEEE MTT-S Int. Microwave Symp. Digest*, June 1992, pp. 1485-1488.

[78] P. Perry and T. J. Brazil, “Hilbert-transform-derived relative group delay,” *IEEE Trans. on Microwave Theory and Techn.*, Vol 45, Aug. 1997, pt. 1, pp. 1214-1225.

[79] T. J. Brazil, “Causal convolution—a new method for the transient analysis of linear systems at microwave frequencies,” *IEEE Trans. on Microwave Theory and Techn.*, Vol. 43, Feb. 1995, pp. 315-23.

[80] A. R. Djordjevic and T. K. Sarkar, “Analysis of time response of lossy multiconductor transmission line networks,” *IEEE Trans. on Microwave Theory and Techn.*, Vol. MTT-35, Oct. 1987, pp. 898-908.

[81] D. Winkelstein, R. Pomerleau and M. B. Steer, “Transient simulation of complex, lossy, multi-port transmission line networks with nonlinear digital device termination using a circuit simulator,” *Conf. Proc. IEEE SOUTHEASTCON*, Vol. 3, pp. 1239-1244.

[82] J. E. Schutt-Aine and R. Mittra, “Nonlinear transient analysis of coupled transmission lines,” *IEEE Trans. on Circuits and Systems*, Vol. 36, Jul. 1989, pp. 959-967.

[83] P. K. Chan, Comments on “Asymptotic waveform evaluation for timing analysis,” *IEEE Trans. on Computer Aided Design*, Vol. 10, Aug. 1991, pp. 1078-79.

[84] M. Celik, O. Ocali, M. A. Tan, and A. Atalar, “Pole-zero computation in microwave circuits using multipoint Padé approximation,” *IEEE Trans. on Circuits and Systems*, Jan. 1995, pp. 6-13.

[85] E. Chiprout and M. Nakhla, “Fast nonlinear waveform estimation for large distributed networks,” *1992 IEEE MTT-S Int. Microwave Symp. Digest*, Vol.3, Jun. 1992, pp. 1341-1344.

[86] R. J. Trihy and Ronald A. Rohrer, “AWE macromodels for nonlinear circuits,” *Proceedings of the 36th Midwest Symposium on Circuits and Systems*, Vol. 1, Aug. 1993, pp. 633-636.

[87] R. Griffith and M. S. Nakhla, “Mixed frequency/time domain analysis of nonlinear circuits,” *IEEE Trans. on Computer Aided Design*, Vol.11, Aug. 1992, pp. 1032-43.

[88] M. Ozkar, *Transient analysis of spatially distributed microwave circuits using convolution and state variables*, M. S. Thesis, Department of Electrical and Computer Engineering, North Carolina State University.

[89] C. Gordon, T. Blazeck and R. Mittra, “Time domain simulation of multiconductor transmission lines with frequency-dependent losses,” *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, Vol. 11 Nov. 1992 pp. 1372-87.

[90] P. Stenius, P. Heikkilä and M. Valtonen, “Transient analysis of circuits including frequency-dependent components using transgyrator and convolution,” *Proc. of the 11th European Conference on Circuit Theory and Design*, Part II, 1993, pp. 1299-1304.