Behind the scenes of the SPICE Circuit Simulator

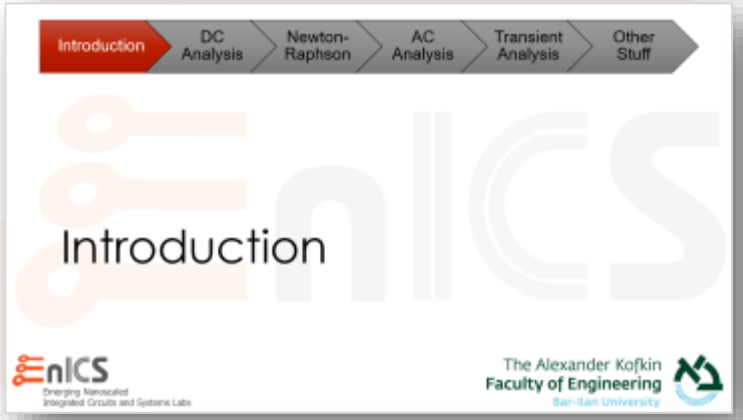
Prof. Adam Teman

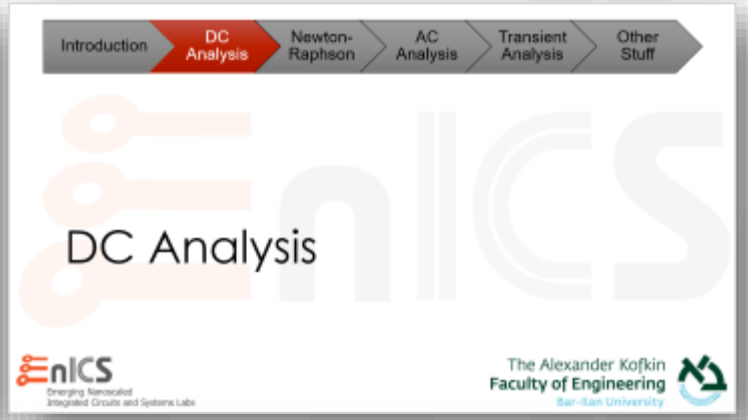
2 April 2022

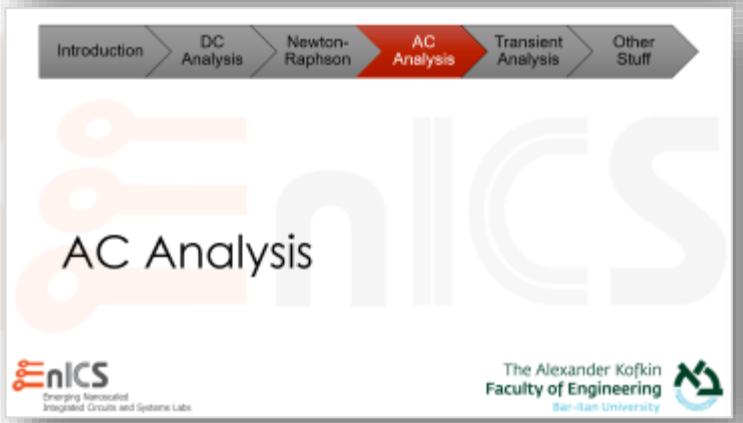
Emerging Nanoscaled
Integrated Circuits and Systems Labs

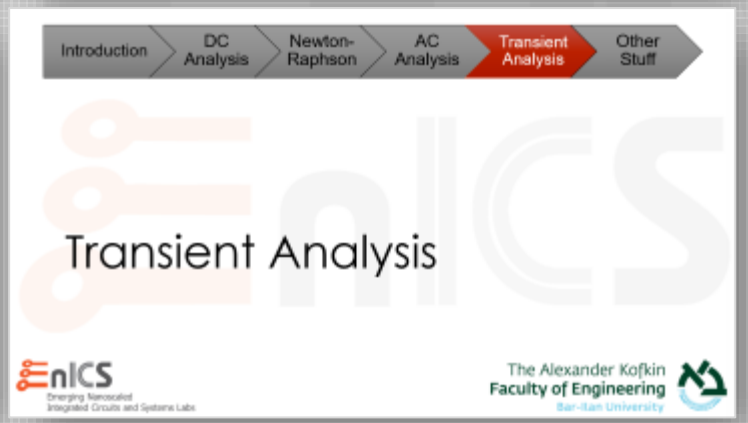
The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

Overview





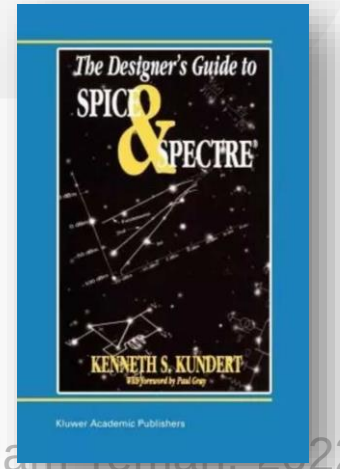




Introduction

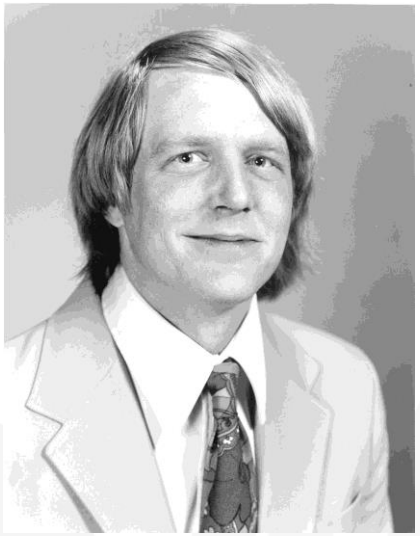
Motivation

- **Are you a "Spice Monkey"?**
- **According to Kenneth Kundert, there are two types of circuit designers:**
- **Reactive users:**
  - Run the simulator and hope that nothing goes wrong.
  - If something goes wrong, try different "tricks" hoping that one will solve it.
  - If not solved, redesign circuit to avoid the problem or don't use simulator…
- **Proactive users:**
  - Anticipate the problems.
  - When one occurs, knows why and what to do.
- **Let's turn you from *reactive* into *proactive* users!**



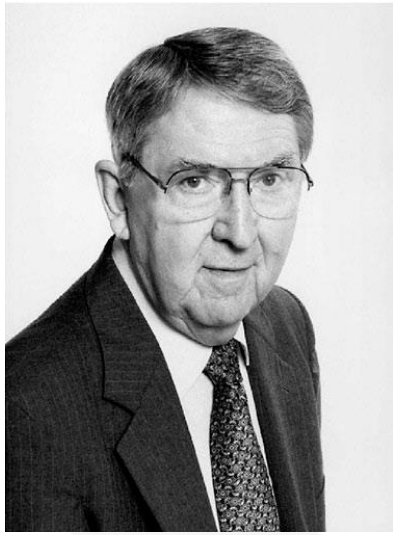
History

- **Circuit simulators started to appear in the late 1960s and early 1970s**
- **Two major contributors:**
  - IBM ASTAP group
  - Berkeley SPICE group
- **SPICE started as a class project of Prof. Ron Rohrer** (called CANCER)
  - 1972: SPICE released by Larry Nagel (under Prof. Don Pederson)
  - 1975: SPICE2 released, 1989: SPICE3 released
- **Why did SPICE succeed?**
  - It was targeted at Integrated Circuit design
  - The source code was made available (for a small price)
  - It was disseminated by Berkeley graduates

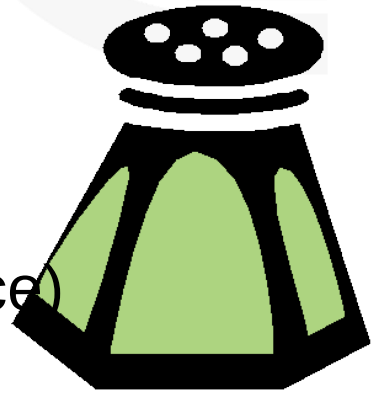


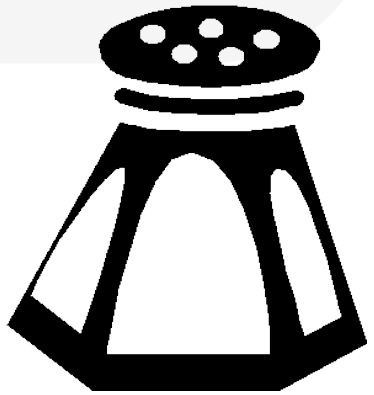
Larry Nagel

Ron Rohrer



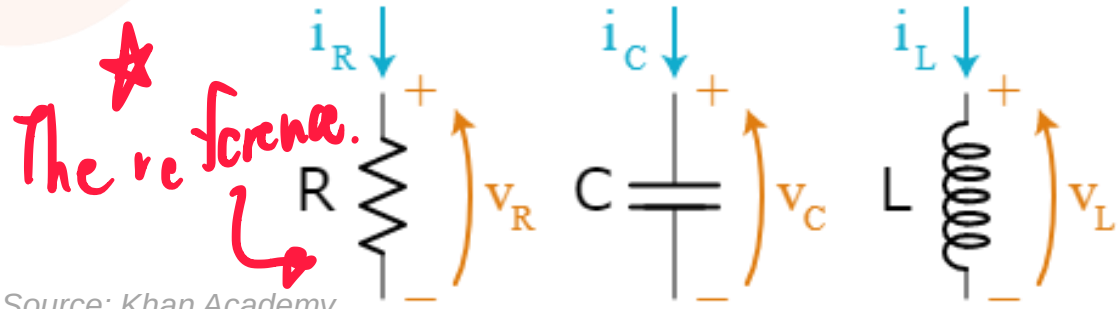
Don Pederson





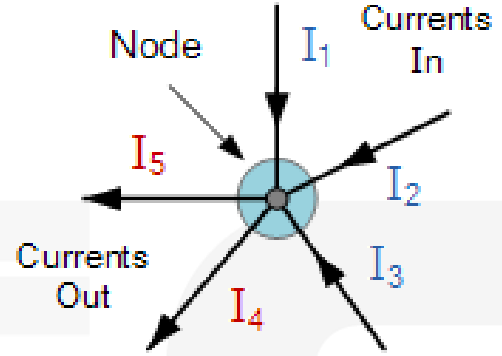
Reminder: Kirchoff's Laws

- **Kirchhoff's Current Law (KCL)**
  - The sum of all currents flowing out of a node at any instant is zero.
- **Kirchhoff's Voltage Law (KVL)**
  - The algebraic sum of all branch voltages around a loop at any instant is zero.
- **Associated Reference Direction**
  - Current runs from the high potential (+) to the low potential (-)



Source: Khan Academy

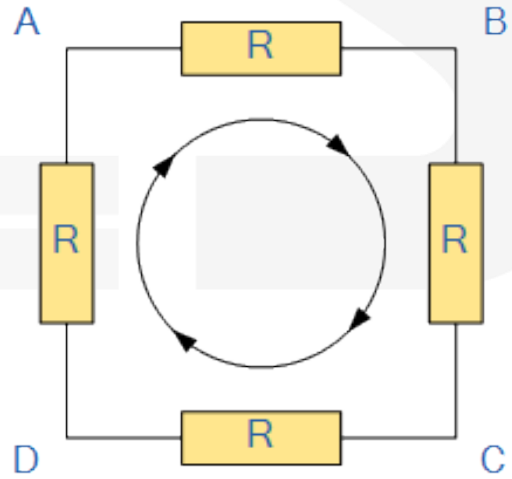
Currents Entering the Node
Equals
Currents Leaving the Node



$$I_1 + I_2 + I_3 + (-I_4 + -I_5) = 0$$

Source: ElectronicsTutorials

The sum of all the Voltage Drops around the loop is equal to Zero



$$V_{AB} + V_{BC} + V_{CD} + V_{DA} = 0$$

Circuit Simulation

- **A circuit simulator is provided with:**
  - Circuit connectivity by means of *netlist*.
  - Component behavior by means of *device models* and *model parameters*.
  - The initial state of the circuit, known as *initial conditions*.
  - Something that is input to the circuit, called *stimulus*.
- **This information is used to:**
  - Construct a set of ordinary differential equations that describe the circuit.
  - Solve the equations using *nodal analysis**, which for a circuit containing resistors, capacitors and current sources is simply:

current (i) entering nodes (v) from resistors

current sources

$$i\left(v(t)\right)+\frac{d}{dt}q\left(v(t)\right)+u(t)=0$$

charge (q) entering nodes (v) from capacitors

$$v(0)=a$$

initial condition

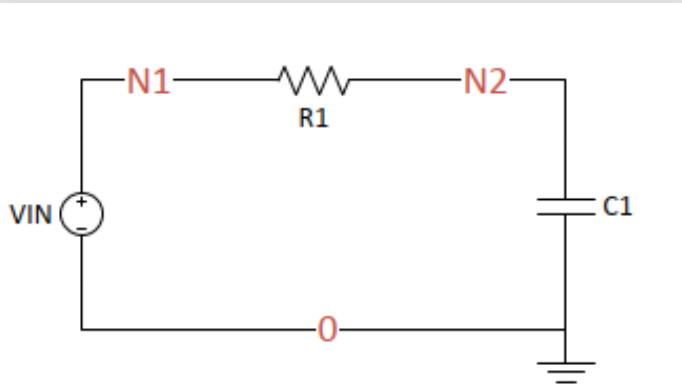
*In practice, modified nodal analysis is used

Main Analysis Types

- **DC Analysis**
  - Find the DC operating point of the circuit, i.e., all voltages and currents.
  - Since it is not possible to explicitly solve a system of nonlinear algebraic equations, the systems are linearized and solved using Newton's method.
  - This is an iterative process the run to convergence.
- **Transient Analysis**
  - Since there is no known method to solve the nonlinear differential equations, the simulator discretizes time.
  - In other words, it solves a DC Analysis for each timestep based on initial conditions.
- **AC Analysis**
  - Find the DC operating point and assume linearity for small signals.

Example: A simple RC Circuit

```
1  RC Circuit
2
3  *** SETTINGS ***
4  simulator lang=spice
5
6  *** NETLIST ***
7  ** Parameters **
8  .PARAM vdd=5 *Supply voltage
9
10 ** Voltage Source **
11 *VIN N1 0 AC 5 SIN(0 5 1MEG)
12 VIN N1 0 PULSE(0 vdd 1u) *5V pulse with 1us delay
13
14 ** Elements **
15 R1 N1 N2 50
16 C1 N2 0 1u
17
18 *** Analysis ***
19 *.AC DEC 10 0.1 100MEG
20 *.PRINT AC V(N2)
21 .TRAN 1n 200u *200us simulation with 1ns steps.
22 .MEAS TRAN tau TRIG V(N2) VAL=0.000001 RISE=1
23 +.  TARG V(N2) VAL='0.63*vdd' RISE=1
24 .PRINT TRAN V(N2)
25
26 .END
```



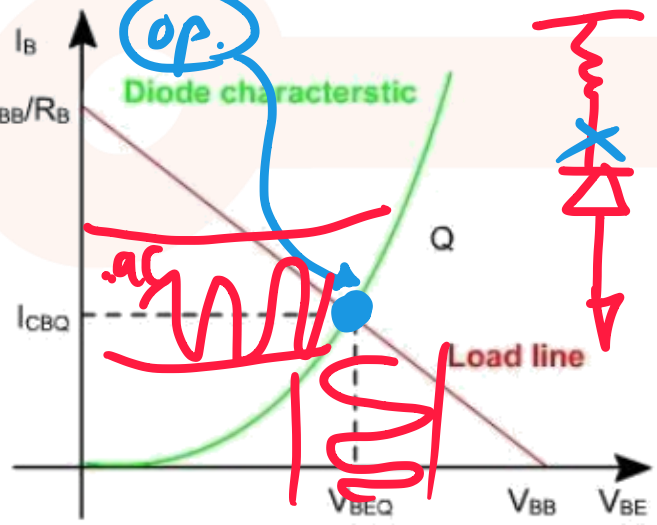

```
spectre rc.cir
```

DC Analysis

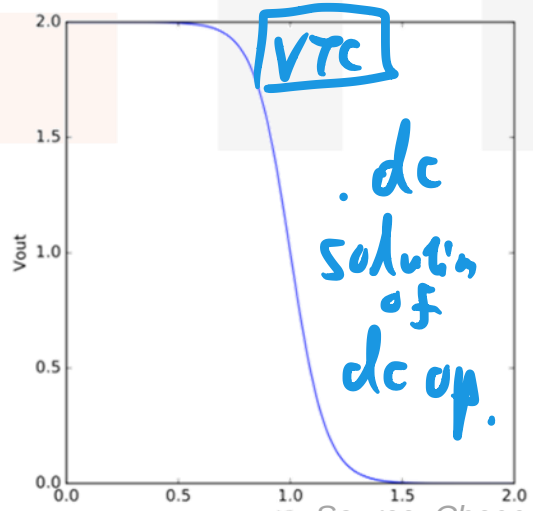
Why DC Operating Point?

(In spice, every is just DC operating point)

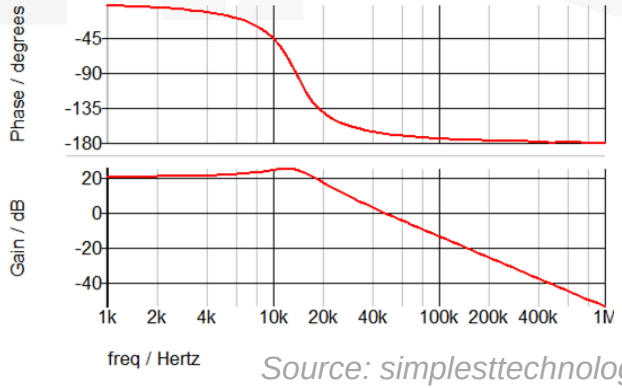
- **Every simulation starts with calculation of the DC Operating Point (DC OP).**
  - `.op` calculates the voltage, current, power, etc. at each component.
  - `.dc` computes the op point as a function of some independent variable. (DC sweep)
  - `.ac` and `.noise` first compute the DC OP and the linearize the circuit around it to compute the small-signal behavior of the circuit.
  - `.tran` computes the DC OP for an initial state of the circuit. Transient simulations are then basically a collection of sequential DC Ops, starting with these initial conditions.



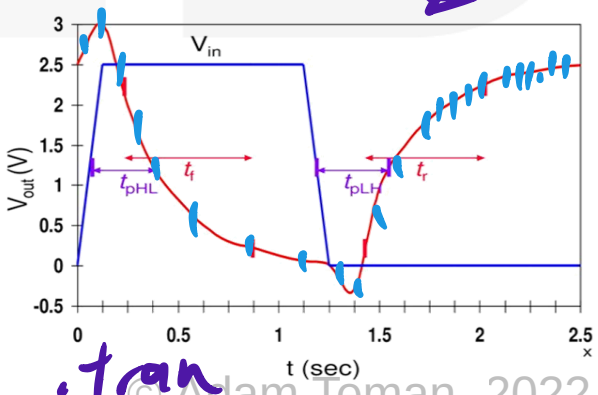
Source: Brain Kart



Source: Chegg



Source: simplesttechnologies



DC Analysis Starting point

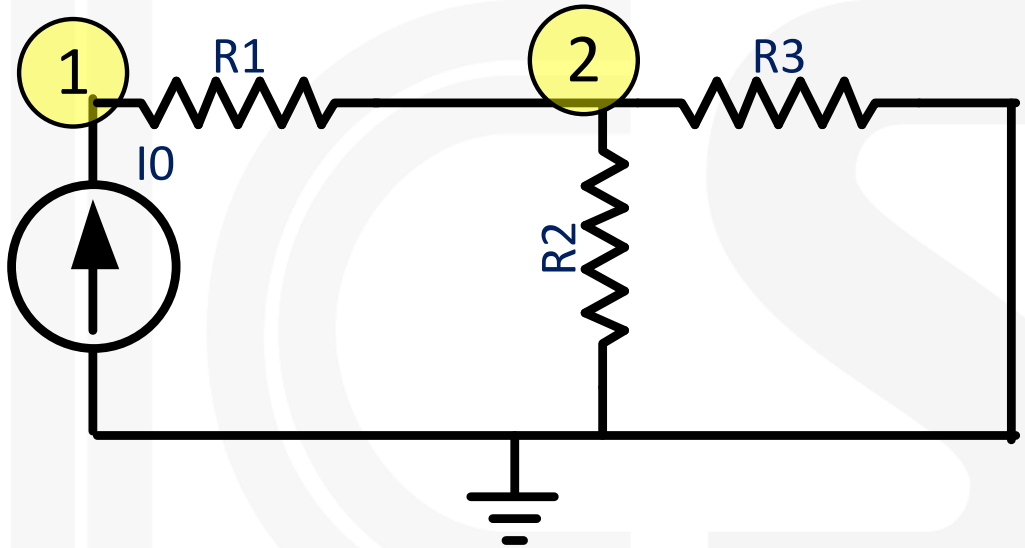
- **DC OPs are equilibrium points, i.e., do not change with time**
  - Independent sources are configured to be constant.
  - $dv/dt=0$ → capacitors are open circuits.
  - $di/dt=0$ → inductors are short circuits.
- **DC OP Starting State**
  - All Caps and Inductors are removed.
  - DC Sources values are assigned.
  - Node Sets are assigned.
  - Time varying part of signal sources is ignored.
  - Initial Conditions are ignored.
- **Now Nodal Analysis can be applied.**

$$i(v(t))+\frac{d}{dt}q(v(t))+u(t)=0$$
$$v(0)=a$$

$$\frac{d}{dt}q(v(t))=0 \quad \rightarrow \quad i(v_{dc})+u_{dc}=0$$

DC OP Nodal Analysis

- **We will write Kirchoff's Current Law (KCL) using *branch constitutive equations*.**
  - Node 1: $\frac{(V_2 - V_1)}{R_1} + I_0 = 0$
  - Node 2: $\frac{(V_1 - V_2)}{R_1} - \frac{V_2}{R_2} - \frac{V_2}{R_3} = 0$
- **Lets now replace with $G=1/R$ and we get:**



$$G_1(V_2 - V_1) + I_0 = 0$$

$$\hookrightarrow \quad G_1 V_1 - G_1 V_2 = I_0$$

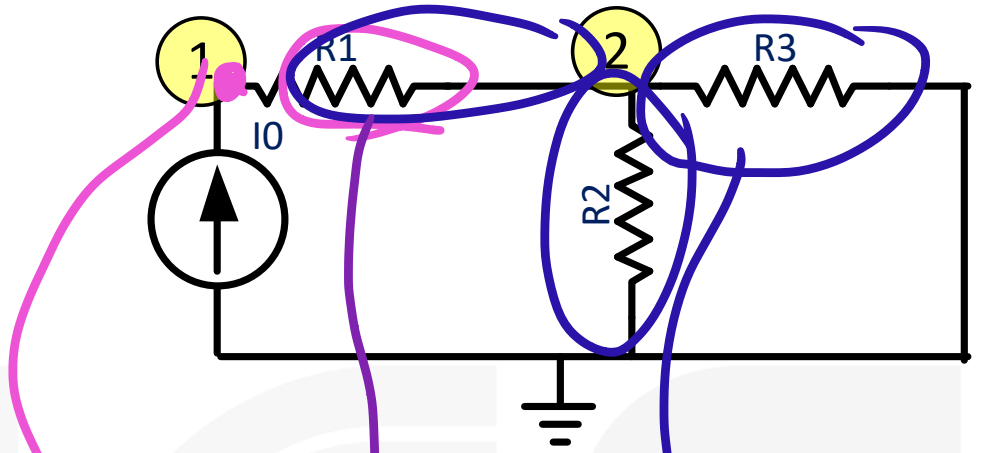
$$G_1(V_1 - V_2) - G_2 V_2 - G_3 V_2 = 0$$

$$\hookrightarrow \quad -G_1 V_1 + (G_1 + G_2 + G_3) V_2 = 0$$

$$\begin{pmatrix} G_1 & -G_1 \\ -G_1 & G_1 + G_2 + G_3 \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \end{pmatrix} = \begin{pmatrix} I_0 \\ 0 \end{pmatrix}$$

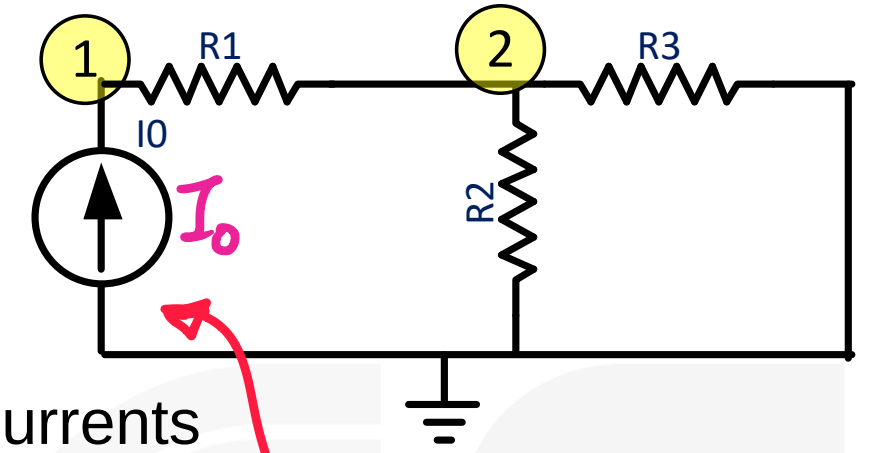
DC OP Nodal Analysis

- **We can directly write the conductance matrix:**
- **Diagonal Elements:**
  - For element a_{ii} add all conductances attached directly to node i.
- **Off Diagonal Elements:**
  - For element a_{ij}, add all conductances connected directly between node i and node j. Multiply by -1.
  - For most off-diagonal elements there are no direct connections, so we get a *sparse matrix*, which is easy to solve.



$$\begin{bmatrix} G_1 & -G_1 \\ -G_1 & G_1+G_2+G_3 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$$

DC OP Nodal Analysis



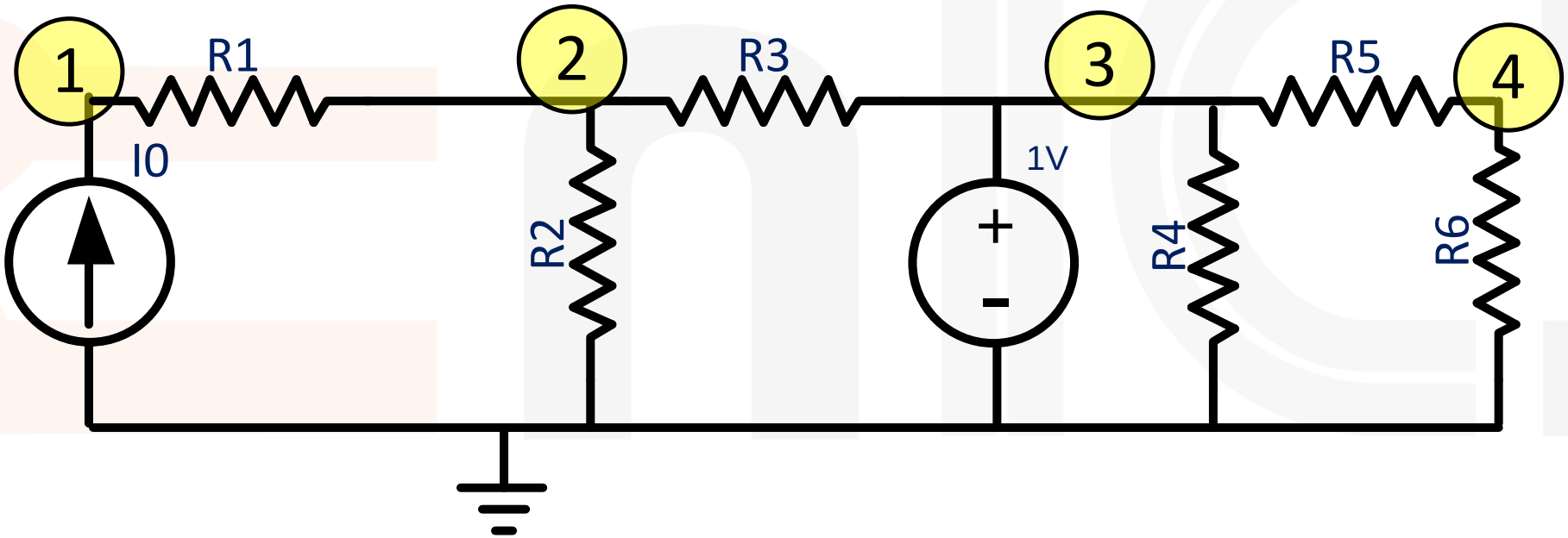
- **Source Vector *(I)*:**
  - For element *i* of the source vector, sum all source currents flowing into *node i* and subtract all currents flowing out of *node i*.
- **Modified nodal analysis**
  - Nodal analysis doesn't work with voltage sources
  - However, we know the voltage on its node, so we get an equation such as $V_1=V_{in}$.
  - But we don't have an ohmic relationship on the voltage source, so we add another unknown to the voltage node vector *(V)*, i.e. I_1
- **Let's see this by example**

★ Conductance matrix

$$\begin{bmatrix} G_1 & -G_1 \\ -G_1 & G_1+G_2+G_3 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} I_0 \\ 0 \end{bmatrix}$$

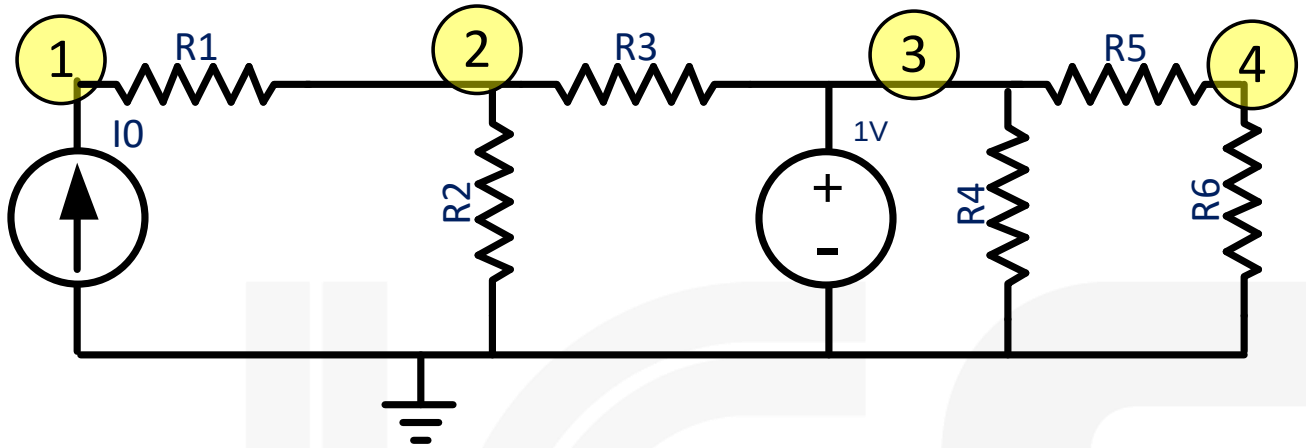
→ Linear system created ⇒ Solvable by

Modified Nodal Analysis Exercise



Solution

- **Start with diagonals**
  - Add the conductances for each node.
- **Next the off-diagonals**
  - Minus the conductances between nodes
  - Zero for all others
- **Current sources**
- **The voltage source adds a variable**
  - I_1 added to node V_3
  - V_3 is equal to 1V



$$\begin{pmatrix} G_1 & -G_1 & 0 & 0 & 0 \\ -G_1 & (G_1+G_2+G_3) & -G_3 & 0 & 0 \\ 0 & -G_3 & (G_3+G_4+G_5) & -G_5 & 1 \\ 0 & 0 & -G_5 & (G_5+G_6) & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ I_1 \end{pmatrix} = \begin{pmatrix} I_0 \\ 0 \\ 0 \\ 0 \\ 1\text{V} \end{pmatrix}$$

Newton-Raphson Method

Emerging Nanoscaled Integrated Circuits and Systems Labs
The Alexander Kofkin Faculty of Engineering
Bar-Ilan University

Nonlinear Devices

- **Modified nodal analysis works for solving circuits made up of linear elements**
  - Resistors, capacitors, inductors, current/voltage sources
- **However, not all devices are linear**
  - Diodes,
  - BJTs
  - MOSFETs
  - …
- **Nodal analysis attempts to solve equations of the form:** $i\left(v_{dc}\right)+u_{dc}=0$
  - However, when nonlinear elements are part of the circuit, these are nonlinear algebraic systems and so cannot be solved directly.
  - We need an algorithmic approach to find a solution

Solving Nonlinear Equations

- **The Newton-Raphson algorithm (a.k.a. "Newton's Method"):**
  - Makes an initial guess on the solution $v^{(0)}$
  - Linearizes the circuit around the guess $J\left(v^{(0)}\right)=\frac{d}{dv}f\left(v^{(0)}\right)$
  - Solves the resulting system of linear equations $v^{(k)}$
  - Re-linearizes the circuit around the new point $J\left(v^{(k)}\right)=\frac{d}{dv}f\left(v^{(k)}\right)$
  - Repeats until the process converges
- **Formally, Newton's method solves:**
  - By repeatedly solving $f\left(\hat{v}\right)=0$

or $J\left(v^{(k)}\right)\left(v^{(k+1)}-v^{(k)}\right)=-f\left(v^{(k)}\right)$

where $v^{(k+1)}=v^{(k)}-J^{-1}\left(v^{(k)}\right)f\left(v^{(k)}\right)$

Step 0: Initialize
set $k \leftarrow 0$
choose $v^{(0)}$

Step 1: Linearize about $v^{(k)}$

$J\left(v\right)=\partial f\left(v^{(k)}\right)/\partial v$

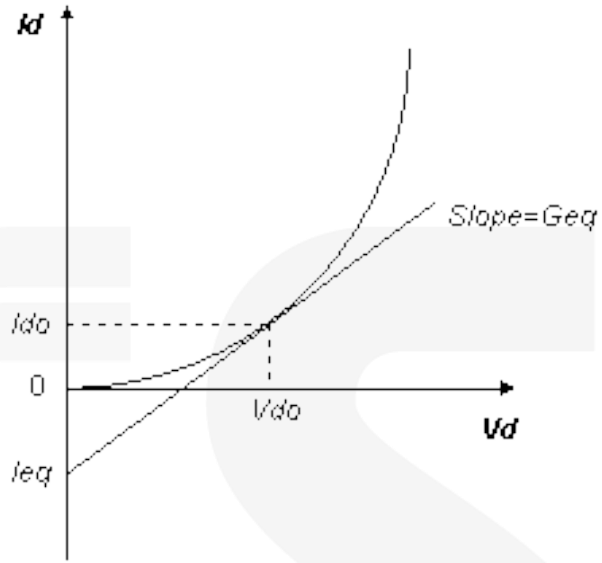
where J_f is the Jacobian of f

Step 2: Solve the linearized system

$v^{(k+1)} \leftarrow v^{(k)}-J_f^{-1}\left(v^{(k)}\right)f\left(v^{(k)}\right)$

Step 3: Iterate
set $k \leftarrow k+1$
if not converged, go to Step 1

Newton-Raphson Method



- **First thing is we need to linearize our non-linear elements**
  - For example, a diode.
  - Linearizing the diode at the operating point gives us: $I_{DO} = G_{eq} \cdot V_{DO} + I_{eq}$
  - So, we replace the diode with this expression (equivalent to an I_{eq} current source in parallel to an R_{eq} resistor)
- **But what is the operating point?**
  - It looks like a chicken and egg question…
  - We *need to know* the operating point (I_{DO} and V_{DO}) to find G_{eq} and I_{eq}.
  - But we are *looking for the operating point*, aren't we?
  - Exactly, so we'll guess, solve and check if we were right.
  - If not, we'll use the solution as our guess, re-linearize, and solve again.

((convergence))

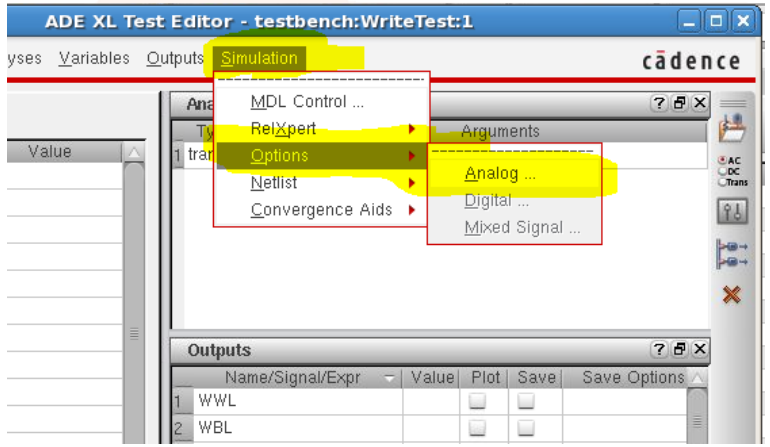
Convergence

- **When do we stop?**
  - We'll never get the *exact* answer. So, we stop based on *convergence criteria.*
- **The Residue Criterion**
  - The first criterion is that KCL should be satisfied to a given degree $\left| f_n\left(v^{(k)}\right)\right| < \varepsilon_f$
  - In practice, a relative criterion is used: 1 $\sum I(node_i) < \texttt{reltol} \cdot |I_{\max}| + \texttt{iabstol}$
  - `reltol` is by default 0.001
  - `iabstol` is by default 1pA
- **The Update Criterion**
  - The second criterion is that the difference in error between the last two iterations is small: $\left|v_n^{(k)} - v_n^{(k-1)}\right| < \varepsilon_x$
  - In practice, a relative criterion is used: 2 $\left|v_n^{(k)} - v_n^{(k-1)}\right| < \texttt{reltol} \cdot \max\left(v_n^{(k)}, v_n^{(k-1)}\right) + \texttt{vabstol}$
  - `vabstol` is by default 1μV

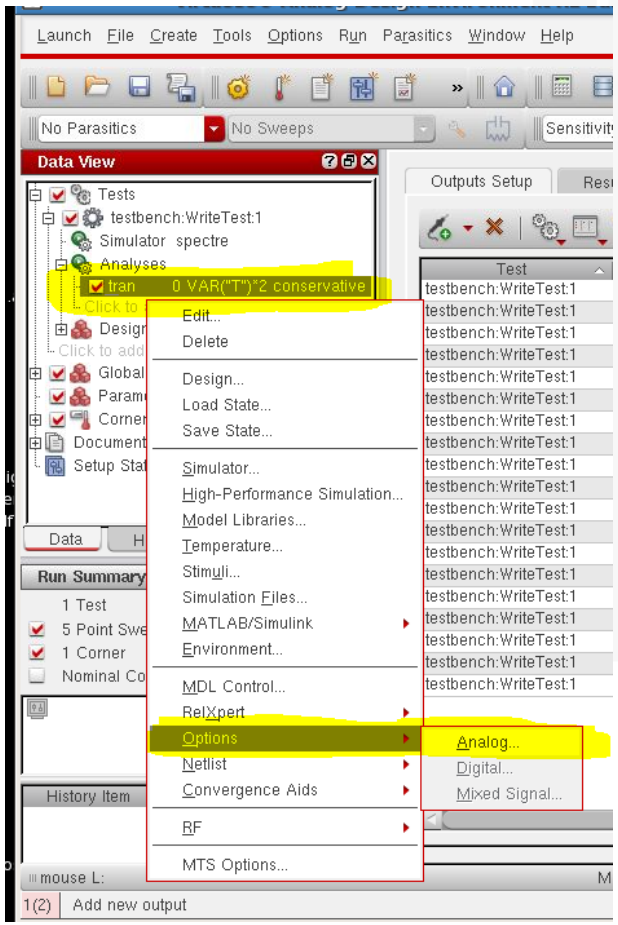
Both Criterions are used for Criteria convergence.

Changing Convergence Criteria

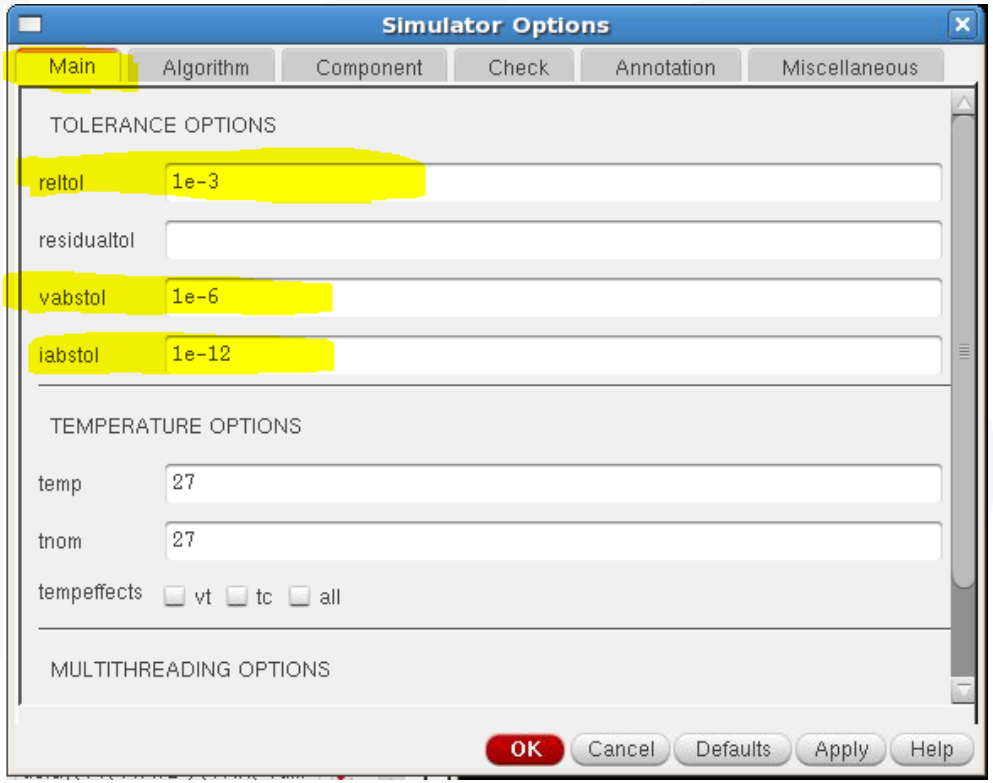
- **Simulation → Options → Analog**



Opening from ADE Test Editor (L or XL)



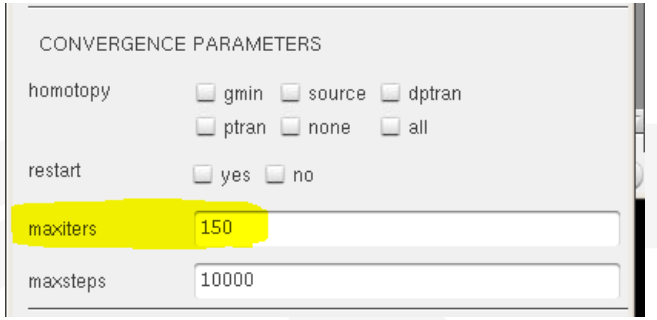
Opening from ADE-XL Data View



Simulation→Options→Analog (Main Tab)

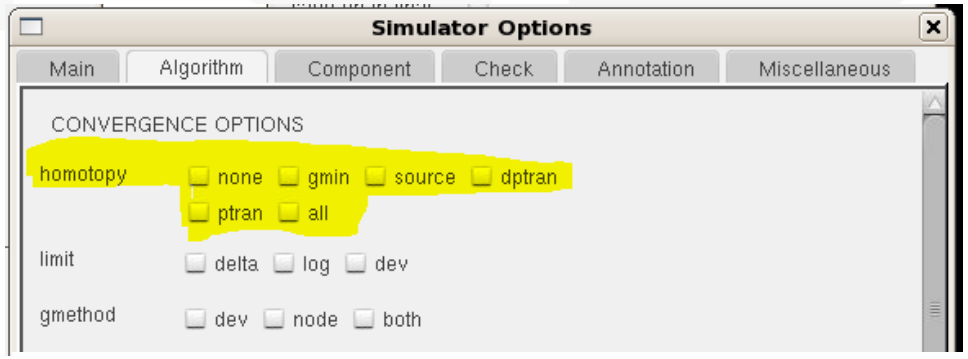
What if the circuit fails to converge?

- **Spectre performs `maxiters` iterations (default 150).**
  - This usually only happens due to bad models or non-physical elements (non-continuities in the transfer curves).



DC Analysis → Options

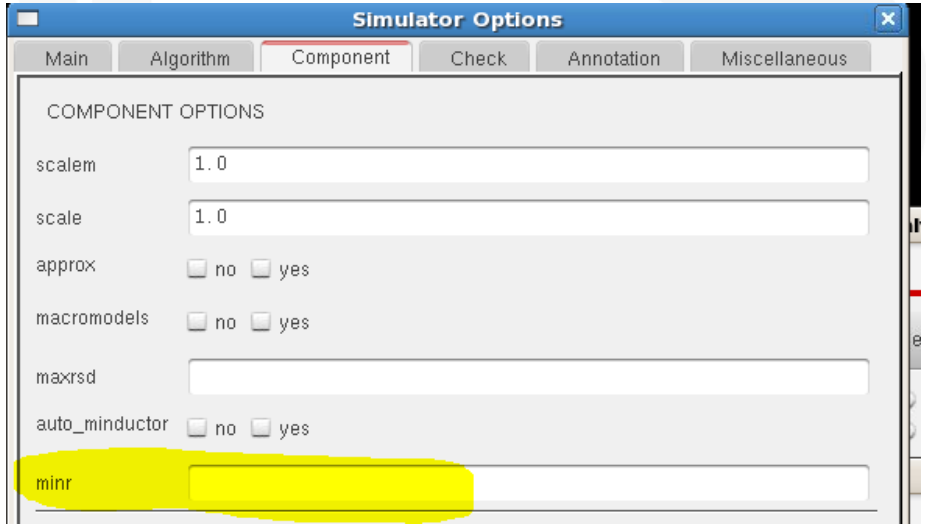
- **If convergence hasn't been met, it tries alternative methods to converge.**
  - These are known as *homotopies*.
  - There are 5 homotopy algorithms: `gmin`, `source`, `dptran`, `ptran`, `none`
  - The default is `all`, which tries each algorithm one after the other.
  - If you see that your solution is converging with a certain homotopy, you can select it without going through all of the options, and thus reduce runtime.



Simulation → Options → Analog (Algorithm Tab)

Convergence aids: `minr`

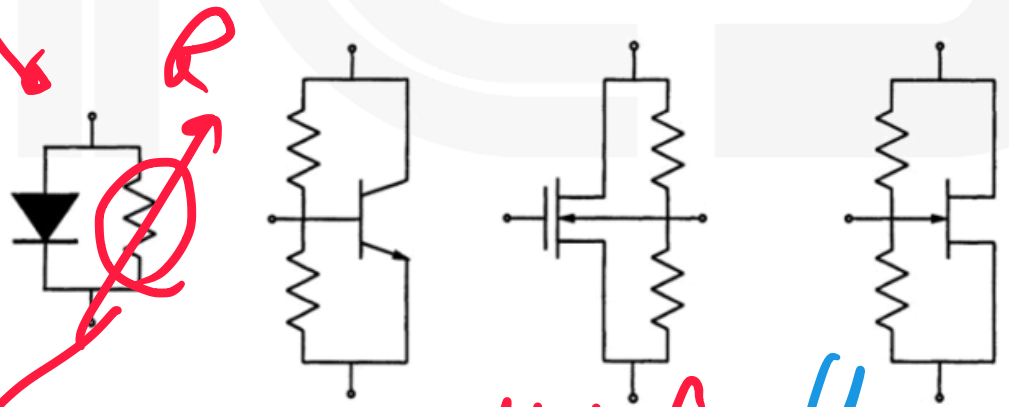
- **Really small resistors cause terrible convergence problems.**
  - A 1µV drop over a 1nΩ resistor results in 1kA of current.
  - This is easily due to a "wrong guess" in the convergence process.
- **The `minr` parameter:**
  - All resistors with R<`minr` will be converted to `minr`.
  - The default is 1mΩ and shouldn't be reduced.
- **In addition:**
  - You also shouldn't extract really small resistors when extracting parasitics…
  - If you do, then increase `iabstol`.



Simulation→Options→Analog (Component Tab)

Convergence aids: gmin

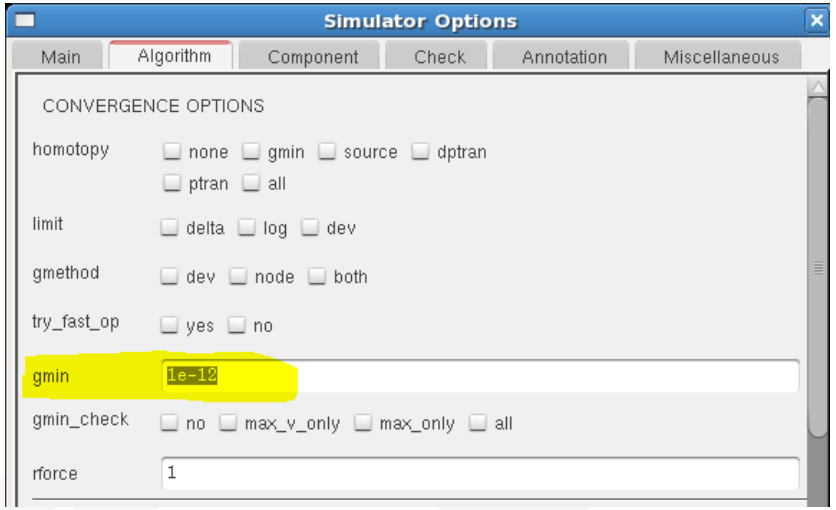
- **Circuits with a *non-isolated solution* have a singular Jacobian**
  - For example, if there is a floating component, there are infinite solutions.
  - Another example is a CMOS inverter with VIN=VDD/2.
    If the FETs have infinite output impedance in saturation,
    the output has a range of voltages that satisfy KCL
- **Therefore, Spectre automatically adds big resistors to ground between potentially floating nets.**
  - This is done for all nonlinear components.
  - The size of the resistors is `1/gmin`.
- **Pay attention – for MOSFETs, a resistor is added between the source and drain!**



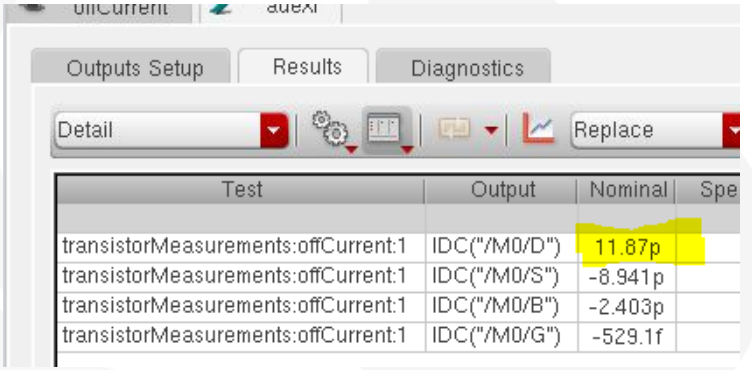
The danger of gmin

- **The gmin resistors are very big, so they will not affect the calculation.**
  - By default, gmin=10^{-12} (R=1TΩ).
- **However, that may not always scale…**
  - For example, for a 1V supply, this results in a "fake" current of 1pA through a closed transistor.
  - Therefore, when dealing with small currents, gmin should be reduced substantially.
- **Example:**
  - A cutoff (V_{GS}=0) transistor I_D=11.7pA with the default gmin.
  - Changing gmin=10^{-18} shows only 9.8pA.
  - That is a 20% increase in leakage that is non-existent!!!

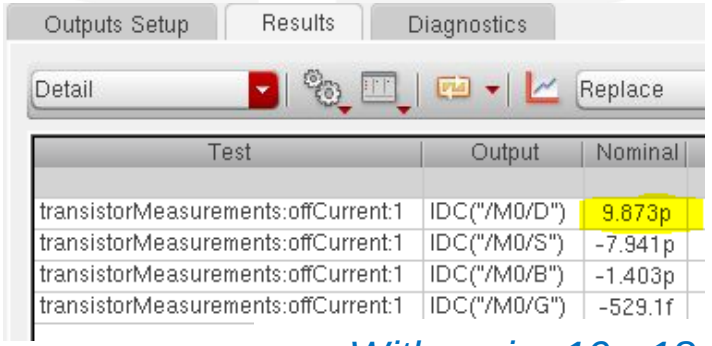
This is not negligible for leakage power analysis, Recommend above gmin for analysis of leakage. However, should make it to a small



Simulation→Options→Analog (Algorithm Tab)



With gmin=10e-12 (default)



With gmin=10e-18

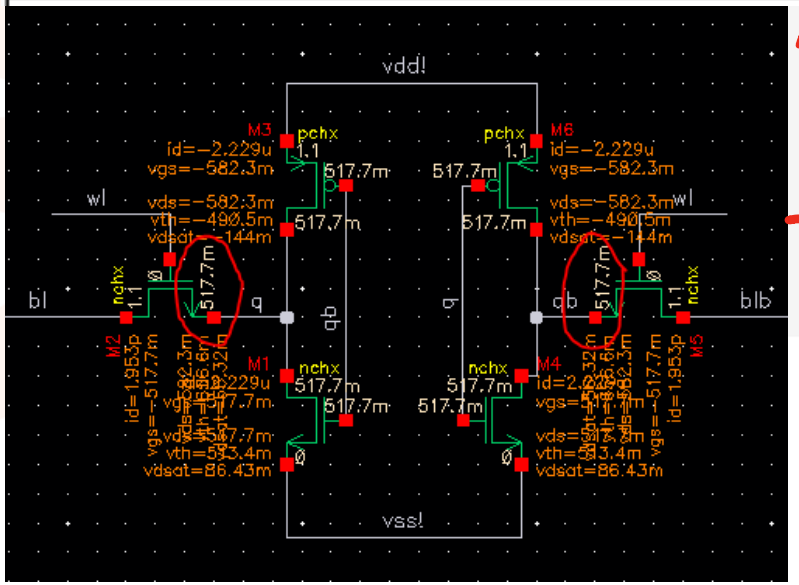
Convergence aids: nodeset

- **Providing Newton's Method with an initial guess is beneficial:**
  - If the guess is close enough to the real DC point, convergence will be faster and will most likely succeed.
- **To bias a multi-stable circuit (e.g., latch) towards a particular solution, an initial guess is provided with the `nodeset` option.**
  - Node Sets connect a DC source with a 1Ω resistor to a defined node.
  - Node Sets are only used during the first convergence iteration and then released.
  - Continuation methods (i.e., DC Sweep) use the previous solution as the initial guess for the next simulation.
  - The OP of the circuit can be saved to a file (`write` command) to load as a `nodeset` for a future simulation (with the `readns` command).

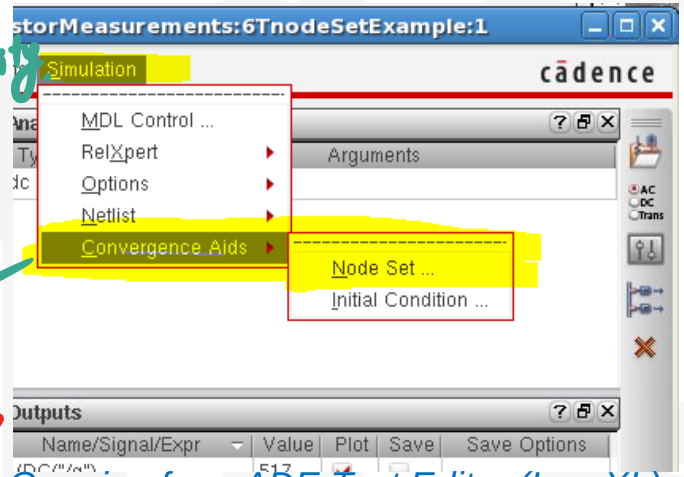
Node Set: 6T Example

- Without setting Node Sets, the 6T settles at its metastable state:

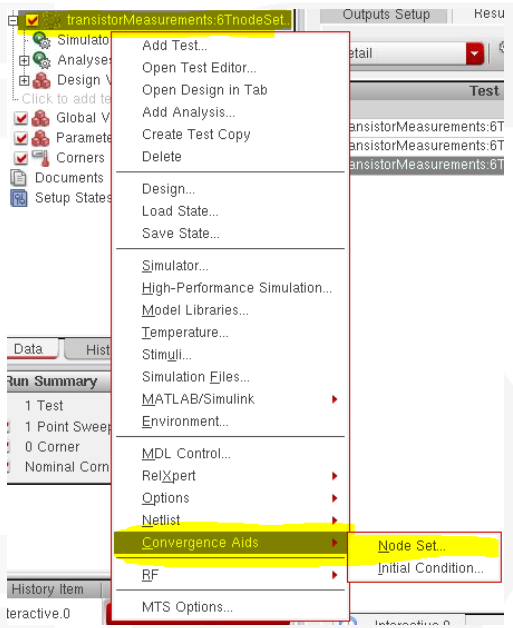
Test	Output	Nominal
transistorMeasurements:6TnodeSetExample:1	VDC("/qb")	517.7m
transistorMeasurements:6TnodeSetExample:1	VDC("/q")	517.7m
transistorMeasurements:6TnodeSetExample:1	none	



- Add `nodeset` statements to `q` and `qb` to bias to one solution



Opening from ADE Test Editor (L or XL)



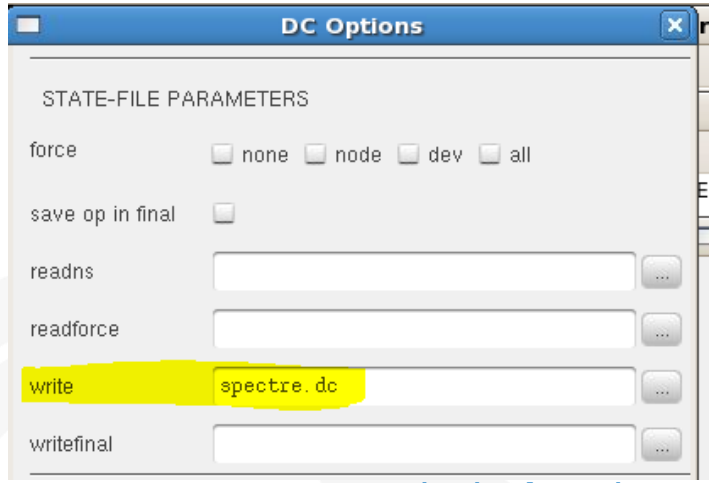
Opening from ADE-XL Data View

- Bitcell settles as expected:

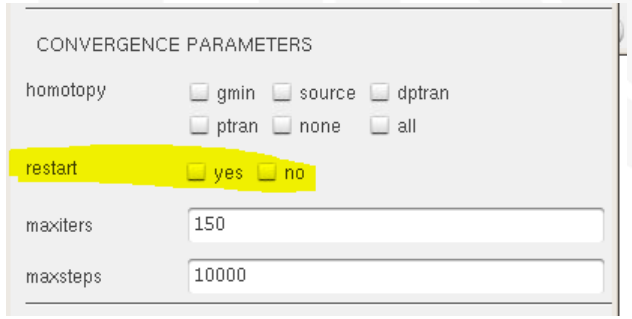
Test	Output	Nominal
transistorMeasurements:6TnodeSetExample:1	VDC("/q")	1.1
transistorMeasurements:6TnodeSetExample:1	VDC("/qb")	139.1n

Saving and loading Node Sets

- **Two options for writing a `nodeset` file:**
  - `write`: File will include state after initial DC solution
  - `writefinal`: will include state after final DC sweep
- **To load a saved `nodeset` as the initial guess of your simulation, use the `readns` option.**
  - Note that `readforce` and `force` are slightly different (generally, don't use them).
- **Another option is `restart`:**
  - `restart=yes` (default) means the DC OP is calculated from the beginning if any change has occurred.
  - Usually, there's no reason to change this.



DC Analysis→Options



DC Analysis→Options

```
q           1.09999976167207
qb          1.39126719681416e-07
vdd!        1.1
vss!        0
I0.M1:int_d     1.09999976160319
I0.M1:int_s     5.07922470231393e-11
I0.M2:int_d     1.09999999998177
I0.M2:int_s     1.09999976165384
I0.M3:int_d     1.09999976189583
I0.M3:int_s     1.09999999975509
I0.M4:int_d     1.38796083077854e-07
I0.M4:int_s     4.0189891047523e-10
I0.M5:int_d     1.09999999961391
I0.M5:int_s     1.3949457641895e-07
I0.M6:int_d     1.39177502299711e-07
I0.M6:int_s     1.09999999995642
I1.gnd_supply:p -4.57967001638684e-17
I1.vdd_supply:p -7.23916508216806e-11
```

DCop file (spectre.dc) for the 6T run

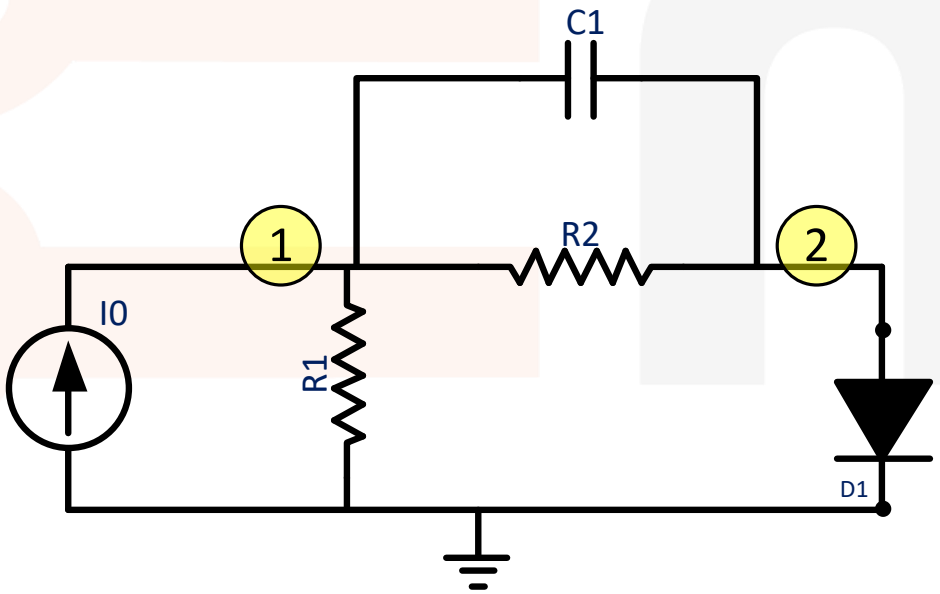
AC Analysis

Start with a DC Operating Point

- **An AC Analysis needs a bias point**
  - Equivalent to a DC Operating Point.
  - In other words, the bias point is found by DC OP convergence
- **However, due to phase shifts, the solution may be different**
  - Usually due to setting a DC Voltage on sources.
- **One thing to do to eliminate some of these differences is**
  ***Do Not set the DC Voltage to 0 on Vpulse/VAC sources*.**

Comprise the AC Matrix

- **Equivalent to the DC Matrix, with caps and inductors added as admittances.**
  - jwC for a cap
  - $1/jwL$ for an inductor



$$\begin{pmatrix} G_1+G_2+j\omega C_1 & -(G_2+j\omega C_1) \\ -(G_2+j\omega C_1) & G_D+G_2+j\omega C_1 \end{pmatrix} \begin{pmatrix} V1 \\ V2 \end{pmatrix} = \begin{pmatrix} I0 \\ -I_D \end{pmatrix}$$

Transient Analysis

Transient Analysis

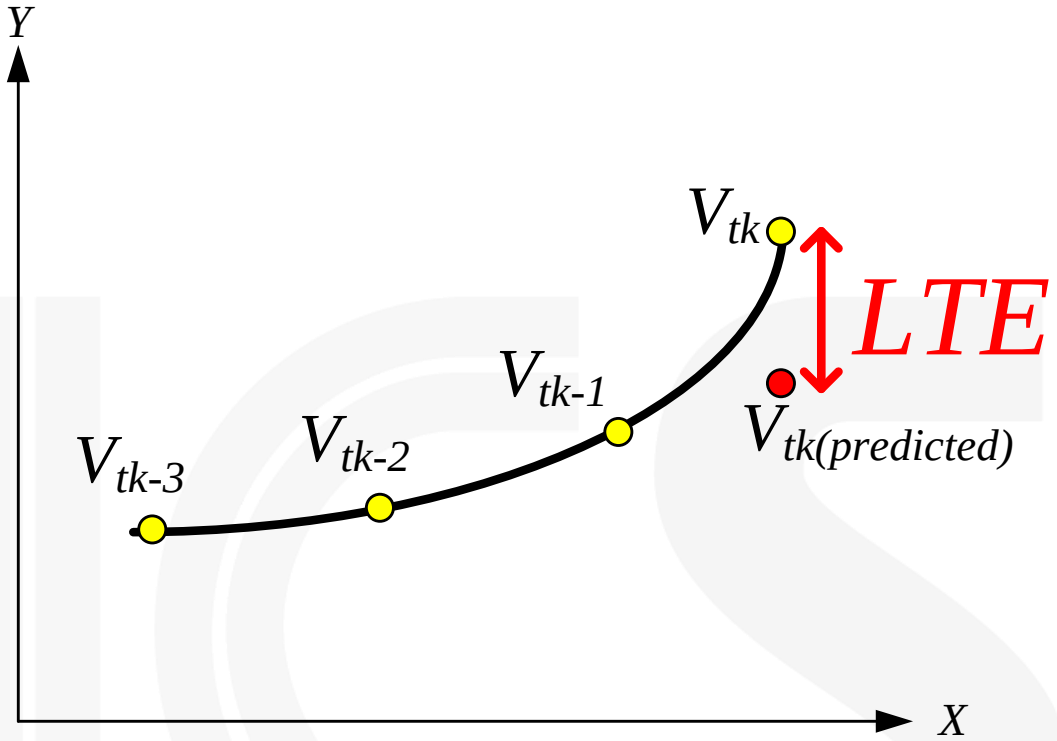
- **Transient analysis generates a system of non-linear ordinary differential equations.**
- **There is no known method to directly solve these equations.**
- **Instead discretize time:**
  - e.g., using the Euler formulation $\frac{dq(t_i)}{dt} \approx \frac{q(t_i) - q(t_{i-1})}{t_i - t_{i-1}}$
  - This converts the problem into a system of non-linear algebraic equations.
- **In other words:**
  1. First a DC OP is calculated (a.k.a. the "initial transient solution").
  2. Then caps and inductors are added and their currents/voltages are linearized according to their integral values.
     - Non-linear elements are linearized.
     - Newton-Raphson is used to find the DC OP for each time step throughout the transient.

Transient Analysis

- **Timesteps are important to tradeoff accuracy vs. runtime.**
- **After each timestep, the simulator:**
  - Decides when the next timestep should be.
  - *Predicts* the values at the next timestep.
  - *Calculates* the DC OP at the next timestep.
  - If the error between the prediction and calculation is bigger than the Local Truncation Error (LTE), a closer (sooner) timestep is taken.
  - The `maxstep` parameter sets the maximum allowed size of a timestep.
- **Breakpoints are set where pre-known discontinuities are simulated**
  - Such as at VPULSE or VPWL points.
- **Calculations around breakpoints are handled separately to ensure convergence.**

Local Truncation Error

- **LTE is the error between the calculated and predicted values at the next timestep.**
- **If the LTE is:**
  - Smaller than the convergence criteria, the timestep is kept.
  - Otherwise, a closer timestep is chosen.



LTE *Convergence criteria*

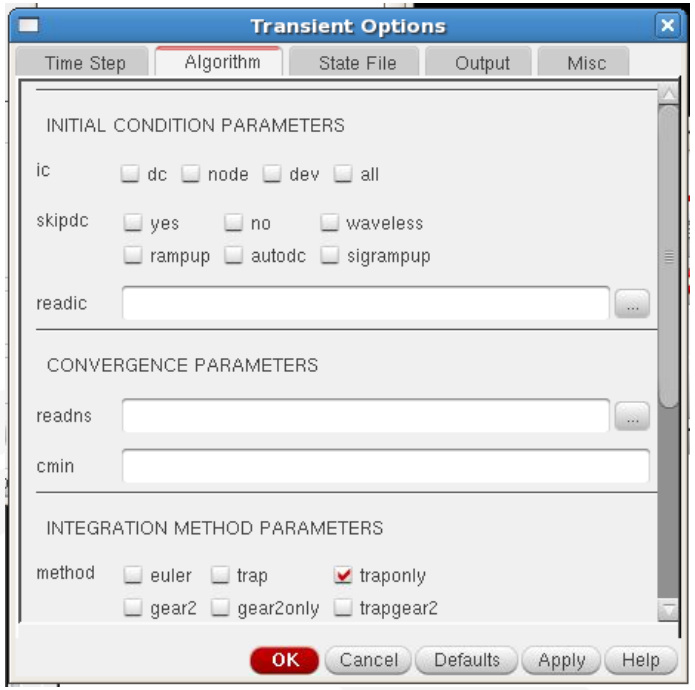
$$\left|V_{tk} - V_{tk(\text{predicted})}\right| < \left|\texttt{lteratio}\cdot\left(\texttt{reltol}\cdot V_{tk(\max)} + \texttt{vabstol}\right)\right|$$

- Calculated value at current node
- Predicted value at current node
- Normalization factor
- Relative tolerance
- Maximum voltage (`relref`)
- Absolute tolerance

`relref` sets the maximum voltage:

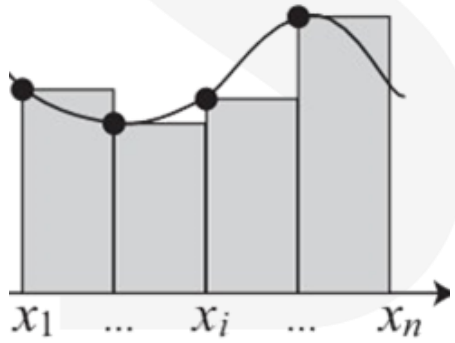
- `pointlocal` – the largest value at this node during this iteration.
- `local` – the largest value at this node so far.
- `global` – the larges value at any node so far.

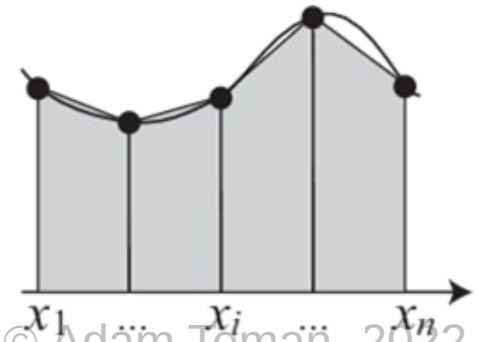
Transient Analysis →Options
(Algorithm Step Tab)



Integration method

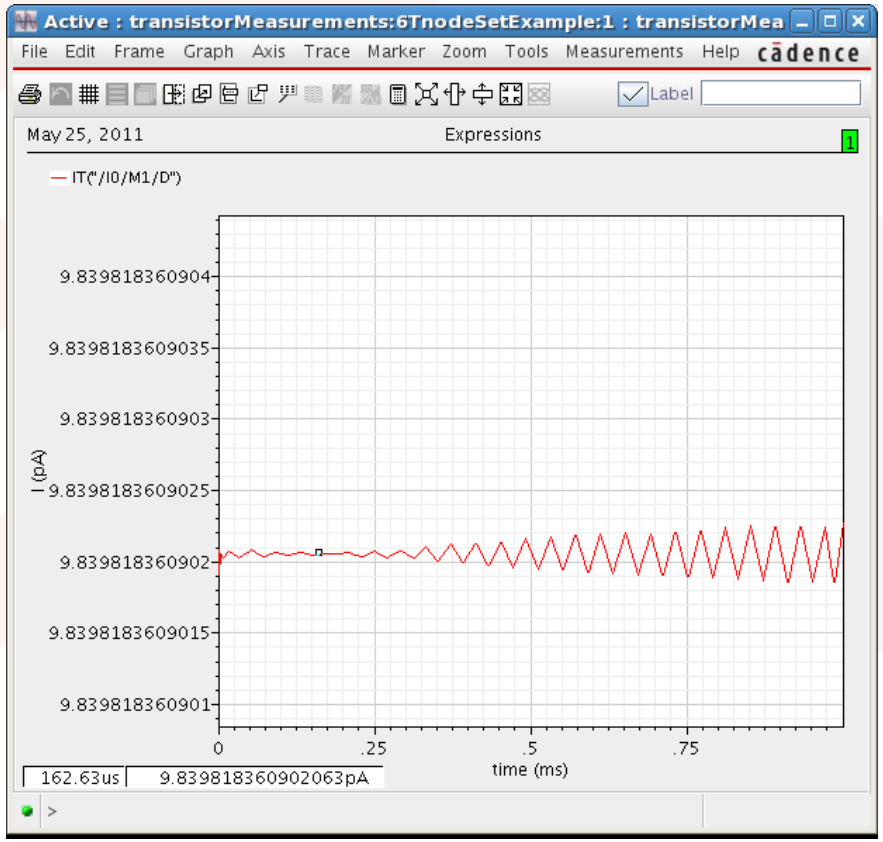
- **Caps** and **Inductors** present an **integral relationship**:
  - $L*I=\int V dt \quad C*V=\int I dt$
- The `method` parameter sets the **integration scheme**.
  - `euler` (first order gear):
    only good around discontinuities
    $$\frac{d}{dx}v(t_{k+1}) \approx \frac{1}{h}\left[v(t_{k+1})-v(t_k)\right]$$
  - `trap` (trapezoidal):
    good but can cause oscillations
    $$\frac{d}{dx}v(t_{k+1}) \approx \frac{2}{h}\left[v(t_{k+1})-v(t_k)\right]-\frac{d}{dx}v(t_k)$$
  - `gear2` (second order gear):
    good curve fitting but can damp oscillations.
    $$\frac{d}{dx}v(t_{k+1}) \approx \frac{3}{2h}v(t_{k+1})-\frac{2}{h}v(t_k)+\frac{1}{2h}v(t_{k-1})$$

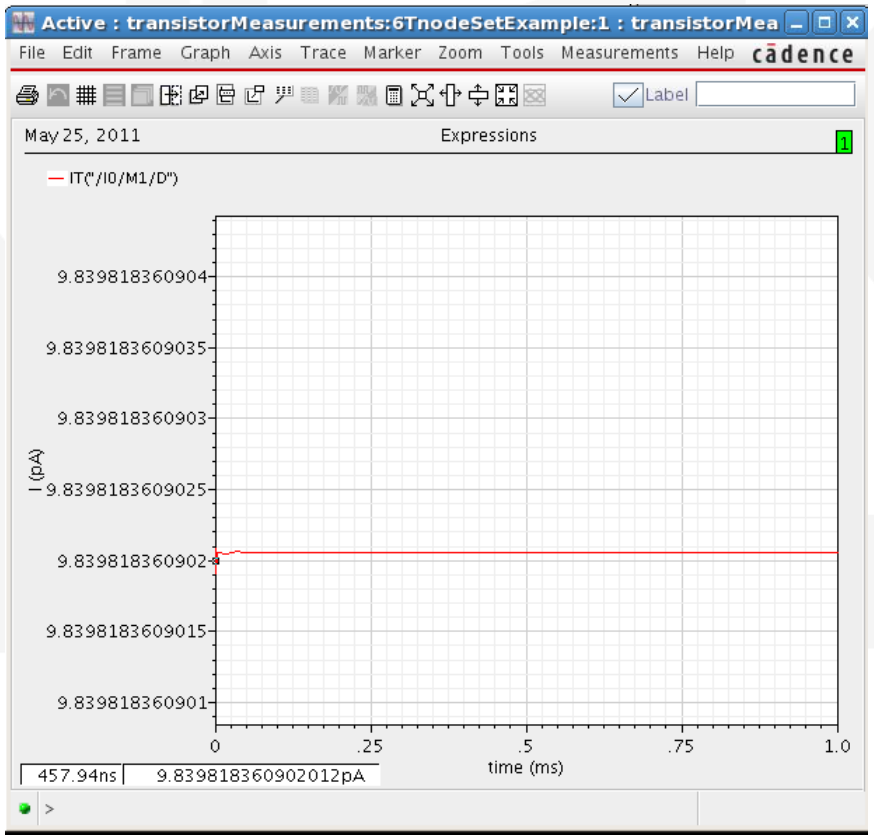




- **If you have "fake" oscillations, it's due to using trapezoidal integration.**

Fake Oscillations

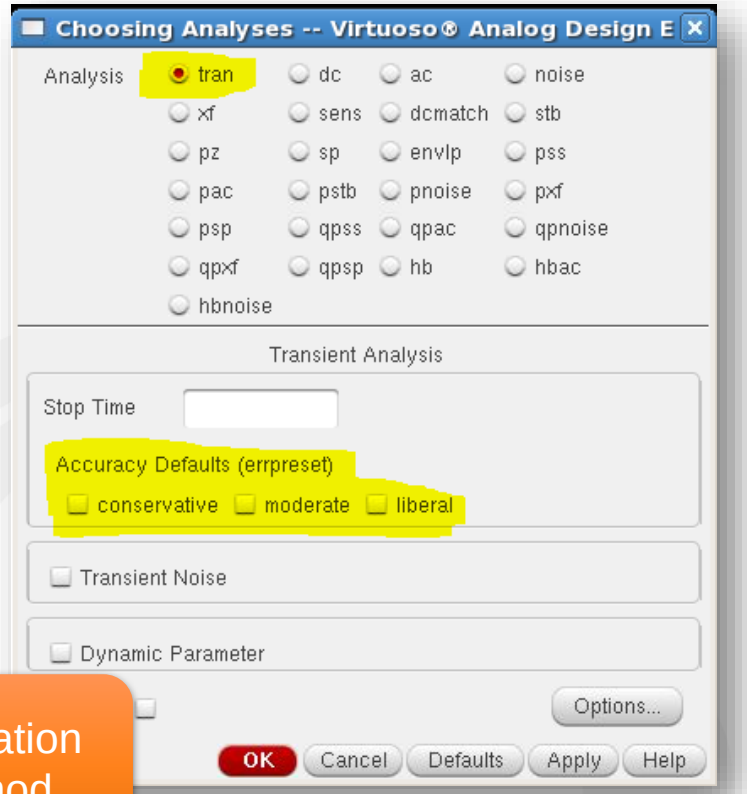




errpreset

★ Trade off between Accuracy & Speed

- **Most of the important parameters for transient analysis are preset according to the `errpreset` parameter:**
  - `liberal` – Fast simulation but can lose accuracy.
  - `moderate` – moderate simulation and accuracy.
  - `conservative` – high accuracy but slow simulation.



Transient Analysis → Options

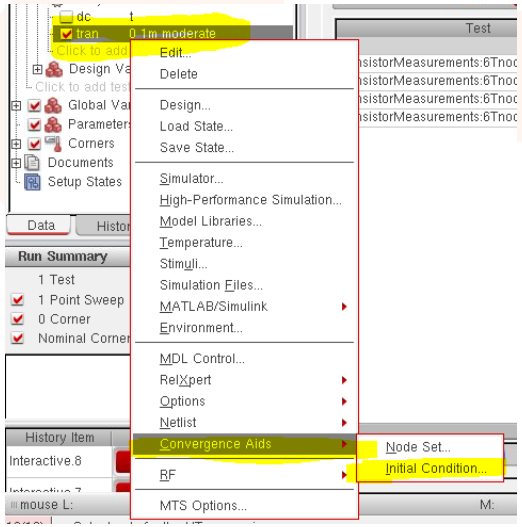
Maximum timestep | Relative error tolerance | LTE normalization | Maximum voltage reference | Integration Method

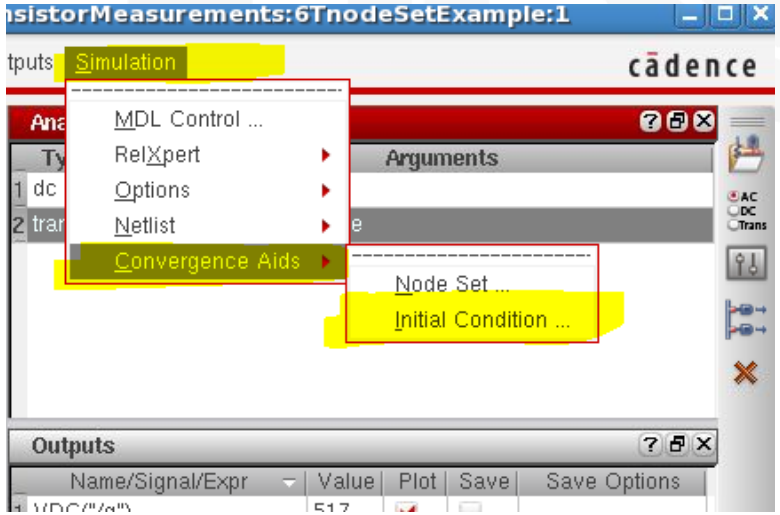
errpreset	maxstep	reltol	lteratio	relref	method
Liberal	$\frac{T_{stop}-T_{start}}{50}$	10	3.5	sigglobal	gear2
Moderate	$\frac{T_{stop}-T_{start}}{50}$	1	3.5	sigglobal	traponly
Conservative	$\frac{T_{stop}-T_{start}}{100}$	0.1	10	alllocal	gear2only

Convergence Aids: Initial Conditions

- **Initial Conditions are the same as Node Sets, but they are not removed after the first DC OP iteration.**
  - Node sets are to *assist in convergence*.
  - Initial Conditions are to *set a value* at a node at the beginning of the transient.
  - DC OP and DCSweep analyses ignore Initial Conditions.
- **Initial Conditions can be set on the test (`.ic`) or on caps/inds.**

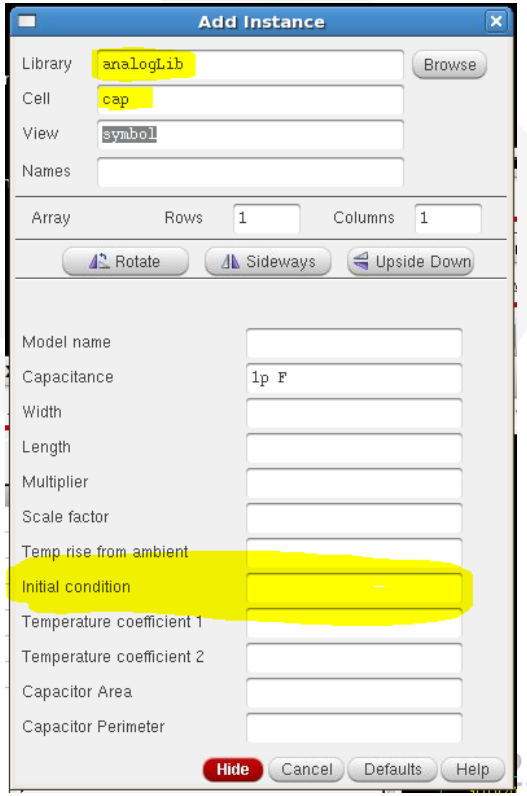
Opening from ADE Test Editor (L or XL)





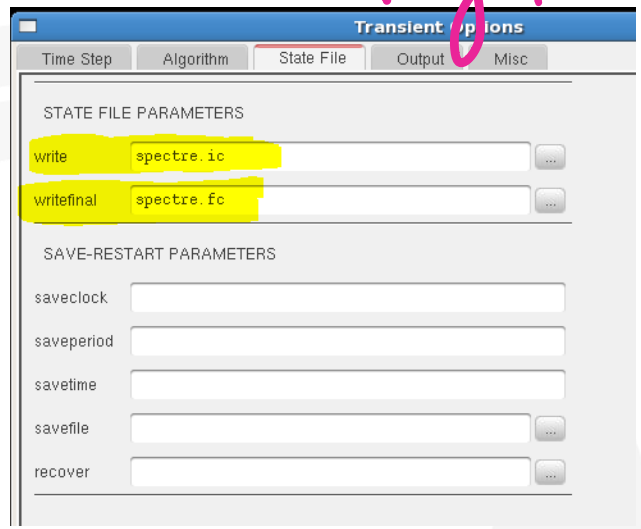
Opening from ADE-XL Data View

On capacitor properties

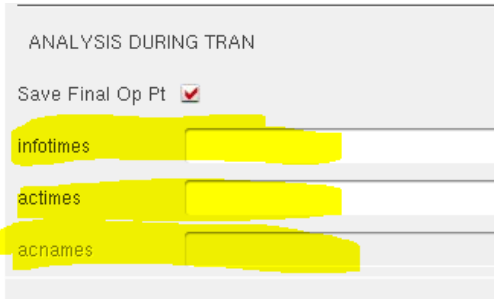


Saving Time Points

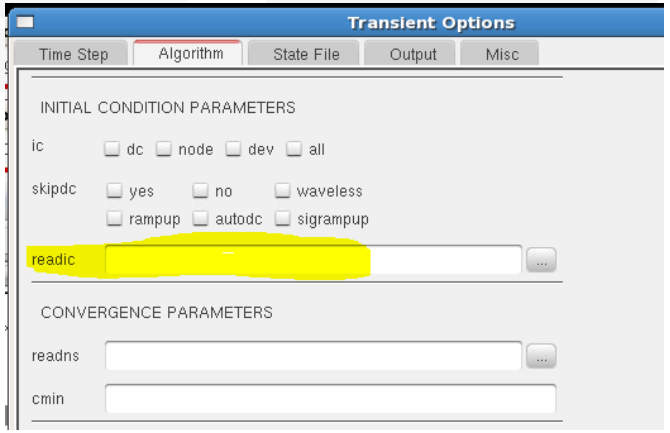
- **Initial conditions can also be written to a file and loaded to a simulation:**
  - `write`: file that ic of *current simulation* are written to.
  - `writefinal`: file that *final state of simulation* is written to.
  - `readic`: file to *read initial conditions* from.
- **Other saving options:**
  - `saveclock`: periodically save in real time minutes.
  - `saveperiod`: periodically save in simulation time.
  - `savetime`: save at specific simulation timepoints.
  - `savefile`: name of the save file.
  - `recover`: start simulation from this file.
  - `infotimes`: times to save DC OP.
  - `actimes`: times to run AC simulation.



Tran Analysis→Options (State File Tab)



Tran Analysis→Options (Output Tab)

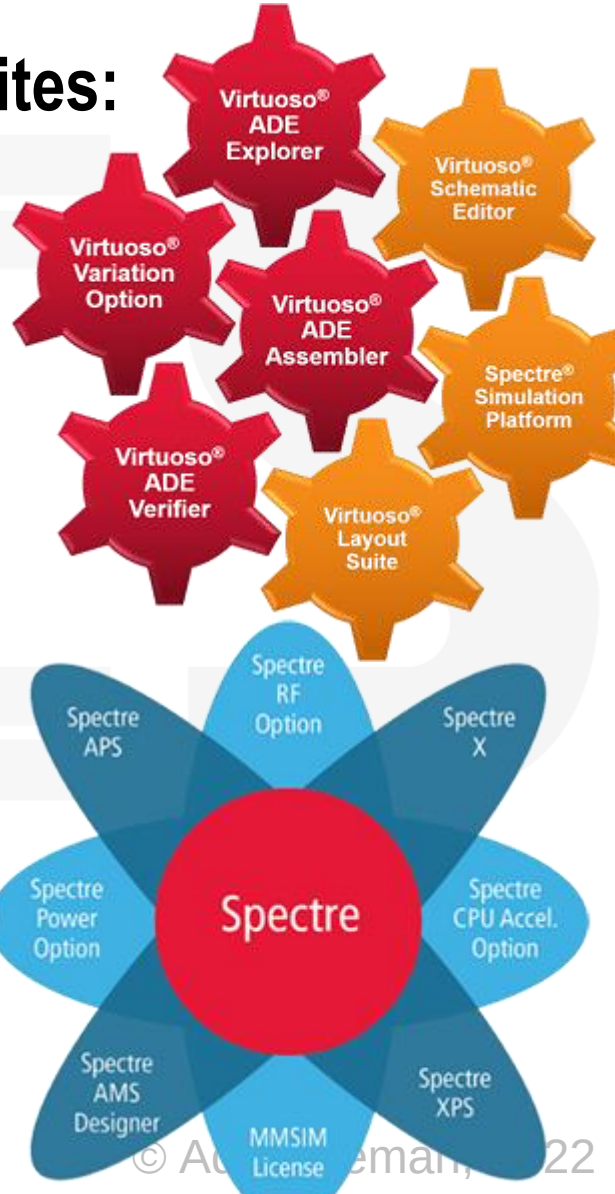


Tran Analysis→Options (Algorithm Tab)

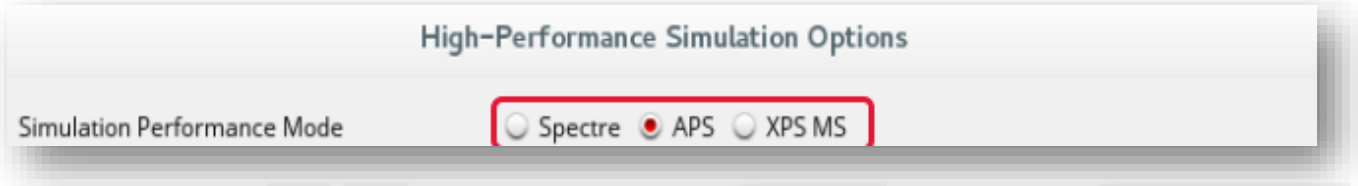
Other Stuff

Tools and Versions

- **The Cadence Custom IC Design includes the following tool suites:**
  - Virtuoso (IC 6.1.8) including schematic and layout editors, Analog Design Environment (ADE), VIVA, etc.
  - Spectre (MMSIM) including APS, Spectre X, XPS, Spectre AMS Designer, Spectre FX, Spectre X-RF, etc.
- **Different simulation options are:**
  - Spectre SPICE engine
  - Spectre Accelerated Parallel Simulator (APS) for high-precision and scalable multi-core simulation
  - Spectre X for high-performance, high-capacity simulation
  - Spectre Extensive Partitioning Simulator (XPS), an advanced FastSPICE engine
  - Spectre AMS Designer for mixing these with Xcelium



Spectre APS

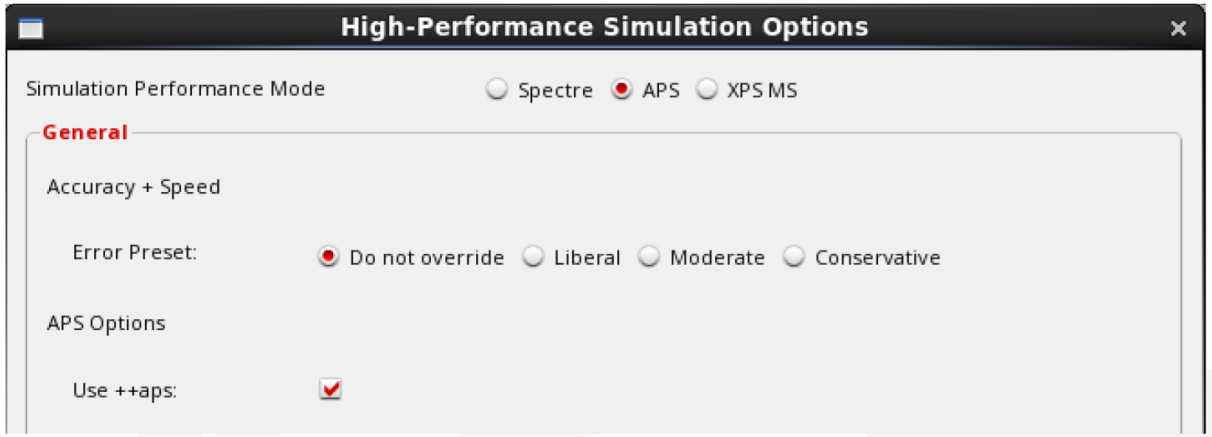


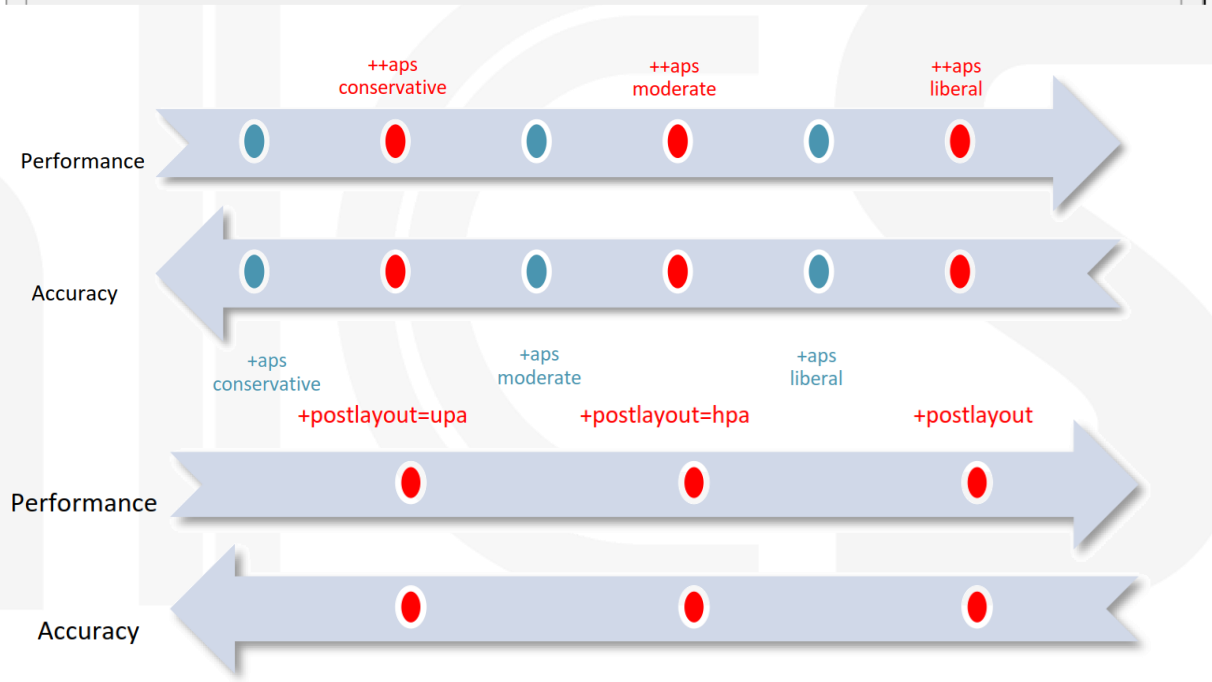
- **Full baseline Spectre accuracy**
  - No change to core Spectre timestep algorithms
- **Same use model as baseline Spectre technology**
  - Just add `+aps` to the command line, or enable the ADE option
- **Significant performance gain on single/multiple cores**
  - 5-100x simulation performance vs. baseline Spectre technology for non-RF
  - 5-20x vs. baseline SpectreRF
- **Much larger simulation capacity**
  - Designs up to 10M transistors, 50M RCs (no reduction)
  - EM/IR simulations with > 500M elements
- **Improved convergence over core Spectre technology**

Unless your design is trivial in size, use the APS option

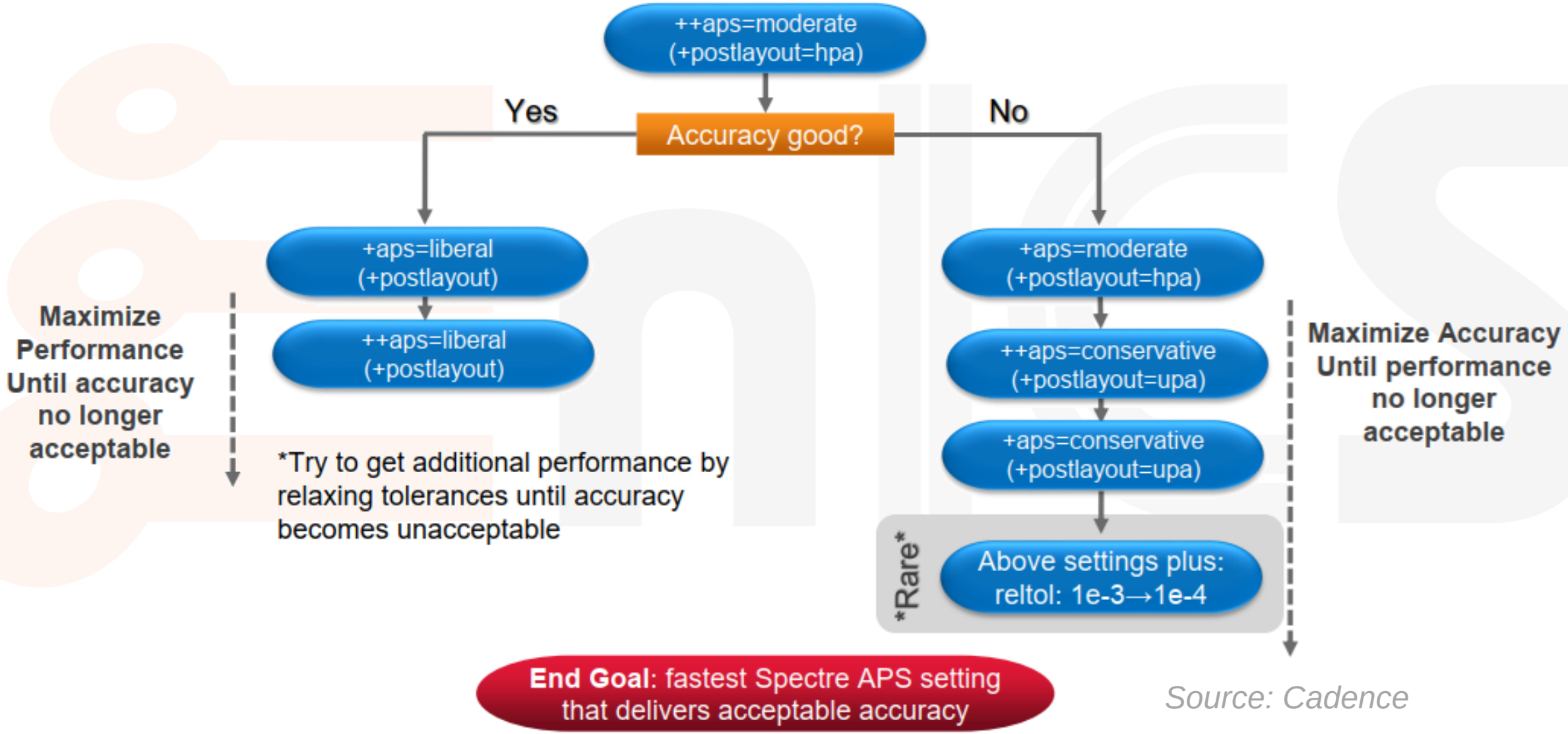
Faster Spectre APS

- **Designed to produce identical simulation results as baseline Spectre.**
  - More accurate: `spectre +aps`
  - Faster: `spectre ++aps`
- **Since postlayout is often hard on convergence and transient simulation:**
  - `spectre ++aps +postlayout`
- **Sometimes lightweight simulation can further speed things up**
  - `spectre ++aps +lite`
- **To manually specify the number of multithreads (default=8)**
  - `spectre +aps +mt=16`





Getting The Most Out Of Spectre APS



Source: Cadence

Spectre X

- **Spectre X is the next generation of Spectre including:**
  - Enhanced simulation performance and capacity with a simple `+preset` use model
  - Highly scalable multi-core simulation with `+mt`, and distributed multi-process simulation with `+xdp`.
- **To run Spectre X with a specific preset:**
  - `spectre +preset=mx input.scs`
  - These presets ignore `errpreset` options
- **For postlayout you can set:**
  - `spectre +preset=mx +postlpreset=cx`
- **Spectre X uses the `+xdp` for distributed simulation on up to 512 cores**
  - Check with your sysadmin what options can work

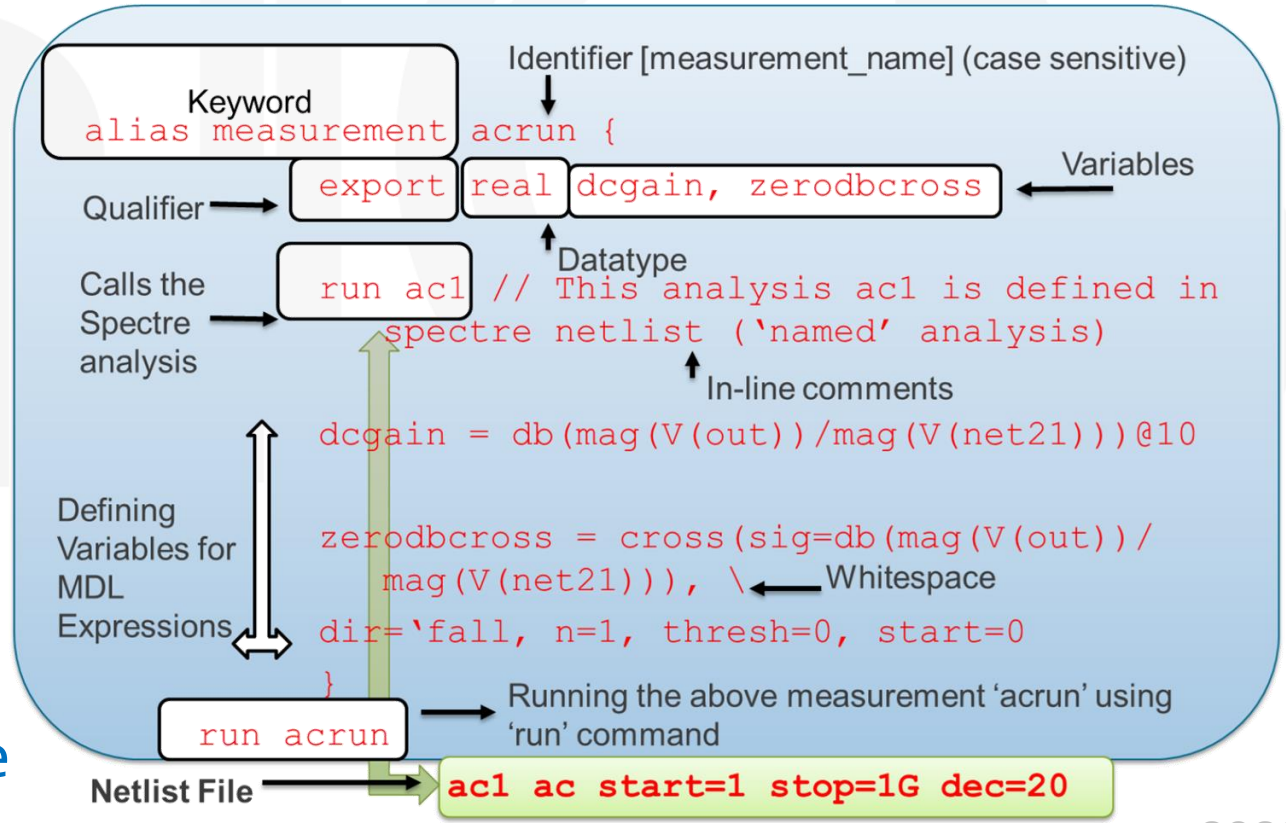
Option Name	When to Use...
cx	when a golden simulation reference is needed
ax	for high-precision analog applications
mx	for most analog applications (default)
lx	for power management and other relaxed analog applications
vx	for custom IC verification

Spectre XPS

- **Spectre XPS is the new advanced FastSpice simulator**
  - `spectre +xps +mt=16 +cktpreset=dram +speed=1`
- **Spectre XPS has `+cktpreset` options for several circuit types**
  - `dram`, `sram`, `sram_pwr`, `pcram`, `flash`
- **There are also many options for reducing parasitics, partitioning, providing more accurate analog simulations, etc.**

Spectre MDL

- **Spectre MDL (Measurement Description Language) is a scripting language that enables you to define measurements and batch process simulations.**
- **Create an mdl control file**
  - "`export`" defines measurements
  - "`run`" tells which analysis to run (from your netlist)
- **Run the `spectremdl` command**
  - `spectremdl -batch myfile.mdl -design input.scs`
- **Postprocess the .raw data**
  - `mdl -b test.mdl -r input.raw -m input.measure`



References

- Andrew Beckett “Using Spectre Simulator Effectively”
- Kenneth S. Kundert, “The Designer’s Guide to Spice and Spectre”, 2003
- Spectre Userguide 20.1