Table of contents

[Hex Open netlist file specification, features and implementation questions 3](#_Toc94279353)

[Vision 3](#_Toc94279354)

[Implemented and not implemented SPICE elements 4](#_Toc94279355)

[SPICE functions planned to be implemented in the first round 4](#_Toc94279356)

[SPICE functions not planned to be implemented in the first round 5](#_Toc94279357)

[Special features of Hex Open 6](#_Toc94279358)

[Parts of the simulator software package 8](#_Toc94279359)

[Command line usage with classic VSUN files 8](#_Toc94279360)

[Integrated future GUI 8](#_Toc94279361)

[Structure of the Hex Open Core 10](#_Toc94279362)

[NR modell építése 13](#_Toc94279363)

[Vezérlő bemenet ill. vezérlő csomópont automatikus felismerése 14](#_Toc94279364)

[Vezérlő csomópontok redukciója 14](#_Toc94279365)

[New and extended structure elements and implementation questions 15](#_Toc94279366)

[Kifejezések 15](#_Toc94279367)

[Indirekt azonosítók 15](#_Toc94279368)

[Component templates 15](#_Toc94279369)

[Rövidzár 20](#_Toc94279370)

[T0 and thermal nodes (Txxxx) 21](#_Toc94279371)

[Floating nodes, non-connecting sub-circuit external nodes and empty sub-circuits 21](#_Toc94279372)

[Irányított ellenállás 21](#_Toc94279373)

[Dissipator 21](#_Toc94279374)

[Hysteresis 22](#_Toc94279375)

[Values of nodes, components and variables 22](#_Toc94279376)

[SUNRED 24](#_Toc94279377)

[Enabling, disabling, destroying components 25](#_Toc94279378)

[SUBCKT replacement 25](#_Toc94279379)

[Parametric SUBCKT 26](#_Toc94279380)

[Control input ~~and output~~ for SUBCKT 26](#_Toc94279381)

[Two-terminal subcircuits 27](#_Toc94279382)

[Variables 27](#_Toc94279383)

[Constants 28](#_Toc94279384)

[Predefined variables and constants 29](#_Toc94279385)

[The .if statement in the netlist description 29](#_Toc94279386)

[Controllers 30](#_Toc94279387)

[Subcontrollers => needed? Not supported yet, should be revised. 32](#_Toc94279388)

[(Sub)controller updates 33](#_Toc94279389)

[Equation given piecewise 33](#_Toc94279390)

[Updating normal nodes and controller nodes, error calculation 33](#_Toc94279391)

[Initialize a node from another node 34](#_Toc94279392)

[Compact models 34](#_Toc94279393)

[Quasi-Constant Nonlinear components 34](#_Toc94279394)

[Math functions 35](#_Toc94279395)

[Samples 35](#_Toc94279396)

[Expressions 35](#_Toc94279397)

[Lines beginning with X 36](#_Toc94279398)

[.forward 36](#_Toc94279399)

[New and extended control operations and implementation questions 37](#_Toc94279400)

[The .if, .while and .for statements in control operations 37](#_Toc94279401)

[Implemented analysis types 37](#_Toc94279402)

[.solutionmethod 37](#_Toc94279403)

[Result saving 38](#_Toc94279404)

[Az iteráció menete 38](#_Toc94279405)

[Subckt / Controller példányosítása 38](#_Toc94279406)

[Bug list 39](#_Toc94279407)

# Hex Open netlist file specification, features and implementation questions

## Vision

Hex Open is a very flexible multi-domain finite simulator. The cause of the flexibility is that the finite model is described by an electrical network. However, a finite model can contain hundreds of thousands or millions of nodes, and other simulation needs are articulated than what a general circuit simulator can provide. Therefore, the concept of the simulator is not to generate an electrical netlist and solve it with a circuit simulator, but to create a finite simulator that can process SPICE netlists. New elements are added to the SPICE model, with the help of which the finite simulation can be properly implemented.

## Implemented and not implemented SPICE elements

The ultimate goal is to implement all the capabilities of SPICE. In the first round, however, the goal is to develop a version of the simulator that can perform the tasks planned in the near future (TELC compact modelling, LED simulations, compact modelling of microchannel systems).

### SPICE functions planned to be implemented in the first round

#### Implemented types of analysis

* DC analysis
* Transient analysis

#### Implemented general structure and conventions

* First line is the title
* Last line: .END
* Number format:
  + Integer: 20
  + Floating point:
    - 2.71
    - 1e-14, -3.2e5
    - T (1e12), G (1e9), Meg (1e6), K (1e3), mil (25.4e-6), m (1e-3), u (1e-6), n (1e-9), p (1e-12), f (1e-15): 1K, 2.5Meg
  + Letters immediately following a number that are not scale factors are ignored, and letters immediately following a scale factor are ignored: 1KHz, 10mA
* Node names are handled as strings. E.g. 0 and 00 are different.
* If a line starts with \* or white space, it is a comment
* .INCLUDE *filename*: if relative path is used, it must be started with a ‘.’. If the path is not started with a ., it is used as an absolute path. If a filename is given without a path, it must be in the same folder as the spice/ho file.

#### Implemented device models

* ~~R Semiconductor resistor model~~
* ~~C Semiconductor capacitor model~~
* ~~D Diode model~~

#### Implemented circuit elements

* .SUBCKT => .ENDS,
  + X
* R: resistor~~, semiconductor resistor~~
* C: capacitor~~, semiconductor capacitor~~
* V: independent voltage source
* I: independent current source
* ~~G: linear voltage-controlled current source~~
* ~~E: linear voltage-controlled voltage source~~
* ~~F: linear current-controlled current source~~
* ~~H: linear current-controlled voltage source~~
* ~~B: non-linear dependent source~~
* ~~D: junction diode~~

### SPICE functions not planned to be implemented in the first round

#### Not implemented types of analysis

* AC Small-Signal Analysis
* Pole-Zero Analysis
* Small-Signal Distortion Analysis
* Sensitivity Analysis
* Noise Analysis

#### Not implemented general structure and conventions

#### Not implemented device models

* SW Voltage controlled switch
* CSW Current controlled switch
* URC Uniform distributed RC model
* LTRA Lossy transmission line model
* NPN NPN BJT model
* PNP PNP BJT model
* NJF N-channel JFET model
* PJF P-channel JFET model
* NMOS N-channel MOSFET model
* PMOS P-channel MOSFET model
* NMF N-channel MESFET model
* PMF P-channel MESFET model

#### Not implemented circuit elements

* L: inductor
* K: coupled (mutual) inductor
* S: voltage controlled switch
* W: current controlled switch
* T: lossless transmission line
* O: lossy transmission line
* U: uniform distributed RC line
* Q: bipolar junction transistor
* J: junction field-effect transistor
* M: MOSFET
* Z: MESFET

## Special features of Hex Open

Hex Open has special features that need to be considered during design and implementation:

* Alternating calculation of electric and thermal fields. (The same applies to the rotated calculation of other fields.): By default, the Newton-Raphson model of the electro-thermal field is an interconnected network model. However, there are cases where it is more efficient to calculate the electric field and the thermal field separately and use the calculation result of one to calculate the other. In this case, an iteration can be computed faster and requires less memory, in return, the number of iterations may increase, convergence may deteriorate, divergence may appear. Which method is more effective depends on the nature of the simulated problem.   
    
  Solution: application of sub-circuits, enabling and disabling of the corresponding parts. Part calculation results for disabled sub-circuits should also be preserved because deleting and re-creating them is time consuming. However, it can be allowed to be deleted with a separate instruction, if required, to save memory. However, the values of the nodes must be preserved, but they must not be updated when inactive, so it is advisable to store the nodes per sub-circuits.
* Boundary condition replacement: If we want to change a boundary condition during the simulation, we have the following options:
  + If the boundary condition type is not, only its value changes: the value can be specified with a variable, the value of the variable can be changed.
  + For a simple boundary condition, if we know in advance that the type will change: specify the parts of the cell with conditions (.IF, .ELIF, .ELSE).
  + For a simple boundary condition, if we don't know in advance that the type will change: replace SUBCKT with the .REPLACE statement.
  + In the case of a complex boundary condition: the boundary condition is defined as a sub-circuit, it is switched by enabling-disabling.
* DC and AC: each node also has a DC and AC version. AC is only generated if AC type analysis is run.
* Diode replaced by resistor for the first few iterations (The diode equation can cause a convergence problem, so for the first few iterations of the DC / transient simulation, it may be worth putting a resistor in place of the diode.): Based on the value of a variable, we choose between the diode and the resistor with the .IF instruction. The simulation is iterated manually. E.g.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | .set iter 0  .while(iter<4)  .iterate .set iter iter+1  .endwhile | .for 4  .iterate  .endfor |

* Partial replacement of the simulated structure: For example, we connect the device under test to another surface or change the finite mesh. In this case, part of the model circuit changes significantly.   
    
  Solution: application of sub-circuits, enabling and disabling of the corresponding parts. Another extension that aids implementation is the initializability of nodes based on the value of other nodes. One node can be initialized based on the weighted average of the values of several other nodes.
* Insert a compact model: the frame program creates the netlist file in such a way that the netlist of the compact model can be easily entered into it manually.
* To save the state for continuation and modified additional simulation: The goal is to be able to resume if a long simulation is interrupted due to a program error, power failure, or other reason. Alternatively, if we want to continue in several different ways from one step of the simulation, the given state should be saved.
* Controller and subcontroller: it is often useful if we can calculate something from the results of the previous iteration and use that value for a component, e.g. to control a controlled source. If we do this not as a circuit, but as a single calculation, then this will be the controller. It can be useful if a controller can even build on the computational result of other controllers in other cells. However, the cells can be computed by different program threads in parallel, and in this case the output value of the controller in the other cell depends on the execution order of the competing program threads, making it unsuitable for control. The solution is the subcontroller: the output value of the subcontroller is calculated by the program thread belonging to the main controller, before calculating the value of the main controller. The subcontroller may also have subcontrollers.

## Parts of the simulator software package

All versions follow the “clever frame” – “strong core” model.

### Command line usage with classic VSUN files

In the classic command line mode, Hex Open can be used similarly to the Hex LED. VSUN files are converted by Frame, but this time the output is not cells and their merge instructions, but an extended SPICE netlist.

VSUN

Files

Frame

SPICE to

Intermediate

Core

To FIM

Additional data on the structure

Extended

SPICE

Netlist

(Extended)

SPICE

Netlist

Intermediate

Files

Simulation

Result

Files

Simulation

Result

Files

FIM

Files

An additional step, the SPICE to intermediate converter (SIC), has been added between Frame and Core. This is because SPICE treats the name of the nodes as a string, so any node name must be assigned a sequence number anyway, and after reading it in the simulator program, essentially all node names in the netlist must be replaced with the sequence number. This is a time consuming process and needs to be done after each load of the netlist.

The SIC is detached from the Core. The output of the SIC is a binary file that can be processed much faster than a text file. Core is created so that can get multiple netlists as input, which can be considered to be concatenated netlists. In this way, Frame can put the netlist created based on the structure and the simulation statements in separate files. Thus, if the same structure is to be simulated e.g. with different excitations or boundary conditions, it is sufficient to modify the simulation instructions.

SIC can also process standard SPICE netlists.

If the result of the simulation is distribution (temperature distribution, voltage distribution, etc.), then we can currently use the classic SUNRED GUI for display, which requires distribution maps in FIM files. This conversion is performed by the "to FIM" module. Of course, this module can only be used for VSUN input files, not for separately created SPICE netlists.

### Integrated future GUI

Creating a GUI that can take advantage of the simulator’s capabilities is very necessary. In SUNRED, the GUI and simulator were separate programs, communicating through files. The advantage of this is that they can be developed completely independently of each other, even in other programming languages.

For Hex Open, the GUI and simulator will be integrated.

Core

GUI

[Extended

SPICE

Netlist]

(optional)

GUI

Structure

Description

Simulation

Results

The reasons for integration are:

* While the aggregate file size of the SUNRED (and VSUN) model is typically up to a few hundred kilobytes, Hex netlists can typically be tens of megabytes or even several gigabytes. Saving them to a file and loading them from there can be very time consuming. The description is worth passing directly to Core in memory.
* During the simulation, the Core can provide continuous feedback on the progress and the results obtained with the simulation, so the user can see on a chart how the examined quantity changes e.g. in a transient analysis over time. Thus, if necessary, the user can stop the simulation and start again with modification, or continue from a saved intermediate state with different settings.

In the integrated version, Core does not perform direct file management, it is entirely up to the GUI. The advantage of this is that the description of the model and the storage of the results can be much more compact than in the command line version. Optionally, the GUI can be instructed to provide an extended SPICE netlist, which the user is free to modify and simulate with the command line simulator, but the GUI cannot display the result because the spatial structure of the simulated model cannot be restored from the netlist.

It's worth adding a tool to the GUI that can display a curve or a map from a text file, and implement the Core to save the results in this format. This allows the GUI to display the results of a command line simulation.

Also it's worth adding a tool to the GUI that can run the command line tools, this would allow users who are less familiar with the command line to work comfortably. In this tool, the user could specify the (extended) SPICE netlists to be simulated (up to several at a time) and the tool would perform the simulation.

It is already clear from this that the GUI can have a lot of features. These are worth organizing into separately feasible tools. E.g. required netlist editor; 3D designer; a cell designer that maps a netlist to an elementary volume; a material parameter handler that can be supplemented by managing more and more physical properties; selection of the physical parameters to be simulated and setting of the boundary conditions and excitations; circuit simulator package; devices for displaying finite simulation results; etc. Before implementing the GUI, the basics of its modular structure must be carefully planned, in the first round only the most necessary parts will be implemented.

Core + 3D model to netlist + some user interface can be a program. The main part of the GUI can be written in other language. (WPF, plugins)

### Structure of the Hex Open Core

The structure of Hex Open Core is designed to handle instruction flow. The instruction flow means that it is not necessarily known in advance what will happen during the simulation. Structure description instructions and then simulation instructions arrive, but then structure description or modification instructions may arrive again. For example, the GUI requests a simulation, then requests some structural or parameter modification after the first results, and then continues the simulation.

Model

Controller

Result

Controller

RP

Save/Load

Instruction

Processor

Simulation

Controller

RP

Scheduler

Network

Model

Simulation

NR Model

SUNRED

Intermed.

File Reader

Result

Saver

Instruction

Packages

from the GUI

Instruction Packages

RP files

Intermediate

Files

Result

Files

GUI

Parallel

Engine

Instructions can come from two sources in the form of instruction packages: intermediate files or directly (e.g. from GUI). An instruction package can contain one type of instruction (structural, simulation, other). The end of the instruction stream is indicated by an end-of-stream instruction for direct instructions and by the File Reader after reading the last file. The files are received by the Core as a command line parameter, only they are processed, no new files can be added afterwards. To continue the execution from a Restore Point (RP) with new simulation files is possible

The Instruction Processor puts the instruction packages in a queue. When a package is executed, the Instruction Processor sends the next package to the appropriate controller for execution.

Incoming instructions may include RP save / load instructions. These may include an instruction requesting RP saves at regular intervals. Regular saves are started by RP Scheduler. RP saving can only take place after a simulation step. If it arrives during / after a model instruction, the Instruction Processor waits for the first simulation step and only then executes it. RP loading can take place at any time.

Instruction packages for creating and modifying models are sent by the Instruction Processor to the Model Controller. The Model Controller creates the Network Model, sets the active / inactive parts.

The Network Model contains the components of the circuit, the connections, the blocks, the actual, previous and starting values (node voltages, component currents, component values) in DC (real) and AC (complex). It builds and maintains the Newton-Raphson (NR) model and the data structures required for Successive Network Reduction (SUNRED). Provides the simulation results required for saving to the Result Controller. Saves and loads itself as instructed by the RP Save / Load unit. The Parallel Engine executes parallelizable tasks on as many threads as the CPU of the computer running the program can handle.

Különböző dt-hez tartozó modellek tárolhatósága: konfig fájlból olvassa, hogy hány dt-hez tárolhat modellt. Automatikus visszalépésnél egy előre definiált halmazból vesz dt értékeket, így véges számú lehetséges dt-t használ. Számolja, hogy melyik dt hányszor szerepelt, és a leggyakoribb n db-ot tárolja.

The simulation results are handled by the Result Saver. The data save process consists of two steps. In the first step, the actual values are copied to be saved (multithreaded). In the second step, the items to be saved are saved / forwarded to the GUI in groups (single thread). The Instruction Processor waits until the first step is completed. The second step, on the other hand, goes in parallel with the processing of the following instructions, but if another save call is received, the Instruction Processor waits until the second step is completed. When a node delete command is received, it also waits until the end of the save.

The Simulation Controller breaks down the simulation instructions into basic steps and controls their operation. For example, a step in a nonlinear DC simulation or in a nonlinear transient simulation is one simulation instruction, but the implementation requires multiple iterations. It makes sure that the error has reached the desired level. Should the time step be reduced in the case of a transient calculation? The scheduled Restore Point save instructions can be inserted after basic simulation steps and not only after a simulation instruction, this is why the Simulation Controller is integrated with the Instruction Processor.

A basic step can consist of four operations: updating controllers, current calculation, Newton-Raphson, updating voltages.

Simulation

Simulation Controller

Controller

Calculation

Update Voltages

*Ui\_act* = *Ui\_prev* + *dUi* ∙ *α*

Update

~~Parameters~~

Components

Calculation of

Branch Currents

Calculation of

Nodal Error Currents

Create / Update

Admittance Matrices

Forward

Substitution

Backward

Substitution

Current Calculation

Newton-Raphson

Controller frissítése: ~~prev = act minden kontorllerre~~, felesleges a prev, a controller bemeneti változói lemásolják a bemeneti értéküket, hogy újraszámoláskor ne legyen versenyhelyzet más kontrollerekkel => minden controller újraszámolása

When updating controllers, the controllers calculate their output values based on the input values. The controllers are evaluated in order; the next controller uses the updated output value of the previous controller. If we do not want to use the updated value, we can use .prev or .start. If the chain of controllers has been designed so that a change does not go through them in one step, the controller update step can be repeated several times in a loop.

The current calculation step can be divided into three further smaller steps. During the parameter update, the value of the dependent components is updated, e.g. temperature-dependent resistors or voltage / current-dependent sources. The current of each branch of the circuit is then calculated and the inflow and outflow currents in the nodes are summed. According to Kirchhoff's law, the sum of currents in each node must be 0, however, a current error other than 0 usually occurs during the calculation. One of the conditions for stopping the iteration is a sufficiently low error current.

In the Newton-Raphson step, the differential version of the circuit is formed at the operating point calculated in the previous iteration, and the error current is taken into account as the excitation. We calculate the differential circuit voltages using the Successive Network Reduction Method in the current version of Core. Differential circuit voltages also mean error voltages, one of the conditions for stopping the iteration is that its maximum absolute error voltage falls below a limit.

In the Update Voltages step, the differential node voltages are added to the node voltages obtained in the previous step. If we find that the convergence is not adequate, we do not add the whole differential node voltage, but only a part of it. Thus, normally the value of alpha will be 1, but in case of a convergence problem 0 < alpha < 1 will be used. Calculation of alpha is not detailed here. (See Dumped Newton Method.)

Az update components lépés kiszámítja a komponens \_value, \_i és \_v értékét.

Az áram és a hibaáram számítás egyszerre történik.

* Minden instance a belső csomópontokra kiszámítja a ∑(JBi\*JBi) áramhibát, amihez hozzáadja a tartalmazott instance-ok áramhibáját. ~~Ha az aktuális subckt instance-ra (pl. global circuit) sunred van előírva, akkor nem számol most JB-t, így az áramhiba csak a komponensek áramhibáinak összege lesz (azaz részleges, de ez a Hex LED-ben is így volt).~~ (Nem sunred-nél nem párhuzamosíthatjuk az áramszámítást, vagy gondoskodni kell az atomizálásról.)
* Minden instance kiszámítja a JA vektort a nem vezérlő bemeneti csomópontokra.

The flowchart shows a typical nonlinear DC simulation or a step of a transient simulation:

Update Controllers

Current Calculation

Newton-Raphson

Update Voltages

Update Controllers

Current Calculation

Converged?

New α?

y

y

n

n

It is also possible to use the quasi-Newton method when the Jacobi matrix remains unchanged over several iterations.

### NR modell építése

Blokk összevonási fa felépítése

Blokkok külső (és belső) csomópontjainak meghatározása, halmazba gyűjtése a levélelemektől felfelé haladva. Külső csomópont az, amihez másik blokk is csatlakozik. Lehet normal vagy centroid, de ezt a sorrend meghatározásánál nem nézzük. Itt még tetszőleges sorrendben kerülnek be a csomópontok sorszámai a halmazokba. A partícionálás alakítja ki a tényleges sorrendet.

Csomópont halmazok partícionálása:

* Bal blokk külső csomópontjainak partícionálása.
  + Ha van szülőelem, annak (partícionált) csomópontjai alapján
  + Bal ág alapján
  + Jobb ág alapján
  + A partícionálás bármikor megállhat, ha minden partícióban egy csomópont marad
* Jobb blokk külső csomópontjainak partícionálása a bal blokk külső csomópontjai alapján
* A bal blokk két gyerekének partícionálása (rekurzív)
* A jobb blokk két gyerekének partícionálása (rekurzív)

A lépések után a partícionált halmazok sorrendben tartalmazzák a csomópontokat, ez alapján a honnan hová hányat lépések meghatározhatók. Figyelni kell a centroidokra.

Egy összevont blokkból egy csomópontra csak egy kimenet kapcsolódhat, ez az összevonás következménye. Mivel egy összevonásban két blokk érintett, egy centroid csomópontra a két összevonandó blokkból 1-1 kimenet kapcsolódhat. A 0. szintű blokk létrehozásakor lehetséges, hogy egy csomópontra egy komponens több kivezetése is kapcsolódik.

### Vezérlő bemenet ill. vezérlő csomópont automatikus felismerése

A subcircuit-ek belsejének cserélhetősége miatt nem érdemes automatikus felismerést alkalmazni.

Lógó vezérlő csomópontok lehetnek, pl. T csomópontra kapcsolódik egy hőmérsékletfüggő ellenállás egy tisztán elektromos cellában. Ilyenkor a csomópont alapértelmezett feszültsége a vezérlő érték.

### Vezérlő csomópontok redukciója

Ha normal módon számítjuk egy subcircuit admittanciamátrixát és hibaáramát, akkor a vezérlő csomópontokhoz tartozó admittanciákat is figyelembe vesszük. Ha sunreddel, akkor ezeket 0-nak tekintjük, kivéve, ha amúgy is ki vannak vezetve a subcircuit-ből. (A controllerek sosincsenek benne az admittanciamátrixokban.)

## New and extended structure elements and implementation questions

### Kifejezések

A komponensek értékének helyén állhatnak. Komponene node-jának helyén nem állhatnak. Ha node-ot akarunk vezérelni kifejezéssel, akkor külön definiáljuk a kifejezést .expression-nel.

Tartalmazhatnak konstansokat, változókat, csomópontokat, függvényeket, +,-,\*,/,^, és előjel aritmetikai műveleteket, <,>,<=,>=,==,!=,&,|,! logikai műveleteket. Elfogadja az && és || operátort is.

A node/változóneveket tehetjük v(N1) függvénybe, így az esetleges ütközések elkerülhetők. A függvények nevét .-tal jelölhetjük. pl. .sin(), .v(), stb. Ez nem kötelező. A node/változónevek elé opcionálisan tehetünk @ karaktert, ha betűvel vagy aláhúzással kezdődnek, de kötelező, ha számmal kezdődnek, kivéve, ha v()-ben vannak. A v()-nek van egy és kétparaméteres változata, utóbbi a két parameter különbségét adja.

### Indirekt azonosítók

A csomópontok, változók komponens.node módon megadhatók. A globális áramkör elemei :node1 ill. :komponens.node módon hivatkozhatók. Csak egyszintű indirekció megengedett, tehát a komponens.komponens.node nem. (Az .actual / .prev / .stepstart nem számít indirekciónak.)

### Component templates

Minden komponens egy komponenssablon példánya. Az alapkomponenseknek alapsablonja van. A subcircuit típusú példányosítás minden sablonra működik, az alapsablonokra is:

x1 n1 n2 \_R(30k)

Mivel a spice nevek alfanumerikusak, aláhúzás nincs bennük, ezért aláhúzással kezdődnek a beépített sablonok.

Beépített sablonok:

.component \_R N1 N2 value => ellenállás   
.component \_RD N1 N2 T value => disszipáló ellenállás   
.component \_C N1 N2 value => kapacitás   
.component \_OPEN => szakadás   
.component \_V N1 N2 value => feszültségforrás   
.component \_VR N1 N2 Rvalue , Vvalue => feszültségforrás belső ellenállással   
.component \_I N1 N2 value => áramforrás   
.component \_IR N1 N2 Rvalue, Ivalue => áramforrás belső ellenállással   
.component \_HYS >T (Tmin Tmax Twidth) => hiszterézis   
.component \_PCC T >H (Ephase\_change Volume) => phase change capacity

Saját sablon:

.component TRth1 N1 N2 >T (base\_value gamma) \_R N1 N2 base\_value\*exp(gamma\*(v(T)-25))  
.component TRth2 N1 N2 >T (base\_value) \_R N1 N2 base\_value\*exp(-0.0025\*(v(T)-25))

A saját sablonok belseje ugyanúgy cserélhető, mint a subckt belseje. Pl.:

Ez kötött: .component TRth1 N1 N2 >T (base\_value gamma)   
Ez cserélhető: \_R N1 N2 base\_value\*exp(gamma\*(v(T)-25))

.replace TRth1 \_R N1 N2 base\_value+gamma\*v(T)

Vagyis csak a fejléc utáni részt írjuk le újra, ami cserélhető.

A .component egysoros, és a hozzá tartozó .replace is egysoros. A .subckt többsoros, a hozzá tartozó .replace is többsoros.

Ha nincs parameter, zárójel akkor is kell a beolvashatóság miatt. Pl. egy fix ellenállás sablon:

.component TR1 N1 N2 () \_R N1 N2 (32k)

Ha a szülő parameters, akkor így definiáljuk:

.component M5Hys >T () \_HYS (61 70 6) T   
.component M5Hys >T () \_HYS (61 70 6) T-5

Controller template csak controller template-re cserélhető, komponens template pedig csak komponens template-re. A controller template-nek csak kontroll node-jai lehetnek. => itt mit is akartam mondani? A fenti példában a \_HYS egy komponens sablonba kerül.

* Beépített cellasablonhoz: többértékű komponens lehetősége. Az elektromos rész hőmérsékletfüggése, a termikus rész hőmérsékletfüggése, a hőkapacitás hőmérsékletfüggése így oldható meg. A kifejezéseket pl. vesszővel lehetne elválasztani. A saját hőmérsékletét kell vezérlésre felhasználni, így a hőmérsékletet ~~kontroll kimenetként~~ **normál bemenetként** meg kell adni. Pl:   
  .component C1 N1 N2 N3 N4 N5 N6 (x y z) \_ThCell1 N1 N2 N3 N4 N5 N6 T (x y z) .inv(2\*exp(7.8e-5\*(T-25))), 2.5Meg

Ha egy belső komponens admittanciamátrixa

,

ahol az 1-2 a normál node-ok, a 3 a vezérlő node, akkor, ha a vezérelt node-ot f(U3) kifejezéssel vezéreljük, akkor a komponens mátrixa ez lesz (V3=U3):

.

Ha két node-ról is vezéreljük:

.

Sima ellenállásnál, ha J1=G(V3)\*V1, (V1=U1-U2):

.

Termikus ellenállásnál (W1=T1-T):

.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A fentieket újra levezetve a Székely könyv alapján:

Csak vezérelt áramgenerátorokat tartalmaz a hálózat.

i: ág (áramhoz)

j: csomópont (áramhoz)

s: csomópont (potenciálhoz)

r: ág (feszültséghez)

A *Kij* jelentése: az i. ág a j. csomópontra kapcsolódike-e mint végpont (+1 vagy -1), ha nem, 0.

*Ellenállás:*

*G(U3)*

*1*

*2*

*3*

1 ág van, *K1j* = 1, ha j = 1, *K1j* = -1, ha j = 2, *K1j* = 0 egyébként, mert az ellenállás az 1 és 2 csomópontok közé kapcsolódik, vagyis az admittanciamátrix második sora az első sor -1 szerese, a 3., 4., … sora pedig csupa 0, ezért fölösleges tárolni.

Ha a vezérlő csomópont megegyezik az ellenállás valamelyik csomópontjával, akkor a harmadik oszlopban lévő tag az első vagy a második oszlophoz adódik, és persze nincs harmadik oszlop. Az első két oszlopban lévő G és -G esetén a negatív előjel az (U1-U2) U1-gyel ill. U2-vel való deriválásából addik, tehát a második oszlophoz nem negatív előjellel adódik a harmadik oszlop.

Több vezérlő csomópont esetén annyi (U1-U2)dG/dUi adódik, ahány vezérlő csomópont van, mindegyik a megfelelő Ui-vel deriválva.

*Áramgenerátor:*

*I(U3)*

*1*

*2*

*3*

*Disszipáló ellenállás 1:*

*G(U4)*

*1*

*2*

*4*

*I(V1)*

*3*

J1 = G(U4)∙(U1-U2)

J2 = -G(U4)∙(U1-U2)2

K11=1, K12=-1, K23=1, a többi 0.

*Áramgenerátor a disszipáló ellenálláshoz:*

*G*

*2*

*3*

*4*

*I(V1)*

*1*

J= -U4∙(U1-U2)2

K11=1, a többi 0.

Mivel a 4-es csomópont vezérlő csomópont, az ellenállás admittanciáját jelenti, a hozzá tartozó Y értékek kiesnek egyesítéskor.

*Disszipáló ellenállás 2:*

*G(U4)*

*1*

*2*

*4*

*I(V1)*

*3*

A fenti „*Ellenállás*” és „*Áramgenerátor a disszipáló ellenálláshoz*” összekapcsolva:

Az ellenállás hőmérsékletfüggése nem jelenik meg a generátor áramában.

*Kapacitás:*

*G(U3)*

*1*

*2*

*3*

*I*

G = C(U3) / ∆t

I = -G(U3)∙V0

J = G(U3) ∙ (U1-U2-V0)

*Feszültségforrás*

*G0*

*1*

*2*

*3*

*I(U3)*

J = G0 ∙ (U1-U2-V0(U3))

*G0(U4)*

*1*

*2*

*3*

*I(U3)*

*4*

J = G0(U4) ∙ (U1-U2-V0(U3))

*Ellenállásos áramgenerátor*

*G0(U4)*

*1*

*2*

*3*

*I0(U3)*

*4*

J = G0(U4) ∙ (U1-U2) – I0(U3)

*Mind*

Majd ellenőrizni kell, hogy az előjelek jók-e (főleg a deriváltas részben).

### Rövidzár

Node-ok G=∞-t eredményező rövidre zárása nem megengedett. Ez akkor van, ha két tényleges node-ot összekötünk. Ezt a szintaktika elvileg nem is engedi, úgyhogy nem tudok példát mutatni.

Pl. ez viszont megengedett:

.component A B C () xxxx A B A C

Ekkor egy külső csomópontra kötjük rá a belső komponens két csomópontját, ami egy sima redukció.

### Non-transparent custom components

Ha valamelyik terminált földeljük, akkor az ahhoz tartozó sort és oszlopot elhagyjuk a belső komponens admittanciamátrixából és inhomogén áramvektorából, így kapjuk a befoglaló komponens mátrixait. Azaz *JRED = JA, YRED = YA, IA = YRED \* UA + JRED, UB = 0*.

*Tamb*

Ha fix feszültségre kapcsoljuk (pl. *TAMB*), akkor így lesz: *JRED = JA + XB \* TAMB, YRED = YA, IA = YRED \* UA + JRED, UB = TAMB*. Ha vezérlő csomópontot kötünk fix feszültsére, akkor az XB 0 lesz, ezért *JRED = JA* lesz.

Az összekapcsolt j-ket össze kell adni (júj = jm+jn), ill. az Y megfelelő sorait és oszlopait össze kell adni (a metszéspontban 4 y összege, a többi helyen 2 y összege).

Sima redukció. Több csomópont összekapcsolása esetén a megfelelő sorok és oszlopok összeadódnak.

### T0 and thermal nodes (Txxxx)

~~It is advisable to define a node T0 connected to ground with a voltage source with a value of \_Tamb (VT0 T0 0 DC \_Tamb). This will be the thermal ground node.~~ The temperature of the thermal ground is @\_Tamb (default 25°C).

Special grounding nodes are denoted by $0...$9. $0 is the same as the plain 0 ground node. $1 is a node with the value \_Tamb. $2...$9 is not currently implemented.

~~In the implementation, for each component connected to T0, a voltage source with a voltage of \_Tamb is placed between T0 and ground, so that there are no additional terminals from the sub-circuits. (All global voltage generators connected between a global node and ground are implemented on a cell-by-cell basis unless the generator current is used for control.)~~

If the name of a node starts with T, it is considered as a thermal node, and \_Tamb is considered as initial value of these nodes. It is the same as specifying each such node with the .NODESET command to \_Tamb. This helps the program find the dc or initial transient solution by making a preliminary pass with the specified nodes held to the given temperature. The restriction is then released and the iteration continues to the true solution. This preliminary setting can help the convergence. If the name of a node only starts accidentally with a T, there is no problem either, as it will get to the actual value in the iteration. A .NODESET command for the node overrides the default (\_Tamb) value.

### Floating nodes, non-connecting sub-circuit external nodes and empty sub-circuits

If nothing is connected to a node, the node voltage will be set to 0 or \_Tamb.

If only control inputs are connected to a node, the node voltage will be set to 0 or \_Tamb. (This is the case when a cell is simulated only electrically but the material is temperature-dependent: the material temperature will be considered as Tamb.) => not allowed => undefined behaviour

If no internal component is connected to an external node in a sub-circuit, a 1 TΩ resistor is placed between the external node and ground (0 or \_Tamb). => not allowed => undefined behaviour

If an external node of an implemented sub-circuit is connected to a node alone, a 1 TΩ resistor is placed between the node and ground (0 or \_Tamb). => not allowed => undefined behaviour

If a sub-circuit is empty (e.g., open boundary condition), a 1 TΩ resistor is placed between each external node and the ground (0 or \_Tamb). => not allowed => undefined behaviour;

\_OPEN component is allowed, it is implemented as a 1 TΩ resistor

### Irányított ellenállás

Egyik irányban máshogy vezet, mint a másik irányban.

### Dissipator

A non-linear dependent current source, it can be connected to a resistor or a diode.

SPICE definition:

B 0 1 D=R5

It can also be defined in the traditional way (except .actual), it means the same but easier to specify:

B 0 1 R5.actual\*V(6,8)^2

It "cools" when connected upside down.

### Hysteresis

A built in hysteresis model is defined as a parametric subcircuit (controller). An example for its usage:

XH T H DefHyster(57 62 7)

Where T is the input (temperature) node, H is the output (control) node, *TH1* = 57°C, *TH2* = 62°C, *SZ* = 7/2 = 3.5°C.

Phase-change capacity is also defined as a subcircuit.

XC T H DefPhaseChangeCapacitor(236.5e+006 1.35e-18)

Where T is the thermal node, H is the control input from hysteresis, the phase-change energy is 236.5 MJ/m3, the volume of the cell is 1.35e-18 m3.

### Values of nodes, components and variables

Kell-e prev, ha act+alfa\*dU-t ad vissza a lekérdezés?

Components, nodes, and variables must have different names (in a namespace). Thus, when saving, if a node is specified, the voltage, if a component is specified, the current is saved.

Each node, ~~component~~, and controller variable has three (real) values: .actual, .prev, and .stepstart. E.g. N1.actual, R1.prev, C1.stepstart. A normal (non-controller) variable has only the actual value.

.actual can be omitted, optional.

Each component has a .\_i and a .\_v value, the current and the voltage of the component. These values are calculated in the Update Voltages and Current Calculation steps so they can be considered as ex-post values: if we use them for control, we will take into account their value calculated in the previous iteration.

Here is an example to understand how these work:

|  |
| --- |
| Starting point: start of a DC simulation or start of a new step of a transient simulation (the previous time step has converged and we have accepted as a valid value, now we move forward by another delta t time and start calculating the circuit voltages for the new time).  xxxxxxxx.stepstart = xxxxxxxx.actual  xxxxxxxx.prev = xxxxxxxx.actual  Here xxxxxxxx means all nodes, components and variables.  Update Controllers => controller variables get new actual values  [If more than one Update Controllers instruction has been issued] => controller\_variables.prev = controller\_variables.actual => Update Controllers => controller variables get new actual values  Current Calculation => components get new actual values  Newton-Raphson  Update Voltages => nodes get new actual values  Update Controllers (see above) => Current Calculation (see above)  Converged? (Uses |prev-actual|) Yes => nodes.to\_be\_saved = nodes.actual  (If |prev-stepstart| is too large in transient, we start the whole step again, but with a smaller delta t, we will not detail this now.)  New alpha?  Yes => xxxxxxxx.actual = xxxxxxxx.prev (i.e. undo) => goto Update Voltages  No => xxxxxxxx.prev = xxxxxxxx.actual => goto Newton-Raphson |

The variable to\_be\_saved is needed because the save is done by a separate thread while the next step is calculated. Thus, the calculation certainly does not corrupt the data to be saved. The nodes.to\_be\_saved = nodes.actual step only occurs if the previous data has already been saved. If not, the program will wait for the save to complete.

The following list shows what the value means for each component.

|  |  |
| --- | --- |
| Resistor | Resistance [Ω] |
| Capacitor | Capacity [F] |
| Source | Current [A] / Voltage [V] |
| Diode | Resistance [Ω] |
| Subcircuit | Not a component, value exists but always 0 |
| Controller | Not a component, value exists but always 0 |

### SUNRED

A backsubs kiszámítja a feszültségeket, elindítja a két, alatta lévő backsubs-ot, majd beírja a megfeleő node-ok dU-jába a kiszámított feszültségeket, így nem less ütközés a szálak között.

In the SUNRED block we can define the steps of successive network reduction. If a SUNRED block is defined in a subcircuit or globally, the active SUNRED blocks are executed when running any analysis. If no SUNRED block is defined, a system of equations is created and calculated for the entire subcircuit/global circuit.

.sunred NAME   
 Bxxxxxx X1 X2 // a block contain exactly two components or blocks   
 B2 X4 X5   
 B3 Bxxxxxx B2   
 .endsr

The SUNRED block contains block definitions. A block definition can only consist of (a) component (e.g. sub-circuit) instances (not templates) or (b) blocks defined in the same SUNRED block. Within a SUNRED block, a component or a block may be contained by only one block directly (indirectly it may be contained by several blocks).

Nonlinear dependent source instance names start with B in the same way as blocks. It is also possible to mark such a component as a SUNRED step source, but its name must not conflict with the name of a block.

A SUNRED block can be enabled, disabled and deleted (destroyed).

The SUNRED block naming convention is the same as for the sub-circuit. The names of the blocks must start with B. This resemblance to the .subckt should not deceive anyone: the .sunred contains existing components, while the .subckt is a template that needs to be instantiated.

By default, all SUNRED blocks are enabled.

Disabling / enabling:

.disable NAME1 NAME2 …   
 .enable NAME3 NAME4 …

Several SUNRED blocks can be active at the same time, there can be connections between them. The last part of network reduction is the merging of active SUNRED blocks.

If a SUNRED block is disabled, its associated NR models will remain in memory. This is useful when using SUNRED blocks to alternately calculate e.g. the electric and thermal field. Because data structures remain in memory, there is no need to free up and reallocate dynamic memory, saving time. However, if the memory is low or the NR models are no longer needed, the associated dynamic memory can be deleted:

.delete NAME1.DC   
 .delete NAME2.AC

The .DC or .AC suffix is required.

NR models can be DC or AC. The DC model represents all simulations when calculating with real numbers, and the AC model means everything when working with complex numbers. If we delete NAME1.DC, but e.g. if NAME1 and NAME3 together form the NR model, the entire common model will be deleted. The deletion to NAME3 can then be issued as well, re-deleting an already deleted model is not a problem. If at any time after deleting we request a simulation of the SUNRED block again, the simulator will create the NR model. Thus, deleting the NR model never causes a malfunction, at most it slows down the simulation.

If a SUNRED block is no longer needed, for example because we are replacing it with another model, we can delete it permanently with the .destroy statement to free memory.

.destroy NAME1

### Enabling, disabling, destroying components

Any component can be switched off by disabling. If we do, the model network will not contain that component in the next simulation step. However, the nodes to which the component is connected will remain, so care must be taken to keep the network regular. If there is a problem with the remaining nodes, it is recommended to place the circuit part in a sub-circuit and turn off the whole sub-circuit.

By default, all components are enabled.

Disabling / enabling:

.disable NAME1 NAME2 …   
 .enable NAME3 NAME4 …

The internal nodes of the disabled component are accessible, queryable, and therefore can be used for control and saved, but their values will not change during the simulation steps.

If a component is no longer needed, we can delete it permanently with the .destroy statement to free memory.

.destroy NAME1

.destroy deletes all internal nodes, so we can no longer refer to them later. If you accidentally refer to the destroyed component, it may cause the simulator program to crash.

### SUBCKT replacement

.replace subname   
 R1 1 2 10k   
 …   
 .ends

The inside of the previously defined subcircuit named "subname" is replaced by the code. Each instance of the subcircuit is replaced from the next iteration. Can be used e.g. to replace a boundary condition with another. The replacement (for the part outside the sub-circuit) must not change the topology of the circuit.

Az eredeti függvény NameToIndex-ét használjuk => az azonos node-ok / változók / subckt-ek az eredetit használják. Az újak új indexet kapnak, a nem használtak is a memóriában maradnak, hátha egy csere után újra kellenek.

A subckt mellett a component és a controller is cserélhető (és a modellt is célszerű lenne). A .replace mindig globális, de teljes útvonallal megadható a cserélendő definíció.

Csak a node-ok és változók őrzik meg az indexüket és tartalmukat, minden más cserélődik.

~~Only the global subcircuits can be replaced, the local ones not.~~

### Parametric SUBCKT

Paraméterként csak konstans, vagy egy másik parameter szerepelhet, kifejezés nem lehet. (Elvi akadálya nincs, hogy kifejezés legyen, később megvalósítható ez is, de csak létrehozáskor kiértékelődő kifejezésre.)

Finite models are characterized by a large number of cells, but these cells come from some of the same types. Many cells differ from others only in that the values of the components differ. The simpler, shorter netlist is aided by the parameterizability of the subcircuits.

.subckt subnam N1 N2 … Nx (P1 P2 … Pn)   
 R1 0 1 P1\*10k   
 R2 1 2 @P2\*12k   
 R3 1 3 P3   
 R4 1 4 @P4   
 .ends

The same rules apply to the format of parameter names as to node names. If the parameter name begins with a number, if it is referenced, the @ character must be placed in front of the name. If it starts with a letter, @ is optional.

Usage:

X1 2 4 (0.3n 0.5n 11u 2.6)

### Control input ~~and output~~ for SUBCKT

.subckt subnam N1 N2 >N3 ~~<N4~~   
R1 0 N4 10k   
R2 N1 N4 5k\*N3   
R3 N2 N4 8k   
.ends

If a '>' is placed before the node parameter, it is a control input. The control input is for control purposes only. ~~If a '<' is prepended to the node parameter, it is a control output. The control output makes the value of an internal node available for control of another component.~~ Ezt a funkciót megvalósíthatjuk az x1.n1 lehetőségével. => **Ha a Jakobi mátrixban is meg akarjuk jeleníteni, akkor normál external node-ként valósítjuk meg, és csak vezérlő bemeneteket kapcsolunk rá.** ~~If two ‘>’ is placed before a node parameter (>>), it is an indirect control input, uses the voltages calculated in the previous step. Indirect inputs do not appear in the Jakobi matrix, so computing one iteration step will be faster, but it is possible that achieving the desired precision will require more iteration steps (Newton-Raphson vs. successive approximation).~~ The input of ~~an indirect~~ a control input can be node, variable, internal node or variable of another component, including controllers.

Először a normál csomópontok jönnek, utána a control bemenetek, nem keveredhetnek.

Ahol nincs semmi, ahhoz sor és oszlop is tartozik a Jakobi mátrixban, a > bemenethez csak oszlop tartozik (nem folyik rajta áram)~~, a >>-hoz semmi sem.~~ (A teljes redukció használja a > bemenetekhez tartozó Jakobi értékeket, a sunred nem.)

X1 1 2 3 4 subnam   
X2 5 6 4 3 subnam

Only nodes 1, 2, 5, 6 are involved in network reduction, nodes 3, 4 are not. Nodes 3, 4 are also created. Node 3 implements the interior (N4) node of X2 and node 4 implements the N4 node of X1. Normal terminal of a component cannot be connected to nodes 3 and 4.

Control inputnak megadható a hívás helyén lévő másik objektum belső csomópontja is (csak egyszeres indirekció megengedett). Pl.

Xaaa 1 2 3 4 valami   
Xbbb 2 6 Xaaa.N5 masvalami

### ~~Two-terminal subcircuits~~

~~In the netlist file, the two-terminal subcircuits do not appear separately, this only matters in the implementation.~~

~~Each basic component is realized as a descendant of the two-terminal ancestor class. These components can also have additional control inputs. Unlike normal subcircuits, they usually do not have an internal admittance matrix and reductions, they only provide their admittance and other values externally.~~

~~The boundary conditions and excitations are placed in subcircuits. When the core is read, it recognizes that these simple subcircuits can be implemented as two-terminals and creates them accordingly. Changing the boundary condition or excitation to another type can be accomplished by replacing the subcircuit.~~

~~The break can be implemented as a zero value current source.~~

### Variables

A controller változóinak van prev és stepstart értéke is, a subckt változóinak nincs, azok egyértékűek. (A program mindkét félét tudná használni mindkét helyen, de jelenleg nem látom értelmét.)

Variables store a value that can be changed, similar to variables in programming languages. Hex variables always store real numbers (not integer, not complex, not string). The variables can be used e.g. to store the value of components; in .if, .while, .for control structures; to control controlled sources, etc. The initial value of the variables is 0.

The naming convention for variables is the same as for nodes. In Hex, the nodes themselves can be used for control, similar to variables. The essential difference between variables and nodes is that (a) a variable cannot have a component terminal attached to it, except for the control terminal, (b) no .prev and .stepstart values, the value is changing when a .set instruction arrives, and (c) a variable is always subcircuit level and not instance level. If the value of a variable is changed, it is changed for all instances. (Like static variables in C++ classes.)

There are also variables in the controllers, but these are different from normal variables. Controller variables are more similar to nodes: they have .actual, .prev, and .stepstart members, and their new state is restored with a new alpha or a new time step. No components can be connected to the controller variables, except for the control terminals, similarly to the normal variables.

The .set statement can be used to define a variable or change its value.

.sstatic NAME1 [initial\_value] => csak kívülről állítható   
.mstatic NAME1 [initial\_value] => csak kívülről állítható   
.svar NAME1 [initial\_value] => csak controller, csak belülről állítható   
.mvar NAME1 [initial\_value] => csak controller, csak belülről állítható   
.svar NAME1 [initial\_value] // only in the global space => sstatic-kal megvalósítva   
.mvar NAME1 [initial\_value] // only in the global space => mstatic-kal megvalósítva   
.set NAME VALUE

E.g.:

.mvar i   
 .set i i+1   
 .set i @i+1   
 .set 68 3.14   
 .set 68 @68+2

R1 0 1 @68

A kontrollerben a változódefiníció sora statikus változó esetén csak példányosításkor fut le, később nem. .set csak nem statikus változóra megengedett a controlleren belülről. Nem statikus változó esetén a definíciókor megadott értékadás a controller minden kiértékelésekor végrehajtódik, ahogy a .set utasítás.

The default value is 0. If the parameter name begins with a number, if it is referenced, the @ character must be placed in front of the name. If it starts with a letter, @ is optional. If used in a place where there is a thing with the same name, e.g. i in the expression of the controlled nonlinear source, then @ is also mandatory.

### Constants

The naming convention for constants is the same as for variables.

.const e 2.7172

R5 6 2 e

### 

### Predefined variables and constants

To avoid conflict with user defined SPICE names, the predefined variables and constants always contain the leading @.

|  |  |
| --- | --- |
| @\_Tamb | Variable, ambient temperature = thermal ground, default = 25°C |
| @\_pi  @\_e  @\_k  @\_q  @\_koq  @\_Ta | 3.141592653589793  2.718281828459045  1.38064852e-23  1.602176634e-19  8.61733026622e-5 (=k/q)  273.15 |
| @\_time | Only in case of transient simulation, otherwise 0.  @\_time.stepstart  @\_time or @\_time.actual  (@\_time.actual = @\_time.stepstart + @\_dtime) |
| @\_dtime | Only in case of transient simulation, otherwise 0.  @\_dtime or @\_dtime.actual |
| @\_freq | Only in case of frequency domain (AC, Bode, …) simulation, otherwise 0.  @\_freq or @\_freq.actual |

### The .if statement in the netlist description

It is possible to conditionally define components. It can be used e.g. to massively change the interior of cells. For example, if we know in advance that we will perform a simulation with different types of boundary conditions in a boundary cell, we can put the circuit parts that implement the boundary condition in .if.

An .if block cannot be empty.

Elements of the .if statement:

.if condition   
 <component definition(s)>   
 .elif condition   
 <component definition(s)>   
 .else   
 <component definition(s)>   
 .endif

E.g.

.If @1<@2   
 R1 1 2 10k   
 .else   
 R2 1 2 11k   
 .endif

Explanation: If the voltage of node 1 is lower than the voltage of node 2, R1 will be defined, otherwise R2. The name of the components must be different in the if and else branch.

Attention! If the inside of the .if is used for control, the appropriate control instructions must also be provided for the controlled component, otherwise malfunctions may occur. For example, if we define a dissipator for R1, the dissipator will dissipate based on the value of resistor R1 and the voltage at nodes 1 and 2 even if R1 is currently turned off in the .if instruction.

Another example:

.if diodestate==1   
 R1 1 3 14.2k\*@diodecurrent   
 .else   
 D1 1 3 …   
 .endif

Where diodestate is a global variable:

.set diodestate 1   
 .iterate   
 .set diodestate 0   
 .iterate

The condition can be complex:

.if A > B && (C <= D || E == F)

Here && means AND, || means OR. & and | also can be used. Precedence is the same as in the C or C++ language.

.if statements can be nested.

Loop (.while, .for) cannot be used for netlist description.

If-else components can be placed inside subckt, so their enabled / disabled status can be handled in the same way as components. In the if-else definition, all components are created, but are placed in enabled / disabled status.

### Controllers

összetett művelet (=vezérlő): műveletek sorozata + lokális változók

művelet: kifejezésfa, nincs saját lokális változója, azaz nem emlékezik

a vezérlőben a kifejezések sorrendben értékelődnek ki, ugrás lehet if-fel vagy ciklussal

a vezérlőben kétféle utasítás van: setter vagy ugró

A kontrollernek kizárólag controller vezérelt kontroll bemenetei vannak, ezt nem muszáj jelölni a SPICE fájlban.

The definition and instantiation of controllers is similar to that of subcircuits.

.controller ctrlnam >N1 >N2 … >Nx (P1 P2 … Pn)   
 …   
 .endc

Xxxxx 0 x21.3 5 ctrlnam

Controller instance names must begin with an X.

Each of the external nodes of the controllers is an indirect input node. The controller outputs (which are the internal instance variables of the controller) can only be accessed indirectly, via instance.variable-name.

A controller that does not have an input node is also conceivable.

E.g.

.controller sqrt in   
 .svar out .sqrt(in)   
 .endc

Xsqrt n3 sqrt

Esource n5 0 Xsqrt.out 0 1.0

The internal nodes, input, and output nodes of the controllers have .actual, .prev, and .stepstart values. Their initial value is 0.

The hysteresis model of the Hex LED simulator implemented with a controller:

.controller Hyster >T (T1 T2 W)

.const W2 W\*0.5

.const TK T1-W2

.const TV T2+W2

.const TM T2-T1

.const TN T1+W2

.mvar H

.svar TC T1+H.stepstart

.svar TL TC-W2

.svar TR TC+W2

.if(T<=TK)

.set H 0

.elif(T>=TV)

.set H 1

.elif(T<TL)

.set H (T-TK)/TM

.elif(T<=TR)

.set H H.stepstart

.else

.set H (T-TN)/TM

.endif

.endc

XH 1 2 Hyster(57 62 7)

A controller may contain other controllers but not other components (e.g. resistor or voltage source cannot be in a controller).

E.g.

.controller HysVO2 T H   
 XH T H Hyster(57 62 7)   
 .endc

XV 1 2 HysVO2

### Subcontrollers => needed? Not supported yet, should be revised.

Nem láncolhatók így. A kontrollerek bármilyen sorrendben kiértékelődhetnek => a kontorllerek kiértékelése kétfázisú. Először a bementi változók/node-ok aktuális értéke lemásolódik, utána a másolatból dolgozik. A prev nem szükséges, a stepstart viszont használható.

When instantiating a controller, you can specify that it be a subcontroller. The subcontroller works the same as a normal controller, but it is guaranteed that the subcontroller is evaluated earlier than its master and its output values are stable when its master reads them, even if the subcontroller is in another cell. (The subcontroller is evaluated by the same process as the controller. First the subcontroller, then the controller. If a subcontroller (A) also has a subcontroller (B), then B is evaluated earlier than A by the same process.)

Xxxxx N1 N2 … Nn .sub(CONTROLLER\_TYPE OWNER)(P1 P2 … PN)

E.g.

X2 1 2 .sub(Hyster X83625.X1)(57 62 7)

The input of a controller can be connected to a node, variable or controller. If connected to a node, node.actual will indeed be the current node value. If it is a (sub) controller, then it depends on the relationship between the two of them:

**A**

**B**

out

in

|  |  |  |  |
| --- | --- | --- | --- |
| "B" | "A" | A.out.actual | A.out.prev |
| Any | "A" is "B" (connects to itself) | previous | previous |
| Controller | Controller in the same subcircuit, defined before B | updated (actual) | previous |
| Controller | Controller in the same subcircuit, defined before A | previous | previous |
| Controller | Controller in other subcircuit | - (not available) | previous |
| Controller | Subcontroller owned\* by "B" | updated (actual) | previous |
| Controller | Subcontroller not owned by "B" but whose ancestral owner is in the same subcircuit as "B" and the ancestral owner is defined before "B" | updated (actual) | previous |
| Controller | Subcontroller not owned by "B" but whose ancestral owner is in the same subcircuit as "B" and the ancestral owner is defined after "B" | previous | previous |
| Controller | Subcontroller not owned by "B" and whose ancestral owner is in other subcircuit than "B" | - (not available) | previous |
| Subcontroller, owned\* by “A” | Any | previous | previous |
| Subcontroller, not owned\* by “A” | Any | Instead of "B", its ancestral owner should be examined, see the corresponding "Controller" line | previous |

\*directly or indirectly

E.g.

XA 1 2 type   
 XB 3 4 .sub(type XA)   
 XC 5 6 .sub(type XB)   
 XD 7 8 .sub(type XA)

XA has two direct descendants (XB and XD) and one indirect (XC). XB and XD each have an owner (XA), which is also the ancestral owner. XC has two owners (XA and XB), of which XA is the ancestral owner.

XE 9 10 type   
 XF 11 12 .sub(type XE)

If "A" is XD and "B" is XF, then we have to look at the relationship of XE ("B") to XA ("A").

### (Sub)controller updates

If a controller is in a disabled sub-circuit, it will not be updated, its output values will be the values valid at the moment of disabling until the controller is re-enabled. A subcontroller will be updated, if its owner is enabled

To update, the controllers are stored in a comb list (list of lists). A comb tooth contains the controllers in a cell, plus the associated external subcontrollers. The tooth can be an array.

### Equation given piecewise

Use a controller with .if statements. Piecewise linear equation can be given by the .pwl function.

### Updating normal nodes and controller nodes, error calculation

The normal nodes and controller nodes are handled differently. Normal nodes are updated in the Update Voltages step, controller nodes are updated in the Update Controllers step.

Current and voltage errors are calculated only on the basis of normal nodes, controller nodes are not taken into account.

### Initialize a node from another node

A list of which node from which can be specified.

It is also possible to specify a ratio list so that one node becomes the weighted average of several nodes.

Only the actual or all values should be calculated.

Thermal nodes should be initialized by the frame / GUI to ambient temperature.

### Compact models

They are managed at the GUI level.

.subckt compact w0000 w0001 w0002 …   
 .subckt sR1 N1 N2 (P1)   
 R1 N1 N2 P1\*20 // 20Ω is divided   
 .ends sR1

\* here come the elements of the compact model

X0000 w0000 N1 sR1(80000)   
 X0001 w0001 N1 sR1(80000)   
 X0002 w0002 N1 sR1(40000)   
 …   
 .ends compact

### Quasi-Constant Nonlinear components

~~A normal temperature-dependent resistor:~~

~~R47 N23 N24 32.65k\*.exp(-0.007\*@N5)~~

~~The value of this resistor is calculated in each iteration.~~

~~R47 N23 N24 .qcnl(32.65k\*.exp(-0.007\*@N5))~~

~~The value of this resistor is only calculated at the beginning of the step. It retains its value in each iteration within a given step, so there is no need to perform network reduction. It is worth using for small nonlinearities.~~

~~It can also be used in DC simulation, in which case e.g. it is worth running 2-3 DC simulations one after the other.~~

Túl bonyolult megoldani. Ha a kifejezés tudja venni a parameter node stepstartját, akkor ezzel megoldható. Lehet, hogy többet számol, viszont a memóriamenedzsment elhagyása miatt akár meg is érheti.

### Math functions

.inv, .sqrt, .exp, .ln, .log, .abs, .asin, .acos, .atan, .asinh, .acosh, .atanh,

.sin, .cos, .tan, .sinh, .cosh, .tanh, .ratio, .pwl, .qcnl, .u, .uramp, v, i

square: a^2   
power: a^b   
u: unit   
v: v(x), v(x,y)

A function name always starts with a letter. The initial letter can optionally be preceded by a period to help distinguish the function name from other names.

.sqrt(X) = or sqrt(X) =

.abs(X) = |X| or abs(X) = |X|

.exp(X) = eX or exp(X) = eX

.inv(X) = 1 / X or inv(X) = 1 / X

.ratio(X1,X2,H) = X1 \* H + X2 \* (1 - H) or ratio(X1,X2,H) = X1 \* H + X2 \* (1 - H)

.pwl(X,X1,Y1,X2,Y2,…) = piecewise linear, X is the actual x value; X1-Y1, X1-Y2,… are the x-y pairs. Unlimited pairs can be used and can have non-constant values, but they must remain consistent, i.e. X1<X2<X3... must always be true.

### Samples

.subckt VO2R N1 N2 T H (MUL)

R1 N1 N2 .inv(.ratio(.pwl(T,25,20,58,54),65000,H))\*MUL

.ends

MUL is L/A

### Expressions

.expression name (n1+n2)^2

R1 N1 N2 @name+5

A kifejezés olyan, mint egy egysoros controller, ami a name nevű változót állítja be. A Jakobi mátrixhoz nem jarul hozzá, így ha lehet, akkor komponenst használjunk helyette.

### Lines beginning with X

A .subckt, a .controller, a .component mindegyike X kezdetű sorban példányosodik. Ezeknek nincs értéke, tehát kifejezést nem írhatunk ebbe a sorba. Vezérlő bemenete viszont kapcsolódhat .expression típusú kifejezésre.

### .forward

Egy komponens belső változójának vagy node-jának a továbbítása vezérlési célból. Így kivezethezők mélyben lévő értékek is.

Pl.

subckt E1 blahblah   
X1 E1 n2 somesubckt   
.forward X1.A A   
X2 n2 n3 A   
ends

Xaaa N223 blahblah   
Xbbb N256 0 Xaaa.A subckt8

A .component automatikusan továbbítja a szülő komponens összes belső értékét, azonos néven. (Mivel .component csak beépített komponensből származhat, ezek közül pedig csak a \_HYS-nek van belső értéke, a H.)

## New and extended control operations and implementation questions

### The .if, .while and .for statements in control operations

The .if statement can be used in the same way as in the netlist description.

An .if block cannot be empty.

The loop statements:

.while CONDITION   
 …   
 .endwhile

.for NUMBER   
 …   
 .endfor

E.g.

.while A<5   
 .set A A+1   
 .iterate   
 .endwhile

.for 5   
 .iterate   
 .endfor

.if, .while, .for statements can be nested.

A vezérlő utasítások a változók összes adatát beállítják a megadott értékre, hogy visszalépésnél a beállított érték maradjon. (Controllerben az ott kiadott set-ek csak az aktuálisat, de a globális vezérlő utasítások ott is!)

### Implemented analysis types

.op

.timestep 2m

### .solutionmethod

.solutionmethod DIRECT | SUNRED

If not defined: if SUNRED is defined for the subcircuit/global circuit, SUNRED will be used, otherwise DIRECT.

### Result saving

.probes probename N1 N2 N3 .text("V1=") N4

.probes probename N5 N6.prev 0 N7 .intvalue(64) .realvalue(3.14) R1.\_V R1.\_I R1.\_value

.saveto text/binary overwrite/append probename filename

.delete probename

If the probename is the same in more than one .probes line, these probes will be appended one after the other. If you would like to replace a probe to another one, you must delete first.

When a .saveto command is given, hex\_open\_core starts the binary file with an "R" or "C" text (see the definition of “text” bellow), meaning real or complex data type. (If append is used, the appended part starts with this letter.) If text file is used, the complex data will be written as 8.32+7.45i.

The binary file contains the data in 4-byte blocks. A block can be a single precision float value, a signed integer or text.

A text in a binary file is one byte=size of the (ASCII) text + the text itself. The maximum length of the text is 255 character. If the length of the text cannot be divided by four, the remaining spaces will be filled with 0 bytes.

* a binary file starts with a byte of value 1 + the ‘R’ or ‘C’ character + two bytes of value 0.

A próbok fejét még azelőtt kiírjuk, hogy szimulációt futtattunk volna! Tehát szabad eredményt menteni szimuláció nélkül is, ezt meg kell oldani. (Esetleg nem tényleges eredményt ment, hanem ahol node kell, ott nullákat.)

### Az iteráció menete

Ha az egész rendszer konstans, akkor elég egy DC lépés.

Egyébként:

Kezdő DC: minden forrást ki kell kapcsolni, csak a Tamb-ra kötöttek maradjanak. => kéne örök idők óta működö bekapcsolt forrás, és azok is be lennének. => iteráció hibahatárig. Hibahatár elérése előtt a kvázi-konstans kifejezéseket egyszer (kétszer?) elengedni.

DC vagy tranziens lépés:   
- források beállítása az új lépésnek megfelelően   
- tranziensnél, ha volt változás a forrásokban, akkor egy nagyon kicsi dt-vel iterálunk stabilitásig   
- normal iteráció stabilitásig (tranziensnél, ha kell, visszalépéssel)   
- stabilitás elérése előtt a kvázi-konstans kifejezéseket egyszer (kétszer?) elengedni

### Subckt / Controller példányosítása

Megkapja a külső csomópontok címeit és a paramétereket.

Tudja a belső csomópontok és példányváltozók számát => lefoglalja ezeket.

Létrehozza a belső komponenseket.

### Bug list

Amikor egy komponens vezérlő node-jára nem kifejezésként, hanem node-ként adtam meg az 1e-2 konstanst, akkor ezt nem vette észre, helyette 1e node nevet feltételezett, a következő parameter -2 lett. Ez nem elég robosztus, továbbá az internalNodesIsThermal vektort nem állította be, tehát az 1e nevű lokális node-ot nem kezeli jól. Lehetséges, hogy a csak vezérlőre kapcsolt node-okkal baj van.

A SubCircuitInstance::initAndRefresh-ben a forwarded kezelést meg kell nézni: a preIdentifyNode és az identifyNode a SubCircuitInstance-é vagy a komponensé kéne legyen? (Azaz a sc->forwarded[i] mit azonosít? Szerintem a subckt-é kéne legyen, és a nodeId a komponens+node azonosítást kéne használja.)

Az expressionCtrlNodesIndex-ben benne vannak azok az internal node-ok is, amelyekre csak egy vezérlés csatlakozik, semmi egyéb, azaz konstans értékűek. Problémát nem okoz, csak felesleges munkát és memóriahasználatot. Nem kritikus a javítása.

.component M5Cth T >H (MUL) \_C T \_Tamb .ratio(1.82Meg,3.54Meg,H)\*MUL => Itt a \_Tamb-ot lokális node-ként azonosítja, 0 értékkel inicializálja. A \_Tamb elvileg egy változó, tehát normál node-ot nem istehetnék rá, de akkor mi a termikus fold? Nincs ilyen? Miért eszi meg gond nélkül a \_Tamb-ot? (Az aktuális forrásfájlban .component M5Cth T >H (MUL) \_C T 0 .ratio(1.82Meg,3.54Meg,H)\*MUL van, ott a 0-t földként azonosítja, ami rendben is van, de valamit kezdeni kéne a termikus földdel.)

Talán inkább úgy kéne definiálni a spice fájlt, hogy ha van valamilyen bejegyzés, akkor teszi a T kezdetű csomópontokat termikus földre? Vagy elég, ha a .TAMB definiálva van 25-re, és a default 0?