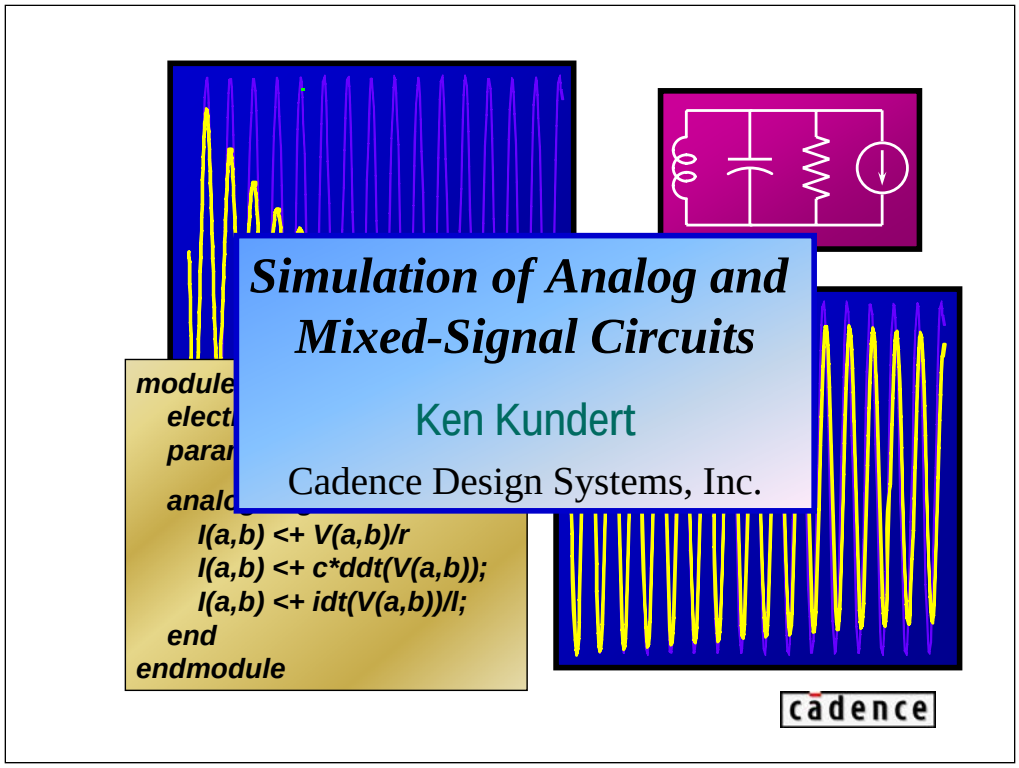
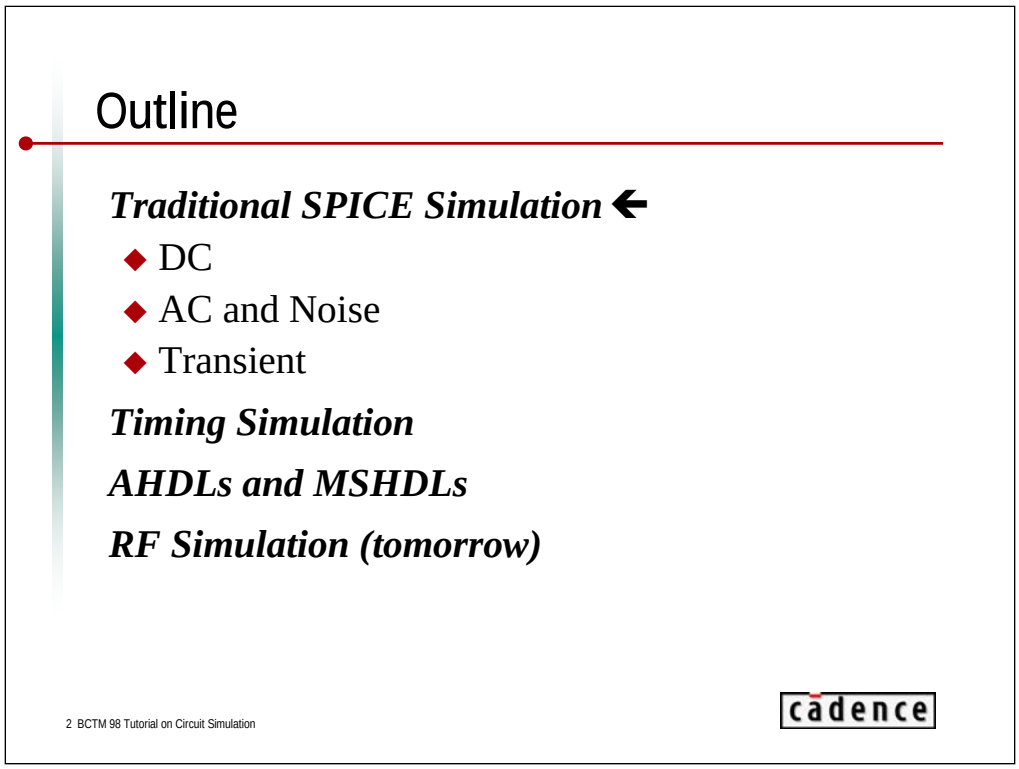


Author

Dr. Ken Kundert is a fellow at Cadence Design Systems and for many years has been the principle architect of the Spectre circuit simulation family. As such, he has lead the development of Spectre, SpectreHDL, and SpectreRF. He also played a key role in the development of Hewlett-Packard's harmonic balance simulator and made substantial contributions to both the Verilog-AMS and VHDL-AMS languages. He has authored two books on circuit simulation, *Steady-State Methods for Simulating Analog and Microwave Circuits* in 1990 and *The Designer's Guide to SPICE and Spectre* in 1995, as well as over a dozen papers published in refereed conferences and journals.

Dr. Kundert received Ph. D., M. Eng., and B. S. degrees in electrical engineering and computer sciences from the University of California, Berkeley in 1989, 1983 and 1979, respectively. He specialized in circuit simulation and analog circuit design.

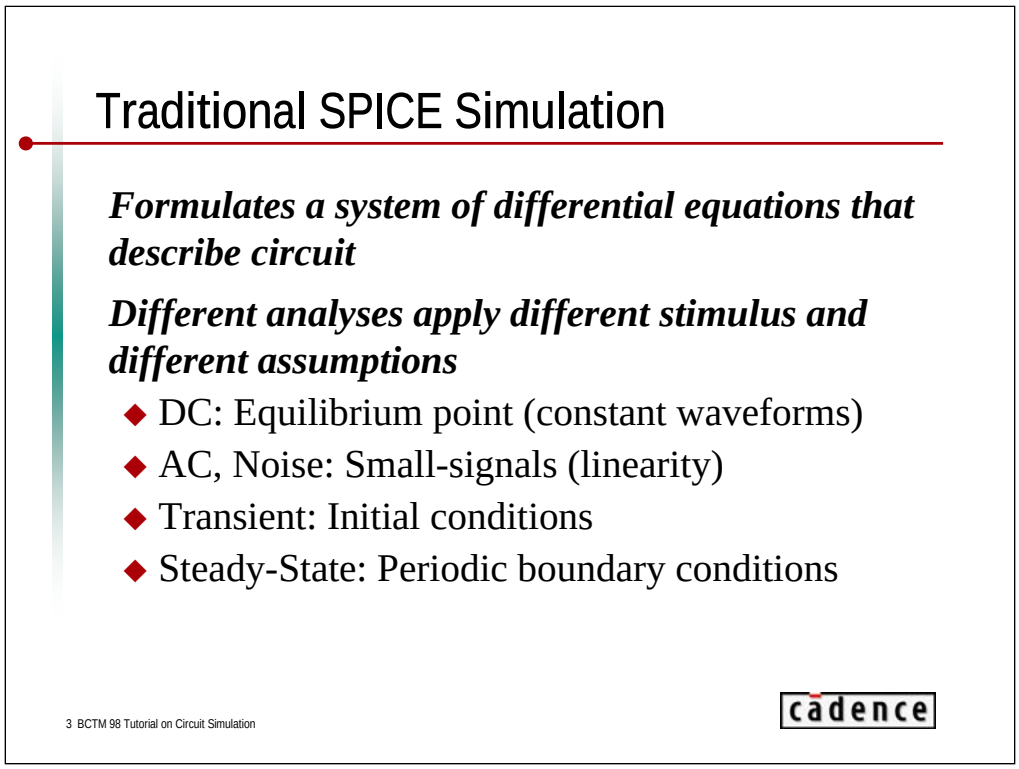


Abstract

This presentation is a tutorial on circuit simulation for analog and mixed-signal circuits. The emphasis is presenting information about the way circuit simulators behave that is useful to the practicing circuit designer. I begin with traditional SPICE simulation and cover DC, AC, transient and Fourier analyses. Timing simulation is introduced and contrasted with circuit simulation. Finally, mixed-signal hardware description languages (MS-HDLs) are presented as a way to improve the productivity of engineers involved in the design of complex mixed-signal circuit and Verilog-AMS is introduced.

Target Audience

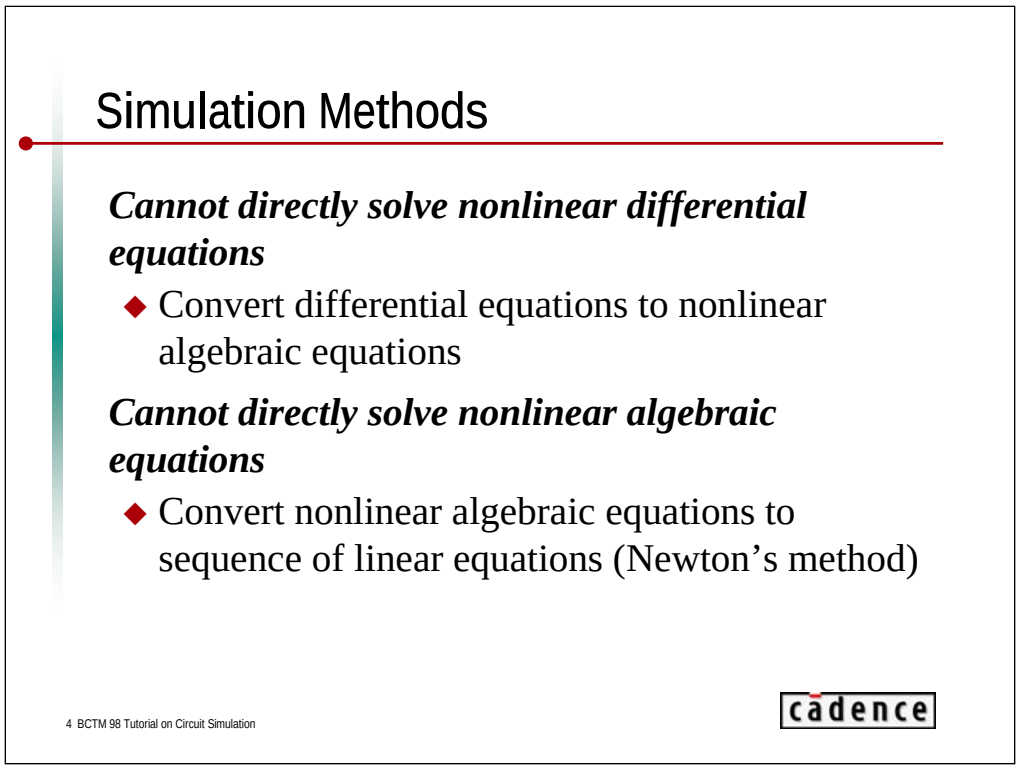
The primary audience is circuit designers and CAD engineers that use or support circuit simulators.



Circuit Simulators

Circuit simulators are programs that from a structural description of a circuit formulate and solve a system of nonlinear differential equations. In particular, the simulator formulates a system of nonlinear first-order differential/algebraic equations. These equations describe the circuit and the simulator solves them in order to predict the circuit response to a specified stimulus. The nonlinear system of equations consists of model equations combined with equations that embody Kirchhoff's laws. The model equations are either hard-code in the simulator as built-in models or are specified by the user in the form of behavioral models.

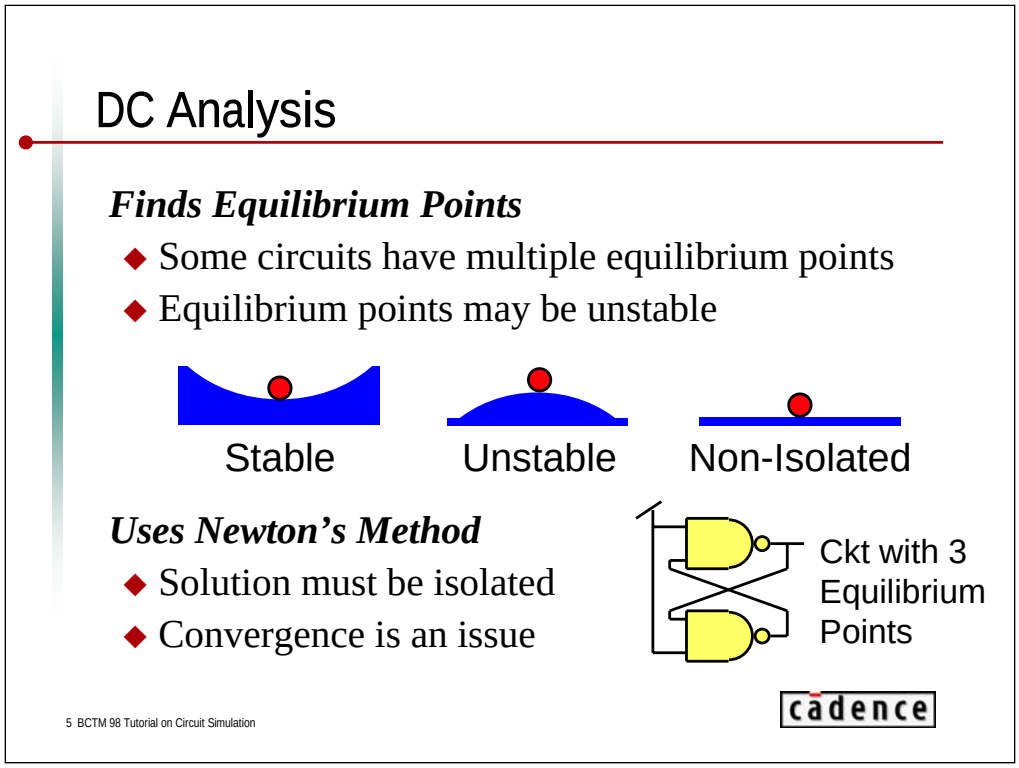
Various analyses apply different assumptions before solving the equations. For example, DC analysis assumes that all waveforms in the circuits are constant valued. Doing so allows the system of differential equations to be simplified to a system of nonlinear algebraic equations that are solved for the equilibrium points of the circuit. AC analysis assumes the stimulus is a small sinusoid applied as a perturbation of the DC equilibrium point and that the stimulus was started long ago so that the circuit has reached its steady-state behavior. Transient analysis assumes the circuit is driven with some arbitrary stimulus and that the circuit starts from a particular initial condition, typically the DC operating point. Finally, a steady-state analysis assumes the circuit is driven with either a periodic or quasiperiodic stimulus and finds the steady-state response. Unlike with AC analysis, the stimulus is not assumed to be small.



Simulation Methods

Once the system of equations that describe the circuit is formulated, it is necessary to solve it. Unfortunately, there is no known approach to solving a nonlinear system of differential equations analytically. Simulators solve these equations numerically. The assumptions in both DC and AC analysis allow the differential equations to be reformulated as a set of algebraic equations. In transient analysis, integration methods are applied that convert the differential equations into difference equations that can be solved by solving a sequence of nonlinear algebraic equations.

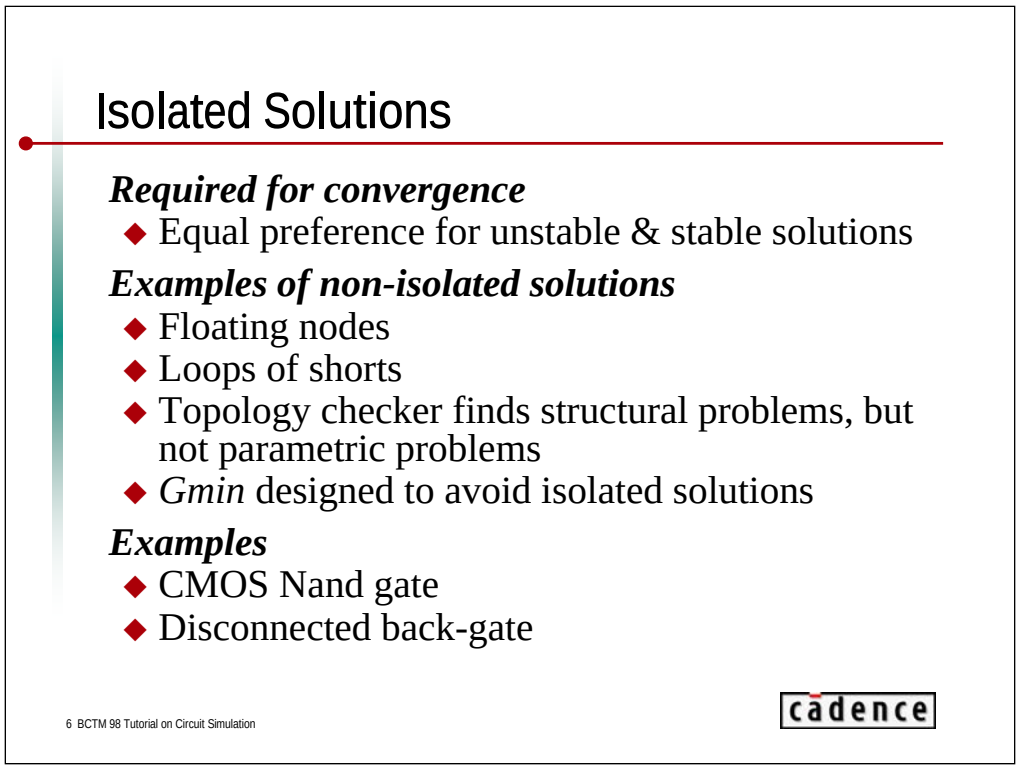
There is also no direct method for solving nonlinear algebraic equations. Instead Newton's method, which is also known as the Newton-Raphson or NR method, is used. It is an iterative procedure for solving nonlinear equations that converts the problem of solving a nonlinear equation into that of solving a sequence of linear equations.



DC Analysis

DC analysis finds solutions to the systems of equations that describe the circuit that are constant valued. It does so by simply discarding time-derivatives (since the solution is constant valued, its time-derivative must be zero). These constant valued solutions are the equilibrium points of the circuit. It is important to realize that a single circuit may have more than one equilibrium point and that there are three types of equilibrium points: stable, unstable, and non-isolated. A stable equilibrium point is such if a circuit is perturbed slightly while sitting at a stable equilibrium point, it will eventually return to it. However, when disturbed, a circuit will not return to an unstable equilibrium point. The latch shown is an example of a circuit with multiple equilibrium points. Two of the equilibrium points are stable (the one associated with the output being high, and the other associated with the output low). The third equilibrium point results with both gates exactly balanced and the output half way between high and low. This equilibrium point is unstable. Any disturbance will cause the latch to rapidly switch and settle into one of its two equilibrium points. DC analysis does not distinguish between stable and unstable equilibrium points and is just as likely to output an unstable equilibrium point as a stable one.

Non-isolated solutions are those where there are a continuum of solutions.

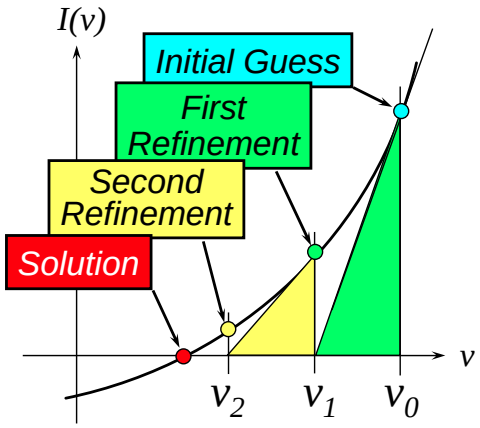


Non-Isolated Equilibrium Points

Non-isolated equilibrium points typically result from floating nodes (nodes with no DC path to ground). It also may occur when loops are formed with components that act as short-circuits at DC (for example, voltage sources or inductors). For practical reasons, circuit simulators use a version of NR that is not capable of solving a system of equations with a non-isolated solution, which for DC analysis is the same as having a non-isolated equilibrium point. Instead, the user is expected to find and fix such problems. Typically, a topology checker is provided that will quickly examine the circuit and find structural problems (nodes with no DC path to ground or loops of short circuits). Occasionally nodes can float because nonlinear devices turn off. This occurs inside a CMOS nand gate. To avoid this problem, most simulators add Gmin, a small conductor of typically 10^{-12} Siemens (10^{12} Ohms) across every nonlinear device that is capable of “turning off”. This generally avoids this problem but Gmin may affect the solution in some cases.

Newton-Raphson Algorithm (NR)

- Find zero crossing starting with an initial guess
- Iteratively refine guess by solving series of linear approximations
- NR also knows as Newton's Method



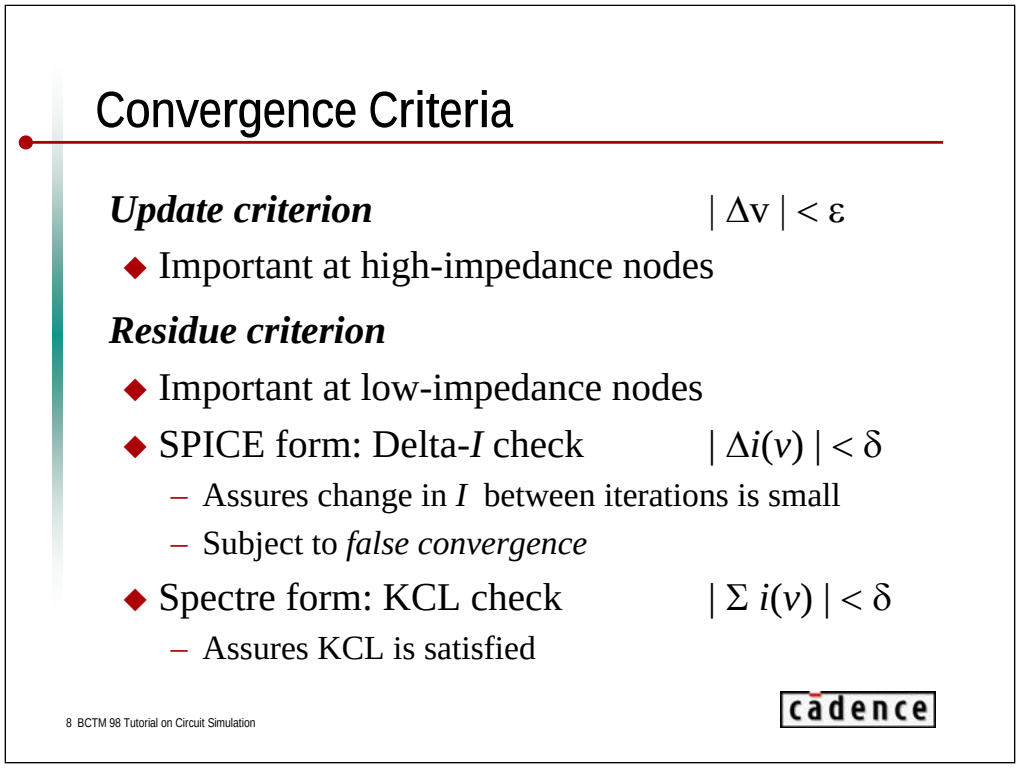
7 BCTM 98 Tutorial on Circuit Simulation

cadence

Newton-Raphson Algorithm

The goal of the Newton-Raphson (NR) algorithm is to find the solution to a nonlinear equation. This is the point where the curve crosses the *x*-axis. It is an iterative procedure that starts with at a starting point that is often referred to as the "initial guess". The equation and its derivative is evaluated at this point in order to construct a linear model of the equation. The value and the slope of the linear model match that of the equation at the evaluation point. The solution to the linear model, the point where the line crosses the *x*-axis, is found and used as the next "guess" as the procedure repeats. If things go well, each guess is closer to the solution than its predecessor. Thus, NR starts with an initial guess and iteratively refines it until it converges to the solution.

The Newton-Raphson algorithm is also referred to as Newton's method.

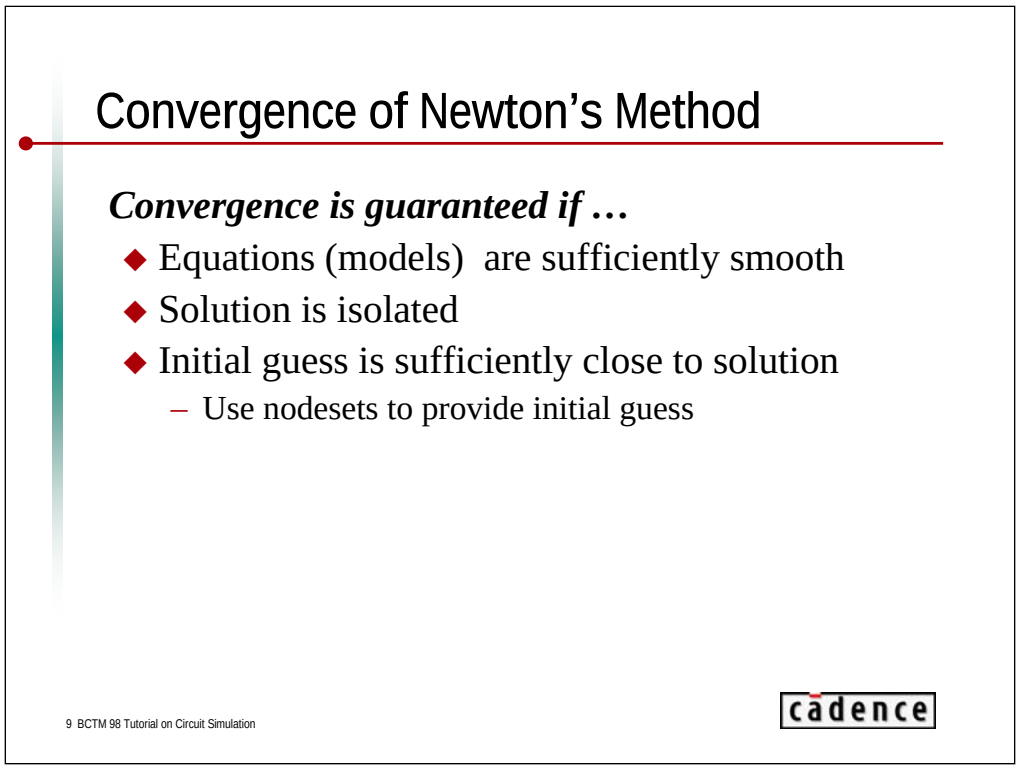


Convergence Criteria

NR is an iterative method, and so some stopping criteria must be applied to prevent it from continuing forever. Circuit simulators generally have two such criterion, both of which must be satisfied before convergence is assumed and the iteration stops. SPICE uses modified nodal analysis, and so it is reasonable to think of NR finding the voltage at which Kirchhoff's current law (though the situation is not quite that simple). In this case, one criterion applies to the update, or the difference in the voltage between two iterations. The second applies to the value of the equation. There are two alternative approaches used in this case. SPICE performs the ΔI check. Thus, it assures that the change in the current through any component between one iteration and the next is small. Notice that SPICE uses two Δ checks (a ΔV and a ΔI check). This means that if for some reason the Newton iteration stalls, which might occur if there is an error in the derivative, then both checks could be satisfied far from the actual solution. In this case, which is known as false convergence, the result computed by SPICE is incorrect.

The alternative to the ΔI check is the KCL check. In this case, the convergence criterion assures that KCL is satisfied within some tolerance. This approach is not subject to false convergence and so results in more reliable results.

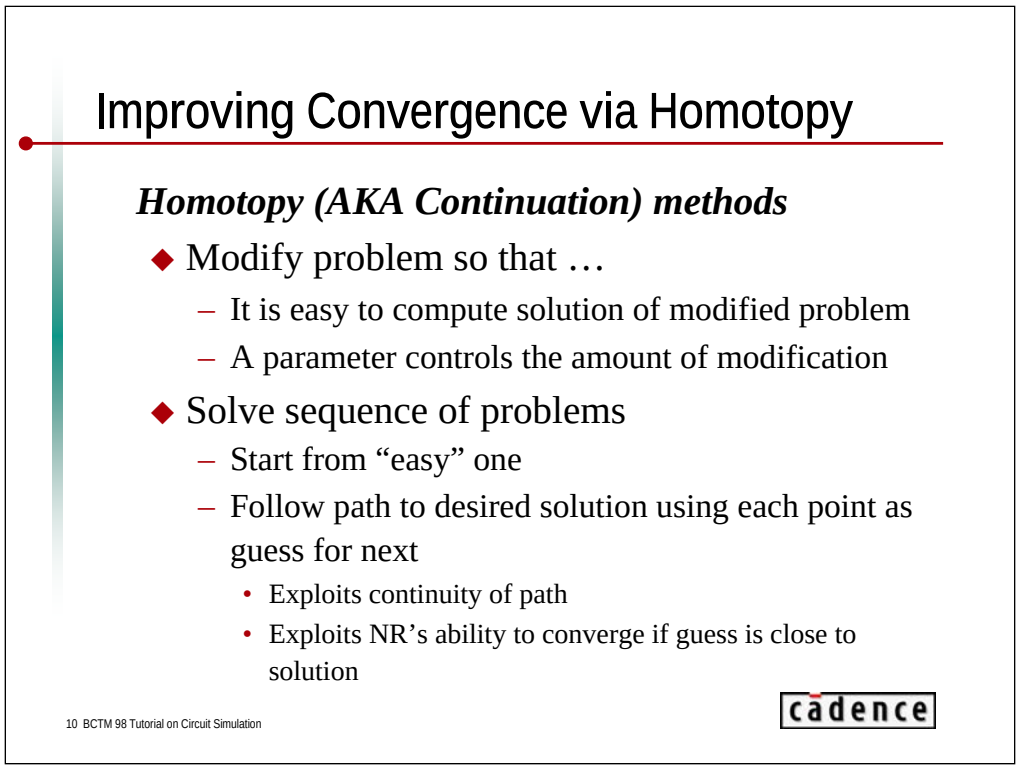
The KCL check is more difficult to implement efficiently than the ΔI check. As a result, the simulators that provide a KCL check provide it as an option that when selected provides improved reliability at the cost of reduced performance. Only one simulator, Spectre, was initially designed to support the KCL check efficiently. As such, there is no performance penalty associated with the KCL check. In this case, the ΔI check provides no advantage and so is not provided.



Convergence of Newton's Method

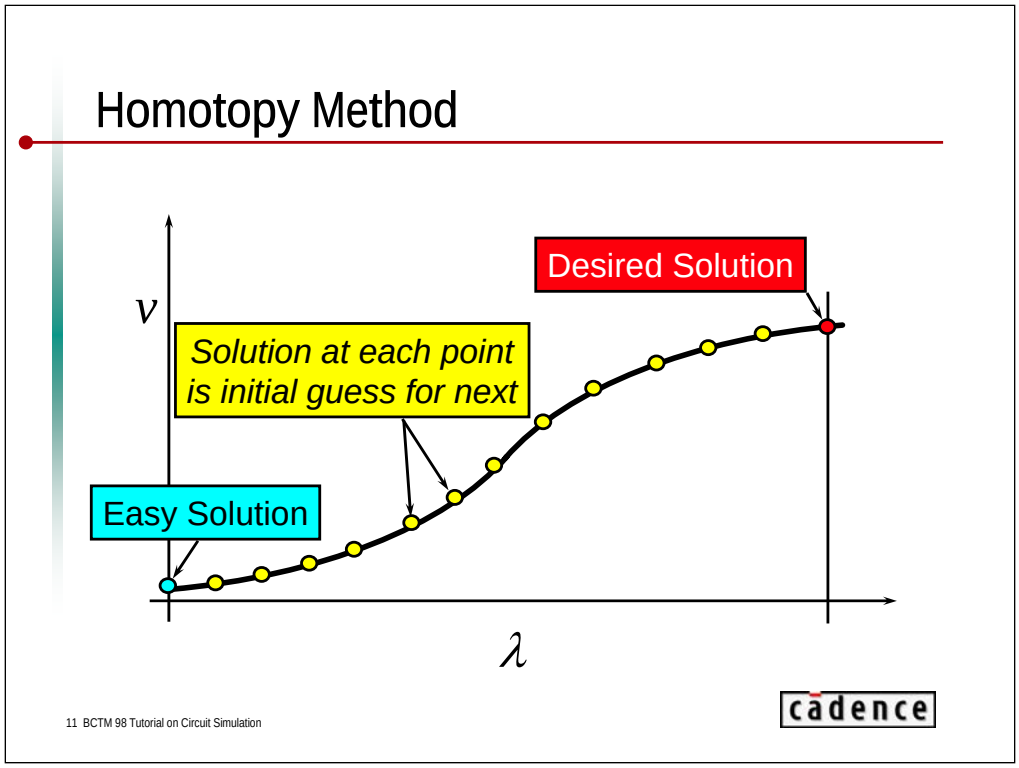
Convergence has long been a problem that has plagued circuit designers that use SPICE. And so designer's are often surprised when told that Newton's method has guaranteed convergence. While true, there are assumptions that must first be satisfied before convergence is assured, and therein lies the problem.

There are three conditions that must be satisfied before convergence is assured. First the circuit equations must be sufficiently smooth. This can be assured by proper modeling (removing the discontinuities from the models). This was a big problem early on, but it is generally much less of a problem today. The second condition is that the solution must be isolated. This was discussed previously. Lastly, the initial guess must be sufficiently close to the solution. This is the difficult condition to satisfy. Nodesets are provided to allow the user to specify the initial guess, but generally the designer does not know the solution well enough to provide a useful initial guess. However, this property exploited by a series of convergence aids that are referred to as continuation or homotopy methods that can greatly reduce the number of cases where convergence is a problem.



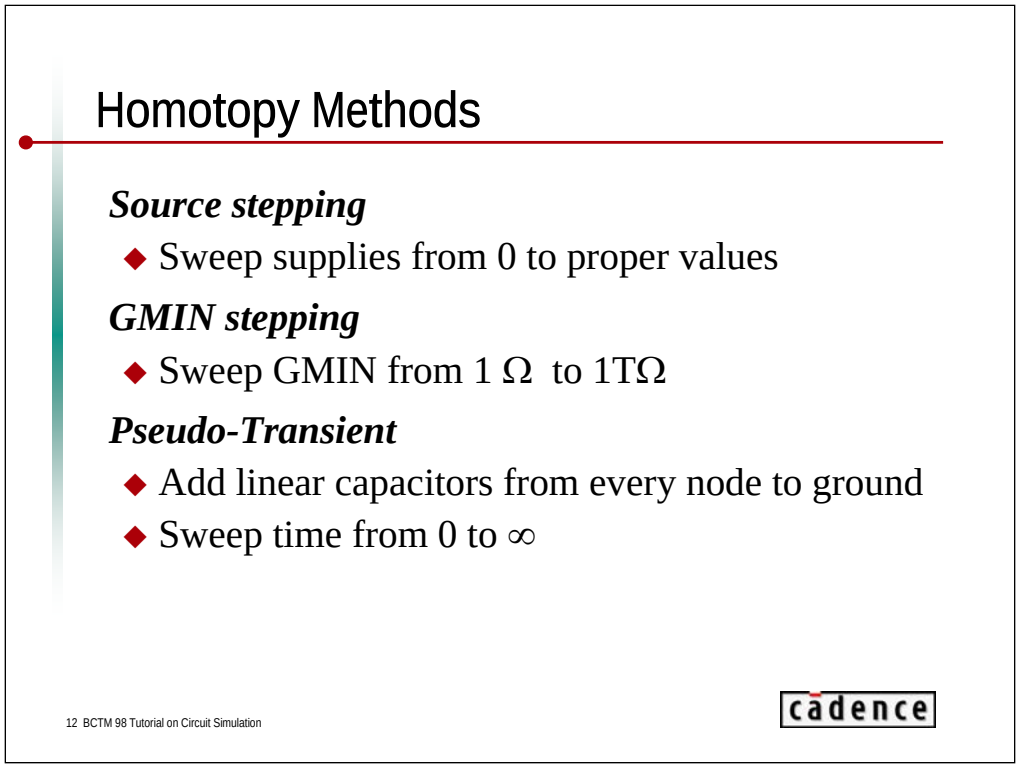
Improving Convergence via Homotopy

When a simulator is unable to achieve convergence with simple NR, it generally switches to a homotopy or continuation method. Homotopy and continuation are two names for the same process. The idea behind this process is to solve a series of problems the first of which is easy to solve and the last of which is the problem for which we need the solution. The series of problems is created by identifying or creating some parameter λ in the equations and varying the parameter. The parameterization must be such that:
1. the equations are easily solved for some value of the parameter, say $\lambda=0$,
2. the equations must revert to their original form for some other value of the parameter, say say $\lambda=1$, and
3. the solution trajectory must vary as a continuous function of the parameter between the two values.



Homotopy Methods

Continuation and homotopy methods start by setting the parameter to its initial value and solving the easy problem. They then step the parameter to towards its final value. On each step the equations are solved using the solution on the previous step as the initial guess. Since the solution trajectory is continuous, it is always possible to choose a step small enough so that the solution for the equations at the previous parameter value is a good guess for the new value of the parameter.



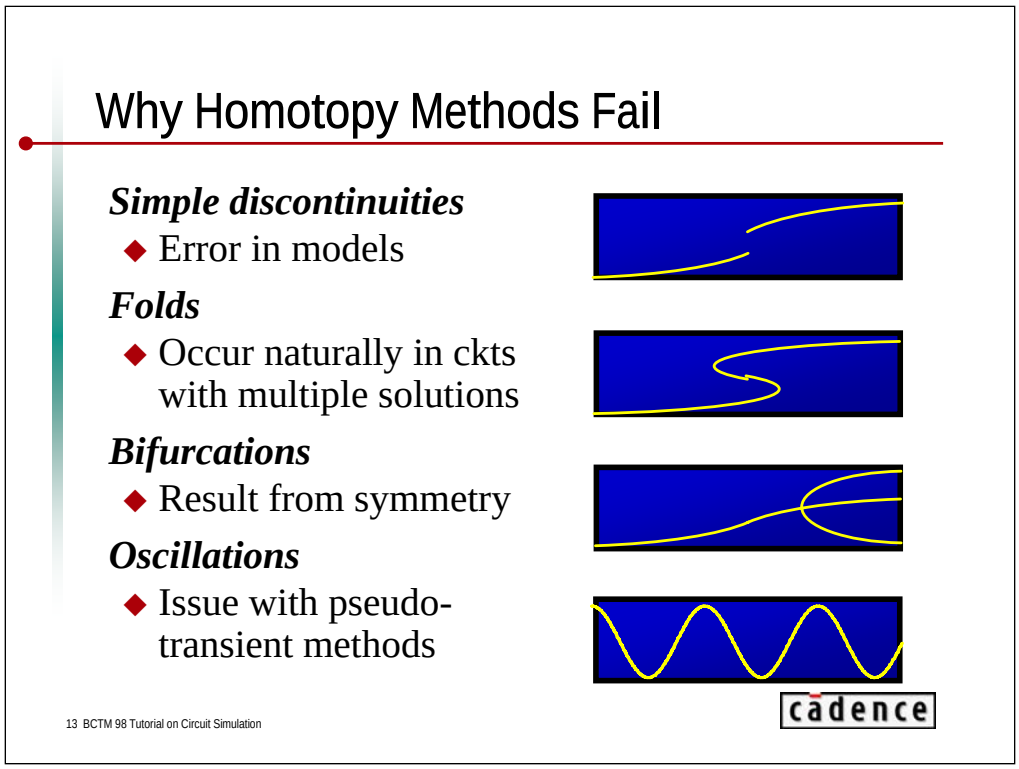
Homotopy Methods

Circuit simulators employ several types of homotopy methods. None are guaranteed to converge, so often several are available from the same simulator and are tried sequentially until one works.

The first method is source stepping. In this method, the parameter multiplies values of the independent sources and sweeps from 0 to 1. When the parameter is 0 the sources are all zero and the solution must also be zero. The parameter is then swept to 1, where the source take their desired values. In practice, source stepping trajectories are plagued by discontinuities called folds and so do not work very well.

Which Gmin stepping, the value of the Gmin conductors are swept from 1 Ohm to 10^{12} Ohms. Gmin stepping is also subject to folds, but is much less susceptible to them that source stepping. In generally Gmin stepping works pretty well.

The final approach is to pseudo-transient analysis. In this case 1 farad capacitors are added from every node in the circuit to ground. Each capacitor starts with an initial voltage equal to zero and a transient analysis is performed. In this case time is the homotopy parameter and it sweeps from 0 to infinity. Pseudo-transient analysis is not subject to folds, however the resulting circuit could oscillate, in which case the method would never converge.



Why Homotopy Methods Fail

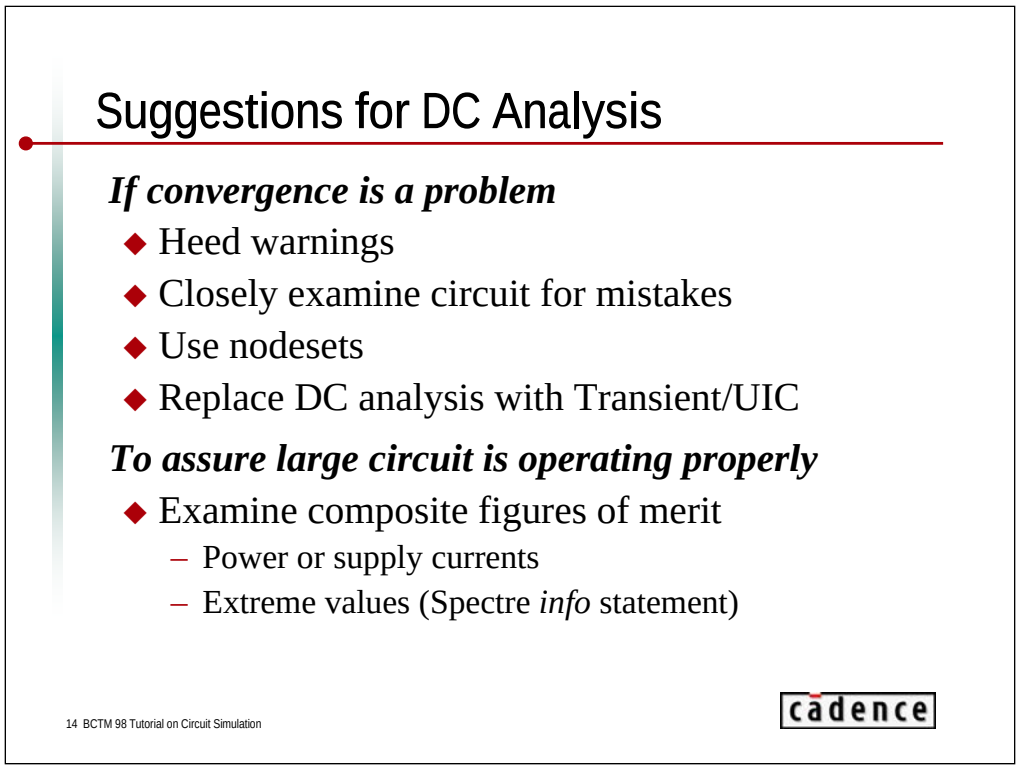
Homotopy methods are reliable if the homotopy trajectory is continuous when the parameter sweep is mapped onto the range λ=0 to 1. There are four common forms of discontinuity that will cause homotopy methods to fail.

The first is the simple discontinuity. This occurs when the model equations themselves are discontinuous.

The second type of discontinuity is folds, which is a natural consequence of circuits having multiple solutions. Homotopy methods can be reformulated to be arc-length methods which are able to handle folds, however such methods must do a certain amount of work to progress beyond each fold, and some circuits can have a large number of folds and so can be impractical to solve with arc-length methods. For example, the simple latch shown earlier had three equilibrium points and so two folds. If two such circuits were combined the resulting circuit could have $3^2 = 9$ equilibrium points and 9–1 = 8 folds. The combination of three such circuits would have 3^3–1 = 26 folds, and so on.

The third type of discontinuity is bifurcation, which results when both the circuit and the initial starting point is perfectly symmetric. This problem is easily avoided by providing a random starting point.

The final type of discontinuity is due to oscillation in the pseudo-transient approach. At first glance pseudo-transient appears to avoid problems with discontinuities. Indeed, it is not subject to any of the three previously mentioned discontinuities. However, pseudo-transient sweeps its parameter time for 0 to infinity. When the time is mapped to λ such that it varies between 0 and 1, the oscillation becomes infinitely fast as λ goes to 1, which is another form of discontinuity.



Suggestions for DC Analysis

Convergence problems are often a result of mistakes in the netlist. For example, unreasonable parameter values can cause convergence problems. Another common cause of convergence problems is forgetting to connect MOSFET back-gates to a bias supply. Also, the simulator may generate warming messages that could lead you the the problem area.

If the problem does not appear to be in your circuit, then you can try to provide the simulator a good guess of the solution using nodesets, perhaps from a previous simulation. Finally, if that is not practical, one can use a transient analysis with the UIC option set. Simply set the independent sources to their DC values and simulate long enough for the solution to settle to its final value. Be sure to write the final solution to a file so that it can be used as a nodeset for subsequent simulations.

Once the simulator has converged, you must still assure that the solution is as expected (the circuit may have been entered incorrectly). The easiest way to do that for a large circuit is to look at composite metrics, such as the circuit power dissipation or supply currents. If these metrics are well outside of the expected range, it generally indicates a problem with the circuit.

Small-Signal Analysis

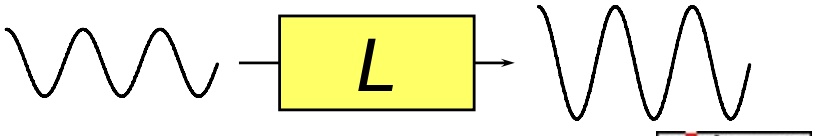
Apply small sinusoid

- Stimulus so small it does not generate nonlinear response
- Linearize circuit about DC operating point

Compute sinusoidal steady state solution

- Transfer functions

AC, Noise analyses

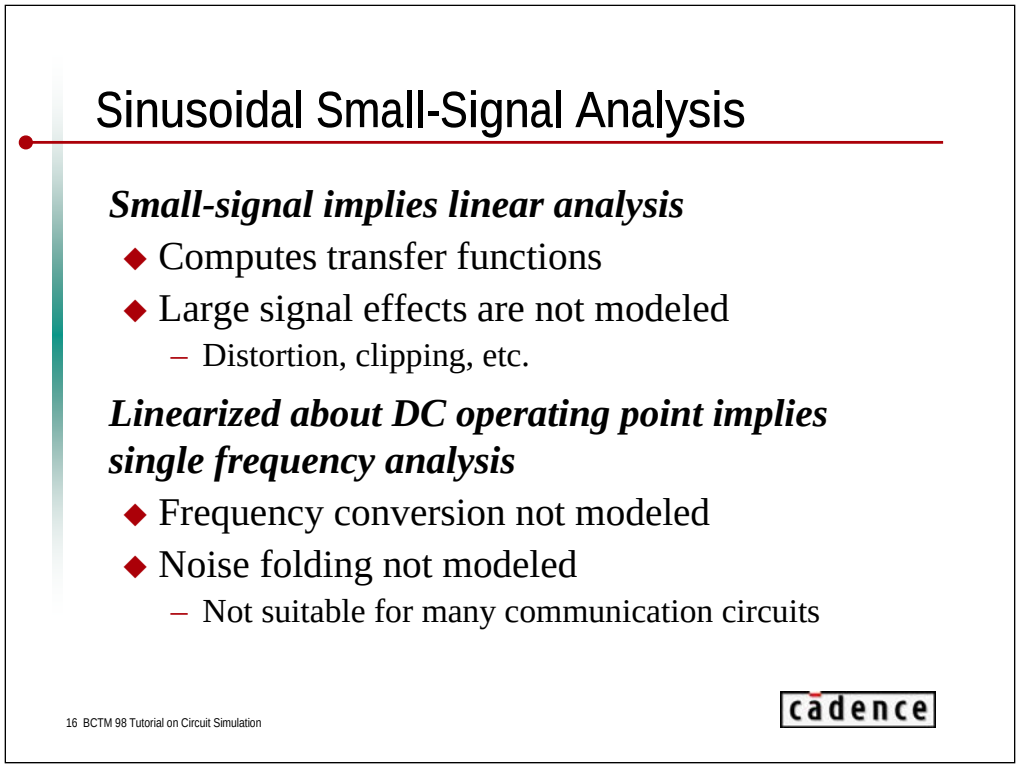


15 BCTM 98 Tutorial on Circuit Simulation

cadence

Small-Signal Analysis

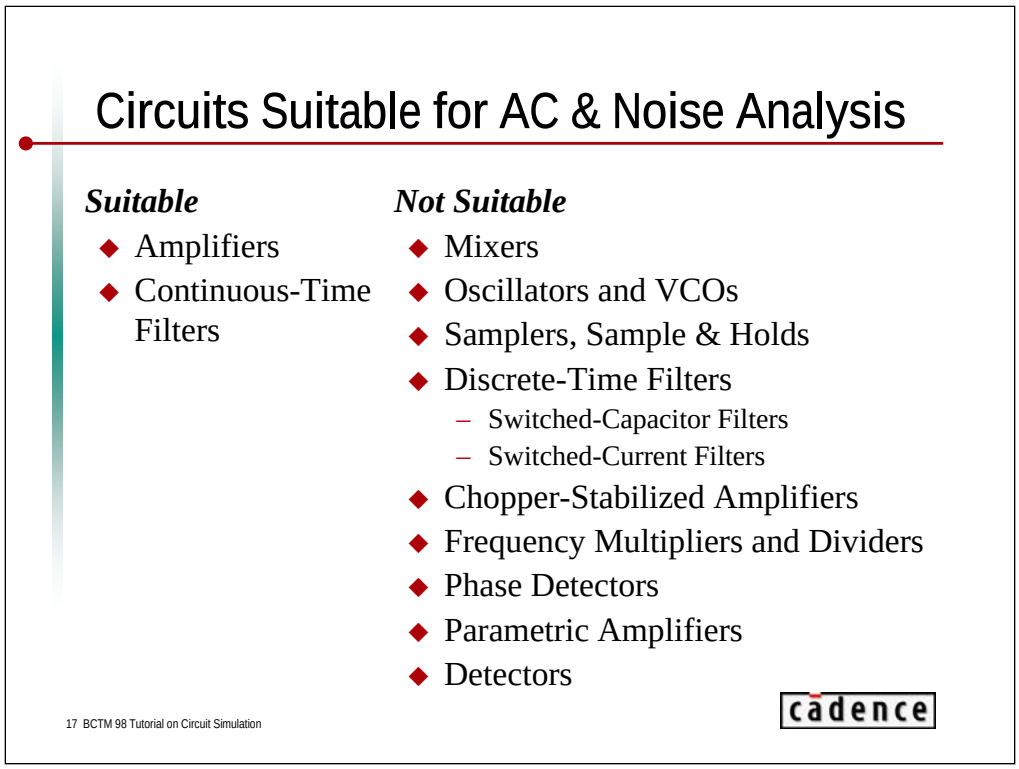
The sinusoidal small-signal analyses, AC and noise, both operate by assuming the DC solution is by a small sinusoidal excitation. The excitation is assumed to be so small that the response can be accurately computed by linearizing the circuit about the DC operating point and solving the linear system. The steady-state solution is computed directly. Since the circuit is linearized about a DC operating point the resulting representation of the circuit is linear and time invariant (LTI). The steady-state response of an LTI circuit to a sinusoid is a sinusoid at the same frequency. Thus, the AC analysis need only compute the magnitude and phase of the response relative to the input. Thus, it is useful for computing transfer functions. Noise analysis is a variation of AC analysis that computes the response of a circuit to collection of small noise sources.



Sinusoidal Small-Signal Analysis

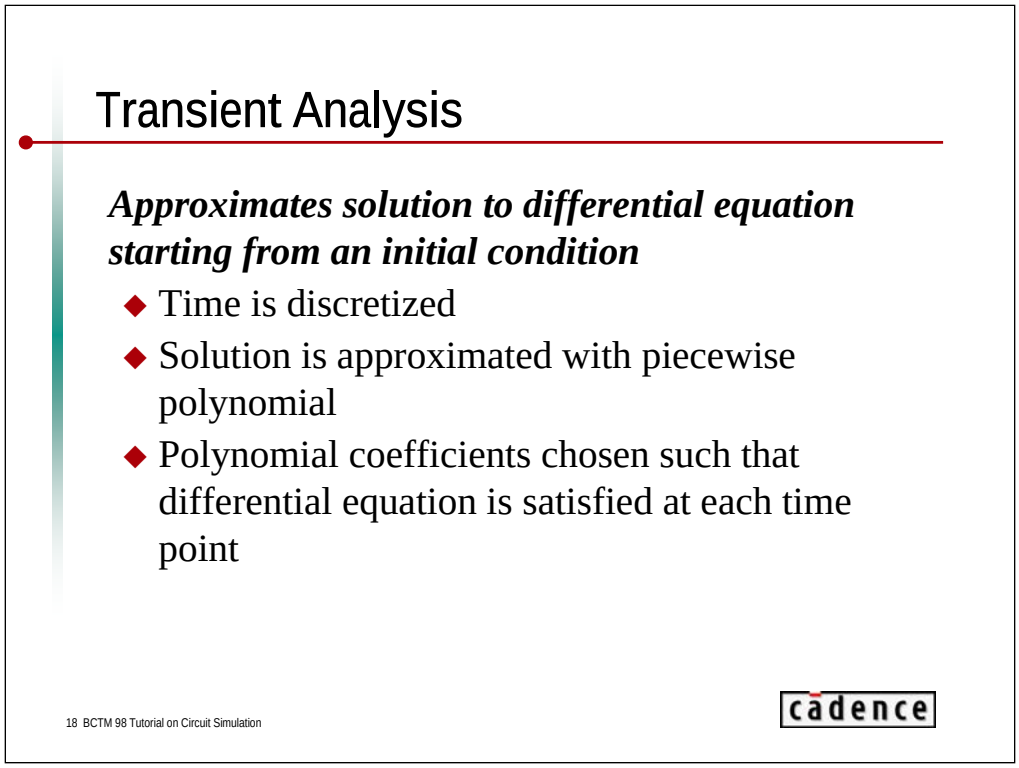
AC and noise are sinusoidal steady-state analyses performed on a linearization of the circuit about its DC operating point. Since the analysis is performed on a linear approximation to the circuit, the amplitude of the response will always be proportional to the amplitude of the specified stimulus. As such, AC analysis computes the transfer function of the circuit and does not take into account large signal effects such as distortion of clipping.

In addition, since the circuit is linearized about the DC operating point, the resulting LTI representation does not accurately model frequency conversion or noise folding. For information on small-signal analyses that accurately account for frequency conversion, refer to the tutorial on RF simulation.



Circuits Suitable for AC and Noise Analysis

The circuits that are suitable for AC and noise analysis are those that operate near their DC operating point. Many circuits, particularly those used in communication applications do not operate about a DC operating point. All of the circuits listed above that are unsuitable for AC and noise analysis only operate properly with a large, usually periodic, signal applied. For example, mixers need an LO signal to operate and switched-capacitor filters need a clock.



Transient Analysis

In transient analysis, the differential equation that represents the circuit is approximated with a difference equation that is then solved one timepoint at a time. This is done by first discretizing time, and then approximating the solution trajectory of the circuit with a low-order polynomial over each timestep. Using a low-order polynomials are used because it is possible to analytically compute their derivatives as a function of their end points. In this way, the derivatives in the differential equation are replaced by finite-difference approximations. Then the polynomials coefficients are chosen such that the finite-difference approximation to the differential equation is satisfied at each timepoint.

Integration Methods

Forward Euler (FE)

$$\dot{v}_{k-1} = \frac{1}{h}(v_k - v_{k-1})$$

Backward Euler (BE)

$$\dot{v}_k = \frac{1}{h}(v_k - v_{k-1})$$

Trapezoidal Rule (TR)

$$\dot{v}_k = \frac{2}{h}(v_k - v_{k-1}) - \dot{v}_{k-1}$$ ***(FE, BE combined)***

Gear's Backward Difference Formula (G2)

$$\dot{v}_k = \frac{3}{2h}v_k - \frac{2}{h}v_{k-1} + \frac{1}{h}v_{k-2}$$

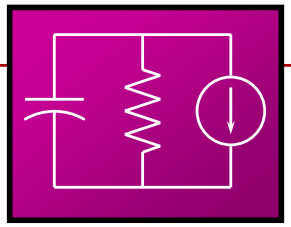
19 BCTM 98 Tutorial on Circuit Simulation

cadence

Integration Methods

There are many different finite-difference approximations that can be used, and each has different characteristics. The ones that are commonly used in circuit and timing simulation are shown above. Notice that the trapezoidal rule is simply the sum of forward Euler and backward Euler.

Example



Formulate KCL

$$C\dot{v}(t) + Gv(t) + i(t) = 0$$

Replace d/dt operator with discrete-time approximation

- Forward Euler — An Explicit Method

$$\frac{C}{h}(v_k - v_{k-1}) + Gv_{k-1} + i_{k-1} = 0$$

- Backward Euler — An Implicit Method

$$\frac{C}{h}(v_k - v_{k-1}) + Gv_k + i_k = 0$$

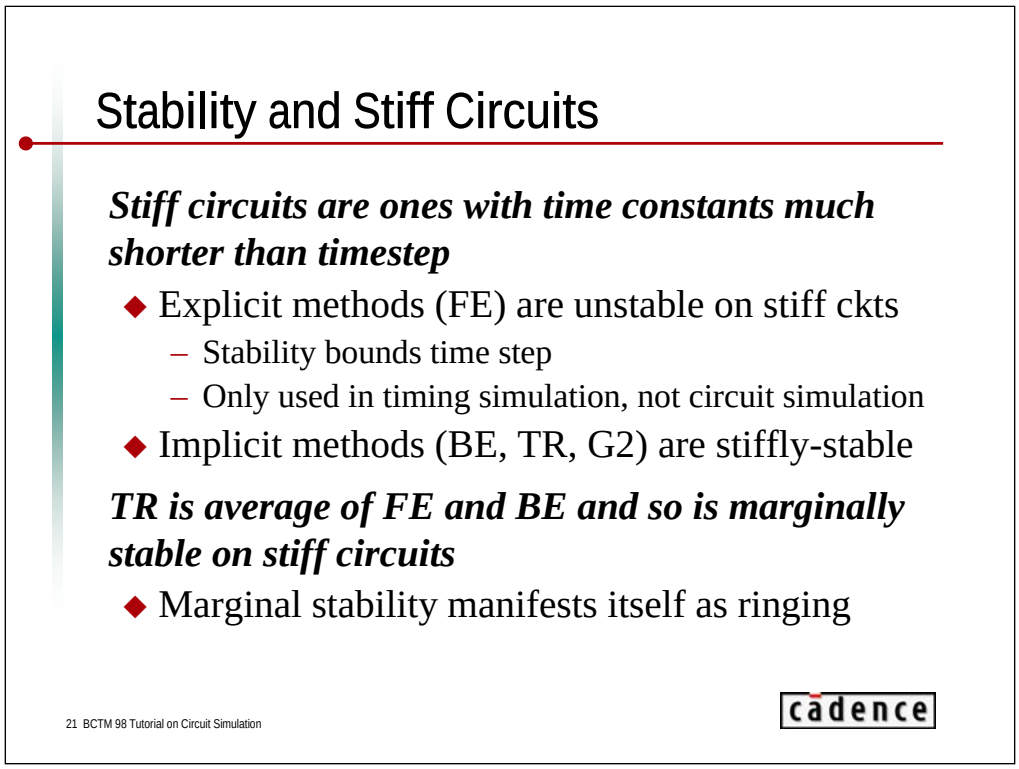
20 BCTM 98 Tutorial on Circuit Simulation

cadence

Example

In this example, the equation to be solved is simply Kirchhoff's Current Law combined with the model equations. The derivative is replaced by either the forward Euler or the backward Euler finite difference approximation. h is the timestep. Doing so converts the differential equation into a difference equation that can be solved one step at a time.

Forward Euler is an explicit method and backward Euler is an implicit method. This distinction is important, though in this example it is not obvious why one is implicit and the other explicit. Unfortunately I will not give a rationale here. If you are interested, I suggest you refer to a book on numerical methods.



Stability and Stiff Circuits

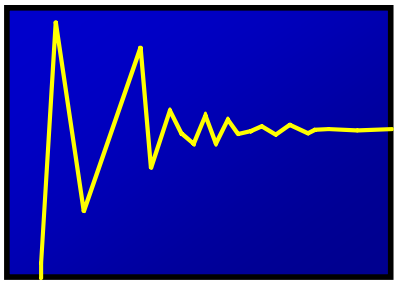
Stiff circuits are those that contain time constants that are much shorter than the timestep. In order for the timestep to be larger than a time constant, the time constant must not be active, otherwise the timestep must be reduced to follow the time constant. Such time constants are common in electrical circuits. Even if the time constants are initially excited, the response decays rapidly. Once their response has died out, it is important to be able to increase the size of the timestep to efficiently follow the response of the rest of the circuit, which is slow relative to the time constant. However, some integration methods, forward Euler in particular, are unstable on stiff circuits. Thus, when using forward Euler on stiff circuits the timestep must be chosen small relative to the fastest time constant to avoid stability problems. For this reason, forward Euler is not efficient when applied to stiff circuits.

Backward Euler, trapezoidal rule, and Gear's second-order backward difference formula are all stiffly stable, meaning that they are stable on stiff circuits. However, trapezoidal being the combination of forward and backward Euler is only marginally stable. As a result, trapezoidal rule will exhibit a point-to-point ringing on stiff circuits.

Trapezoidal Rule Ringing

TR rings on stiff circuits

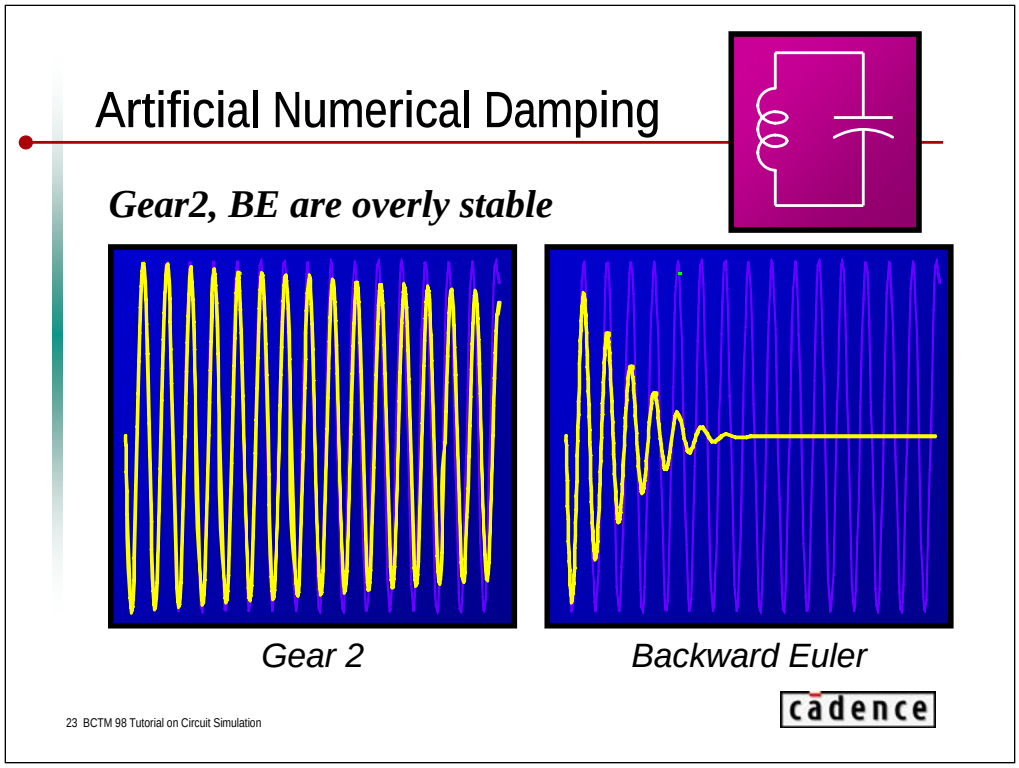
- Marginal stability
- Characteristic point-to-point ringing



cadence

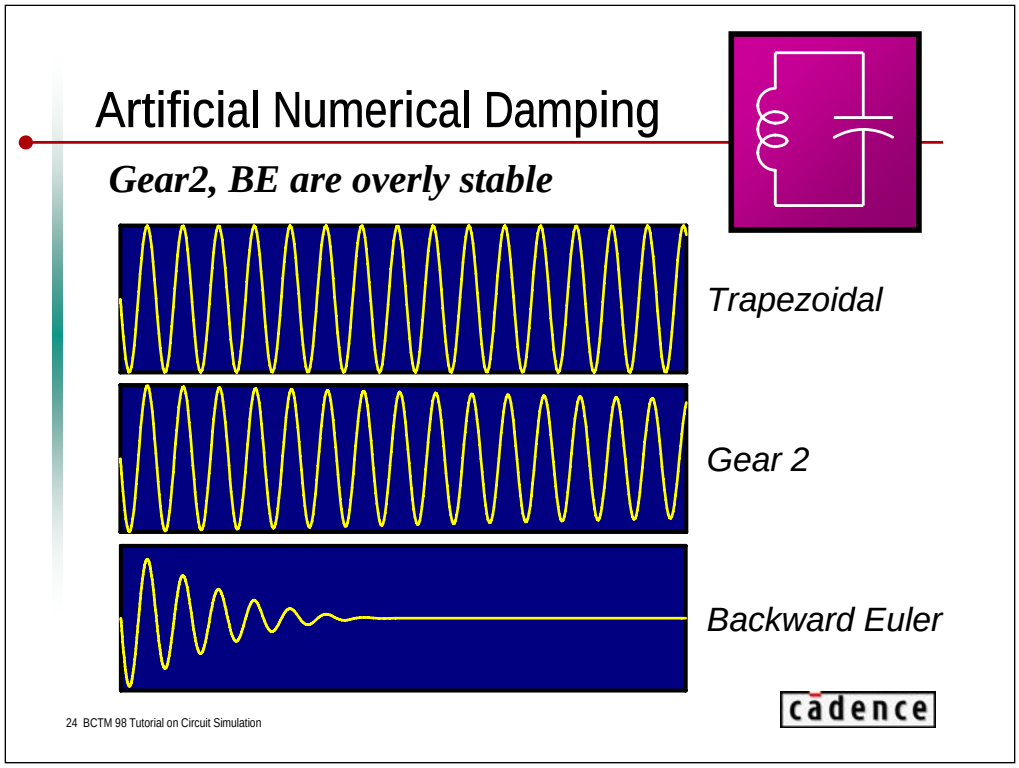
Trapezoidal Rule Ringing

Trapezoidal rule exhibits a characteristic point-to-point ringing when applied to stiff circuits. This ringing can be reduced and often eliminated by adjusting tolerances so that the timestep size is reduced.



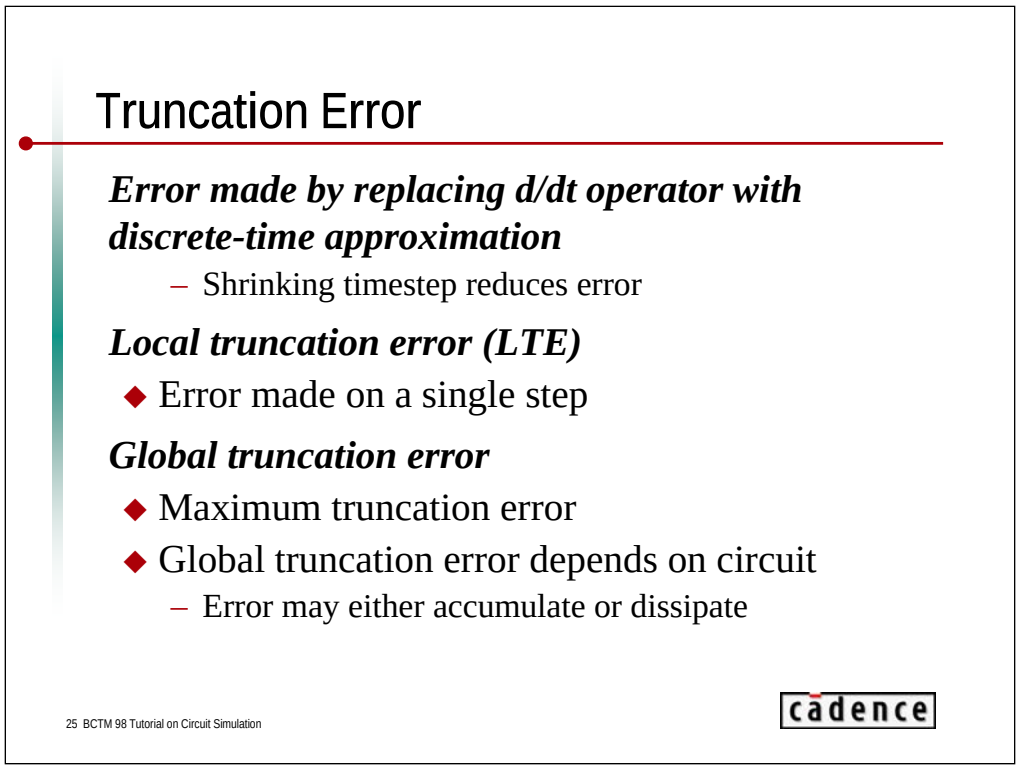
Artificial Numerical Damping

Backward Euler and Gear2 are not without faults. Both are overly stable and so exhibit artificial numerical damping. Thus they add loss to circuits. This is noticeable when simulating an LC tank circuit. If excited, it should oscillate forever because the loss mechanisms are not included in the simulation. And if trapezoidal rule is used, that is exactly what happens. However, if either Gear2 or backward Euler are used, the amplitude of the oscillation asymptotically decays. They add damping even though none exists in the circuit. Backward Euler adds considerably more damping than Gear2.



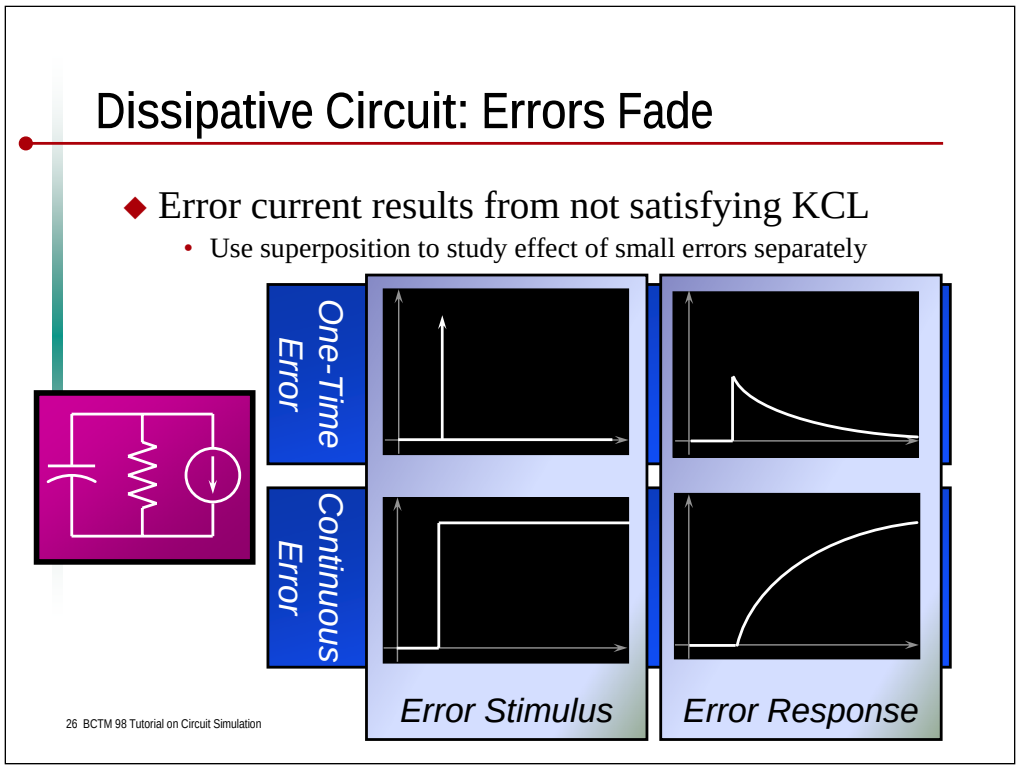
Artificial Numerical Damping

Backward Euler and Gear2 are not without faults. Both are overly stable and so exhibit artificial numerical damping. Thus they add loss to circuits. This is noticeable when simulating an LC tank circuit. If excited, it should oscillate forever the loss mechanisms are not included in the simulation. And if trapezoidal rule is used, that is exactly what happens. However, if either Gear2 or backward Euler are used, the amplitude of the oscillation asymptotically decays. They add damping even though none exists in the circuit. Backward Euler adds considerably more damping than Gear2.



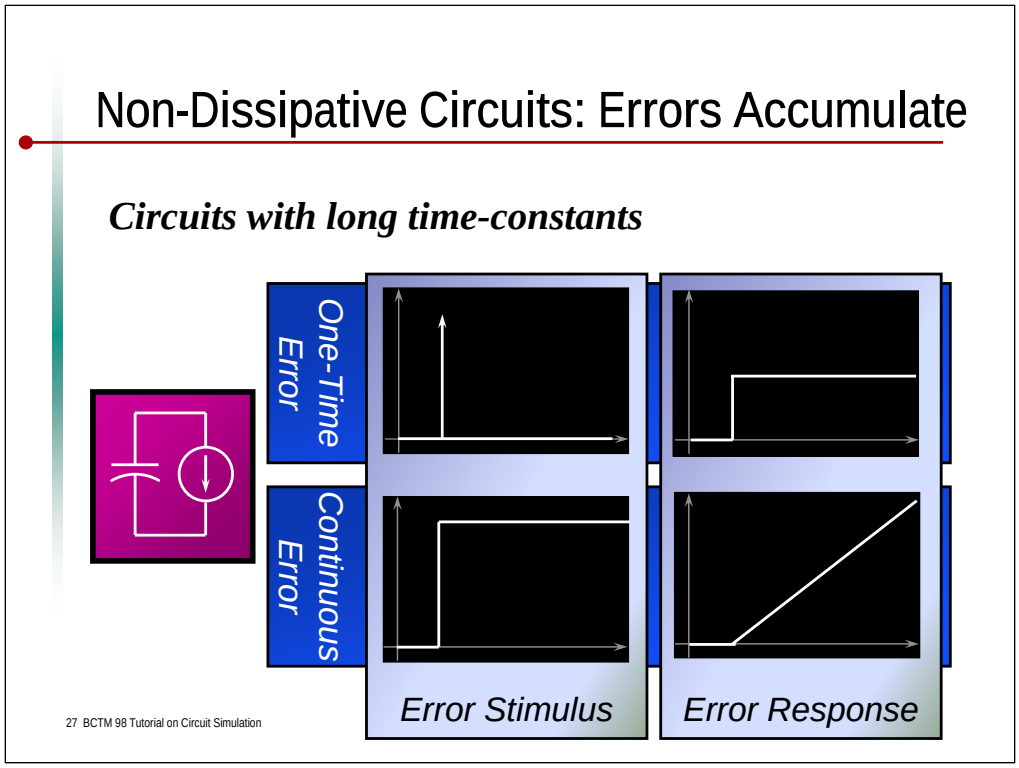
Truncation Error

Truncation error is the error made by an integration method when it replaces the time-derivative with a finite-difference approximation. At each time point a small amount of error is created. This is referred to as local truncation error or LTE. The global truncation error is the accumulated effect of the LTE made on each step. The LTE is determined by the integration method, but how that error accumulates to form global truncation error is determined by the circuit.



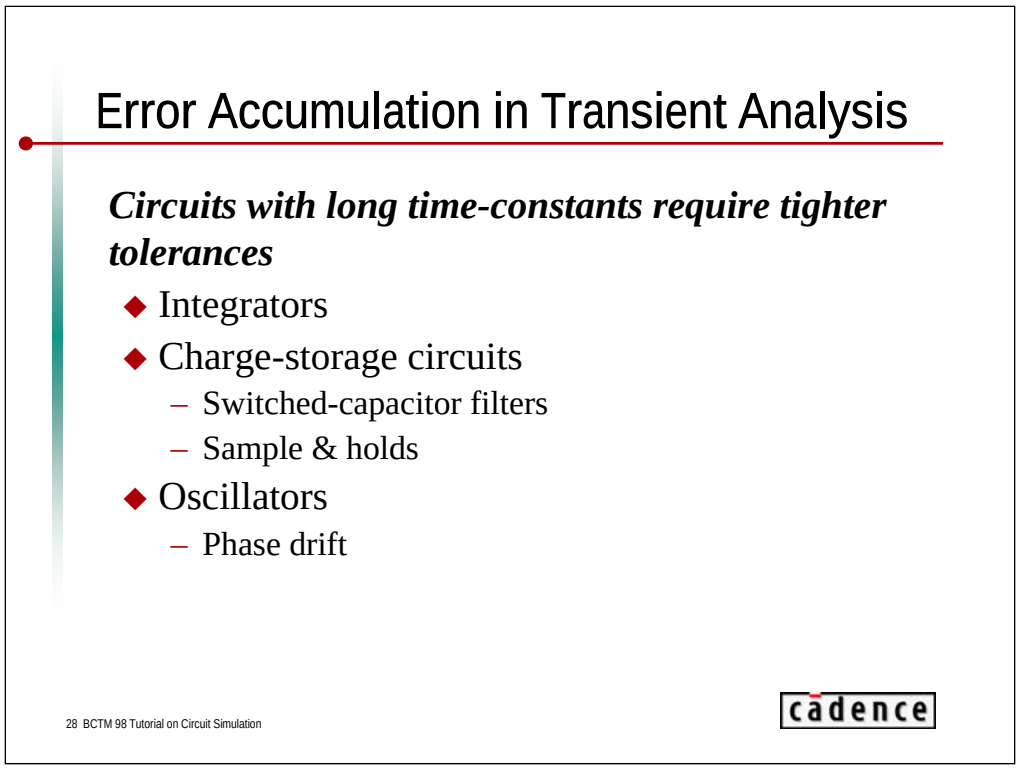
Dissipative Circuit: Errors Fade

If the circuit is dissipative, i.e. if the circuit has no long time constants, then the error made on each timestep will fade with time. The time constants of the error will be those of the circuit.



Non-Dissipative Circuit: Errors Accumulate

If the circuit has long time constants, then the local truncation errors tend to accumulate. The circuit above has an infinitely long time constant and the effect of a one time error is retained for ever. The effect of a sequence of errors is integrated.



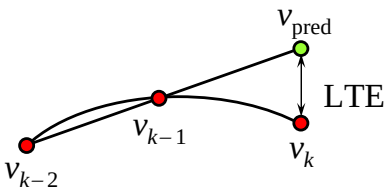
Error Accumulation in Transient Analysis

If the circuit has long time constants, then the local truncation errors tend to accumulate. Thus, one has to be more careful on circuits with long time constants. Examples of such circuits include oscillators, integrators, and charge-storage circuits. In the case of an oscillator, the errors will cause the phase to drift. The integration methods tend to under estimate curvature and so in an oscillator the LTE is correlated in such a way to cause the phase to drift so that the simulator slightly underestimates the oscillator frequency.

Estimating Local Truncation Error

Simulators choose timestep to control LTE

- Integration methods are exact if trajectory follows low-order polynomial
  - First-order methods (FE, BE) are exact for lines
  - Second-order methods (TR, G2) are exact for parabolas
- LTE is difference between actual trajectory and trajectory extrapolated using low-order polynomial



$$LTE = v_k - v_{pred}$$

cadence

Estimating Local Truncation Error

In transient analysis the size of the timestep is chosen so as to control LTE. To do so, one needs an estimate of the LTE.

In transient analysis, the waveform is discretized and approximated by a piecewise low-order polynomial. If the actual solution trajectory follows a low-order polynomial, then solution computed by the integration method will be exact. Thus, if the true solution can be represented exactly by the polynomial used in the integration method, there will be no LTE. In this case the solution predicted by extrapolating with a polynomial of the same order from previously computed points will exactly match the true solution. Thus, the measure we will use for the LTE is difference between the extrapolate solution and the computed solution. If that LTE measure is above a threshold, the timepoint will be rejected and the timestep shrunk. If it is below a threshold, the next timestep will be larger than the one used at the current step.

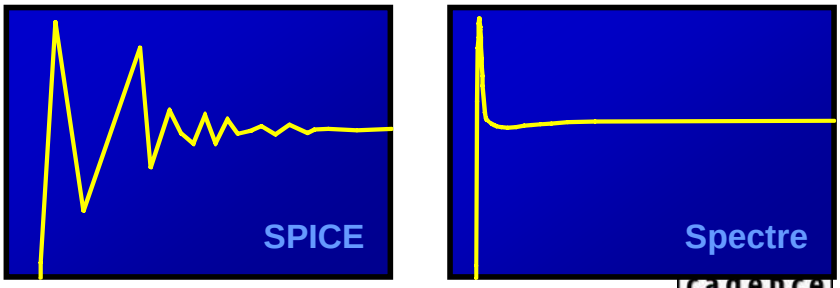
Controlling Error in Charge vs. Voltage

Voltage is interesting quantity, not charge

SPICE controls error in charge

- Large voltage error possible on stiff circuits
  - Small charge error on small capacitor could cause large error in voltage

Spectre controls error in voltage

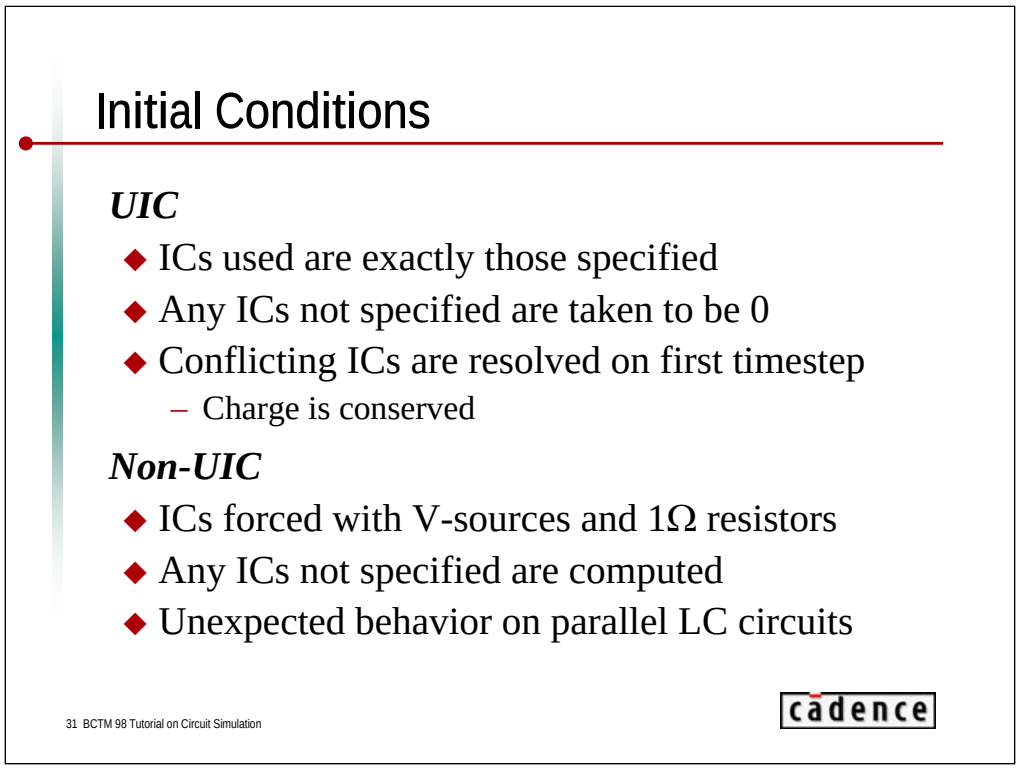


30 BCTM 98 Tutorial on Circuit Simulation

cadence

Controlling Error in Charge vs. Voltage

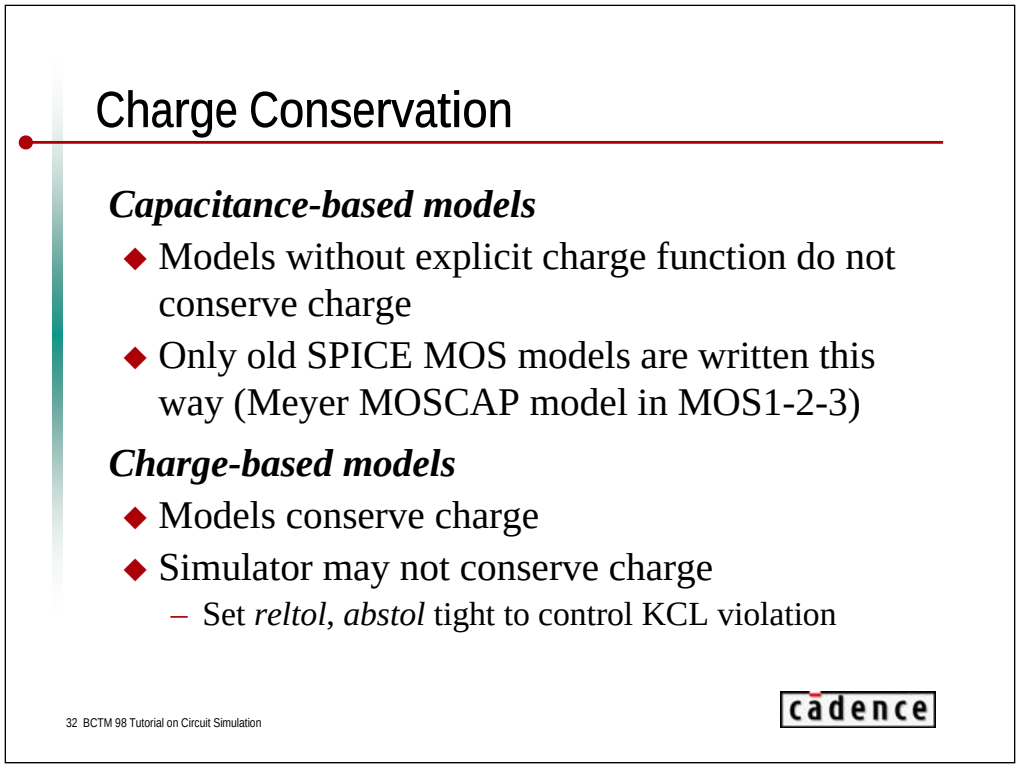
SPICE performs LTE computations based on charge waveforms. But charge is an uninteresting variable. The variable that is of interest to the user is most often voltage, so it is best to control the LTE in voltage rather than charge. On stiff circuits, the LTE in charge can be quite small even though the LTE in voltage is large. As a result, SPICE can give poor answers as shown above. Both waveforms are the voltage waveform computed by the simulators using default tolerances. The fact that Spectre controls LTE in voltage rather than charge results in its answer being much more accurate.



Initial Conditions

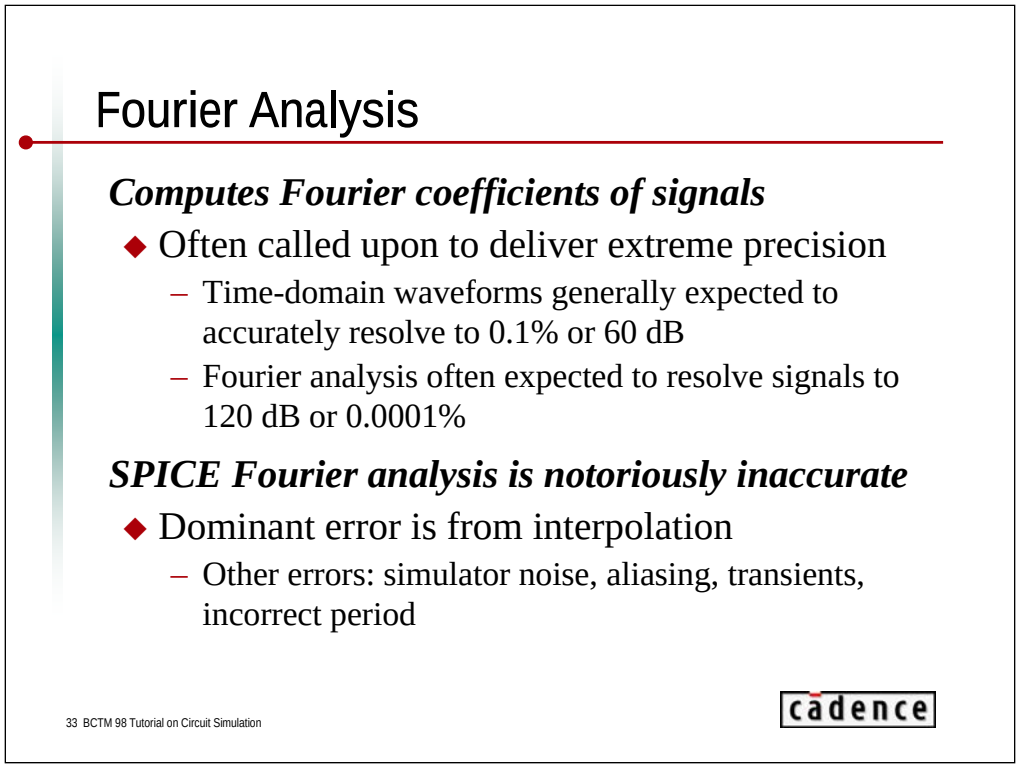
There are two types of initial conditions, UIC and non-UIC. UIC initial conditions are used exactly as specified, however any unspecified are taken to be zero. Any conflicting initial conditions, which might occure on loops of capacitors or cutsets of inductors, are resolved in one timestep by conserving charge and flux.

Non-UIC initial conditions are forced replacing capacitors with voltage sources in series with small resistors and by replacing inductors with current sources in parallel with large resistors. The DC solution is then computed. This works reasonable well in many cases and unlike with UIC initial conditions, allows a small set of initial conditions to be specified with the remaining set computed. However, in some situations, such as with parallel LC tanks, the results are unexpected.



Charge Conservation

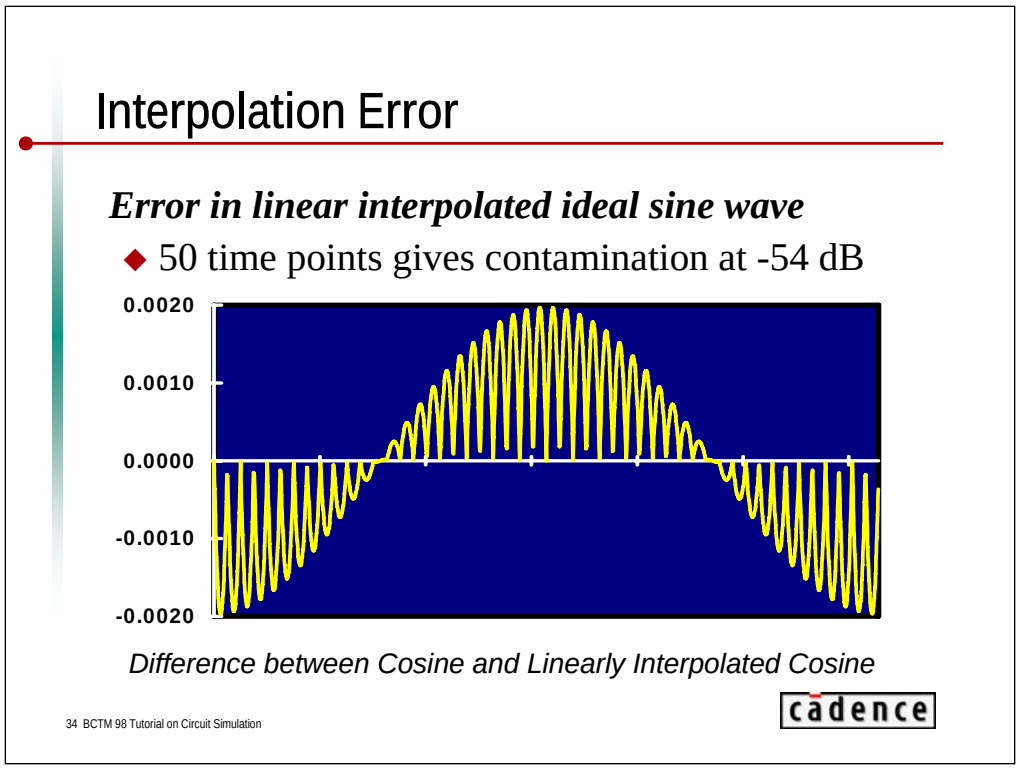
Old MOS models often do not conserve charge and so should not be used where charge conservation is important, such as on charge storage circuits such as switched-capacitor circuits. However, even when charge conserving models, the simulator itself only approximates KCL on each step and so does not precisely conserve charge. To improve this situation, tighten *reltol* and *abstol*.



Fourier Analysis

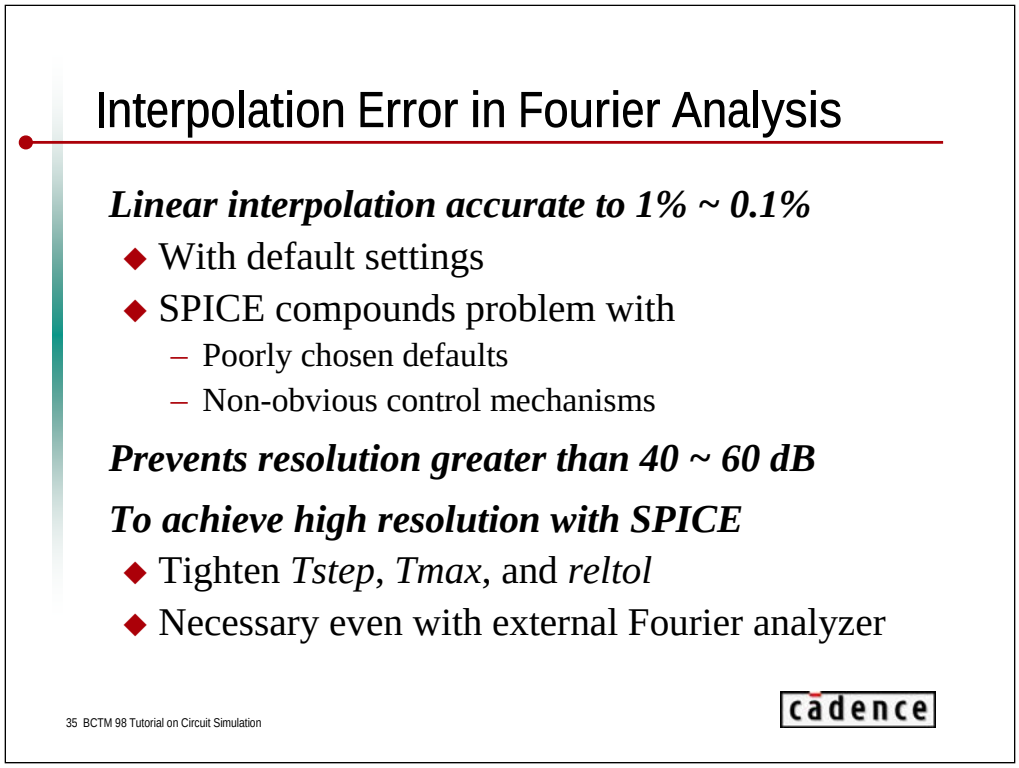
Fourier analysis is used to compute the spectrum of transient waveforms. Because of the nature of Fourier analysis, it is often called upon to deliver extreme precision. For example, users often want 120 dB of resolution from Fourier analysis, which is 1 part per million, whereas they are happy with 60 dB of resolution or one part per thousand with the results from standard transient analysis.

SPICE's Fourier analysis is notoriously inaccurate, often giving no better than 40 dB accuracy.



Interpolation Error in Fourier Analysis

There are many sources of error in Fourier analysis, including incomplete settling of the waveform, using the wrong period, aliasing, and simulator noise. However, the most troubling source of error results from interpolation. Since SPICE generates unequally spaced points and Fourier analysis requires equally spaced points, the waveforms must be interpolated before Fourier analysis. The above figure shows the error generated as a result of interpolation. An ideal cosine wave was sampled and linearly interpolated using 50 points. This interpolated waveform was then subtracted from the original waveform to show the error created by interpolation. The spectrum computed for the interpolated waveform is equal to the spectrum of the original waveform plus the spectrum of the error waveform shown. The error waveform has spurs as high as –54 dB which directly act to limit the resolution of the Fourier analysis.



Interpolation Error in Fourier Analysis

Interpolation errors typically limit Fourier analysis resolution to 40-60 dB with default settings. The timestep must be significantly reduced in order to lower the interpolation error. SPICE compounds the problem by providing poorly chosen defaults and non-obvious control mechanisms. To improve Fourier analysis accuracy, tighten time step and the tolerances. Methods for doing this are described in my book *The Designer's Guide to SPICE and Spectre*, published by Kluwer Academic Publishers in 1995.

Outline

Traditional SPICE Simulation

- DC
- AC and Noise
- Transient
- Fourier

Timing Simulation ←

AHDLs and MSHDLs

RF Simulation (tomorrow)

cadence

Dynamic Timing Simulation

Fast, reduced accuracy simulation for large MOS digital circuits

Basic approach

- Partition into small subcircuits
- Explicit integration methods
- Simplified Models

Requires strong assumptions about circuit

Consequence of violates assumptions

- Incorrect, though often plausible, results

cadence

Partitioning

Partition into single node or small subckts

- Reduces/eliminates cost of matrix operations
- Exploits multirate behavior

Partitioning requires …

- Loosely coupled subcircuits
  - Largely unidirectional subckts with weak feedback
  - Capacitors to ground at every node
- Scheduling
  - Subckts evaluated in order of signal propagation
  - No tight feedback loops

cadence

Explicit Integration Methods

Timing simulators use forward Euler

- Eliminates cost of matrix
- Eliminates cost of nonlinear solve
  - Assuming only linear capacitors

Explicit integration methods require …

- Nonstiff circuits
  - All time constants roughly same size

cadence

Simplified Models

Linearize capacitors

Discard floating capacitors

- Approximate with grounded Miller capacitors

Discard subtleties of models

- Back-gate bias effects
- Subthreshold effects
- Nonlinear capacitance

cadence

Limitations of Timing Simulation

Suitable only for MOS digital circuits

- Not all MOS digital circuits
  - Memories (memory cell, sense amps)
  - Busses can be a problem
- No analog circuits
- No bipolar circuits

cadence

Mixed-Signal Timing Simulation

Timing simulation extended to handle small analog & bipolar subcircuits

- Timing simulation on digital sections
- Circuit simulation on analog sections

Requires

- Reliable identification of analog/bipolar sections
- Full circuit simulation on those sections

cadence

MS Timing Simulation Requires ...

Subcircuit-based partitioning

- Analog/bipolar circuits must be kept together
- Partitioner must reliably recognize analog cells
- Partitioner must reliably handle feedback

Implicit integration methods

- At least for analog/bipolar subcircuits

Two sets of models

- Accurate set for analog subcircuits
- Approximate set for digital subcircuits

cadence

Timing Simulation vs. Circuit Simulation

Timing simulation only beneficial when digital circuitry dominates simulation time

- Little performance benefit over circuit simulation if analog circuitry dominates

Timing simulation much riskier

- Much higher chance of incorrect results
- Simulators often require significant manual tweaking to avoid
  - Inaccurate results
  - Performance problems

cadence

Outline

Traditional SPICE Simulation

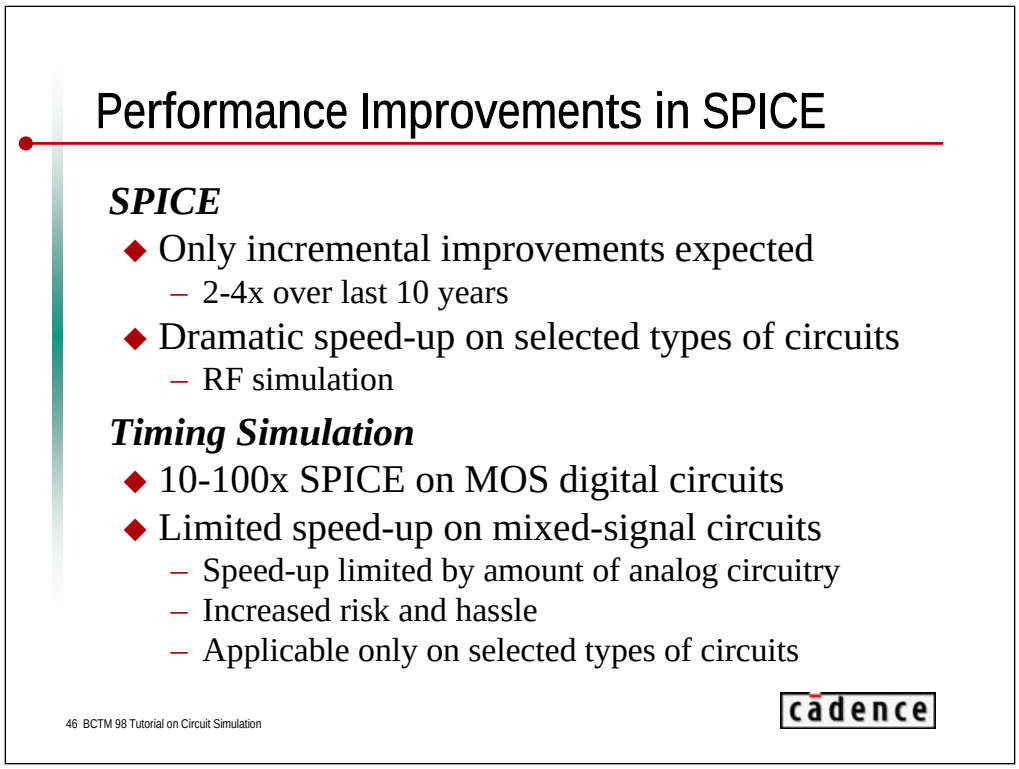
- DC
- AC and Noise
- Transient
- Fourier

Timing Simulation

AHDLs and MSHDLs ←

RF Simulation (tomorrow)

cadence



Performance Improvements in SPICE

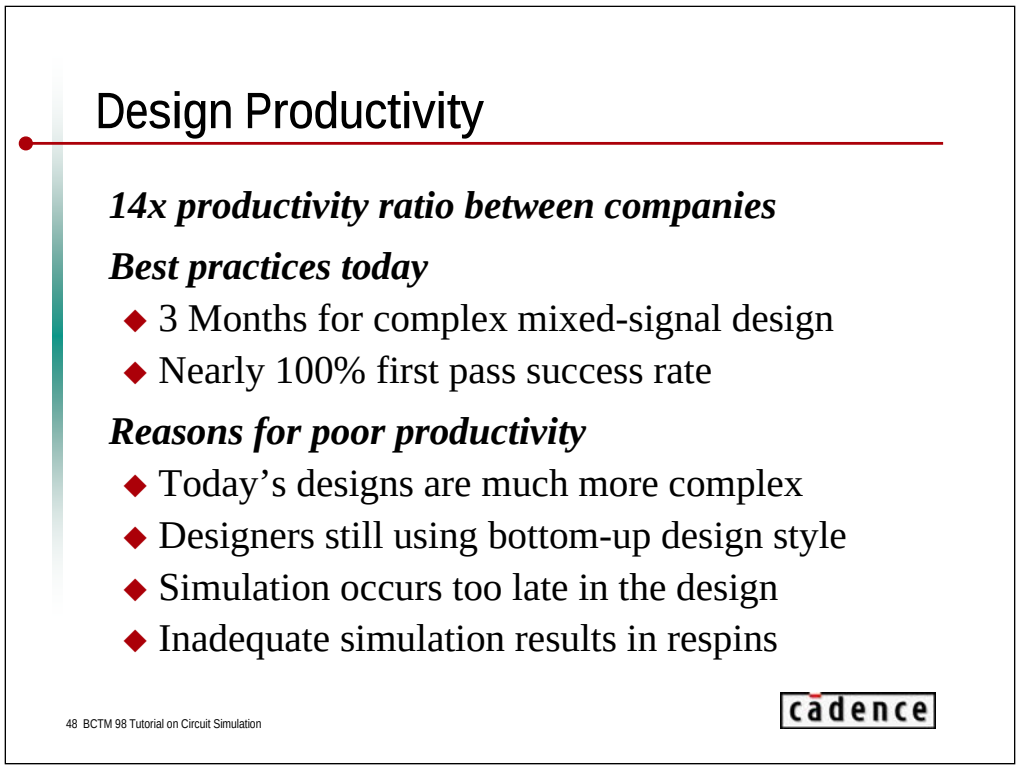
SPICE-like simulators have incrementally improved their performance over time. There has been perhaps a 2-4x speed performance improvement over the last ten years independent of any improvements due to the underlying hardware. And except certain classes of circuits where it is possible to apply special purpose simulation algorithms, no dramatic performance improvements are anticipated.

Timing simulation offers dramatic performance improvements on MOS digital circuits, however the speedup on mixed-signal circuits is limited by the analog circuitry. Typically the best that is achieved is a 2-5x speedup. In addition, timing simulation often struggles on analog circuitry. Users often have to spend considerable time adjusting the simulator to get it to behave properly, which is not always possible.

Some are proposing simulation on parallel processors as a way to accelerate SPICE simulation. However, this approach also provides limited speedup, typically 2-4x, an only on specific types of circuits.

Performance Improvements?

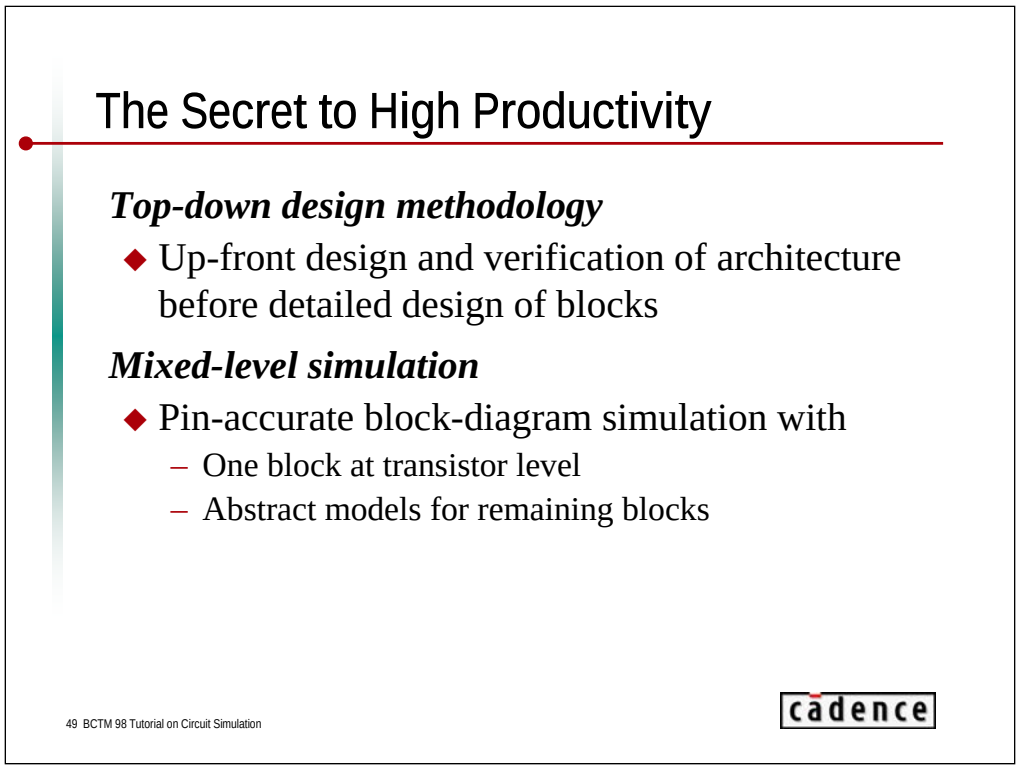
Improving the performance of SPICE will help designers be more productive, however substantial productivity improvement only comes by changing ones approach. In particular, using only transistor level simulation, regardless of how fast the simulator, still requires the designer to design to the transistor level before the design can be verified. Productivity can be increased dramatically by moving the verification earlier in the design. This is the idea behind using an hardware description langauge.



Design Productivity

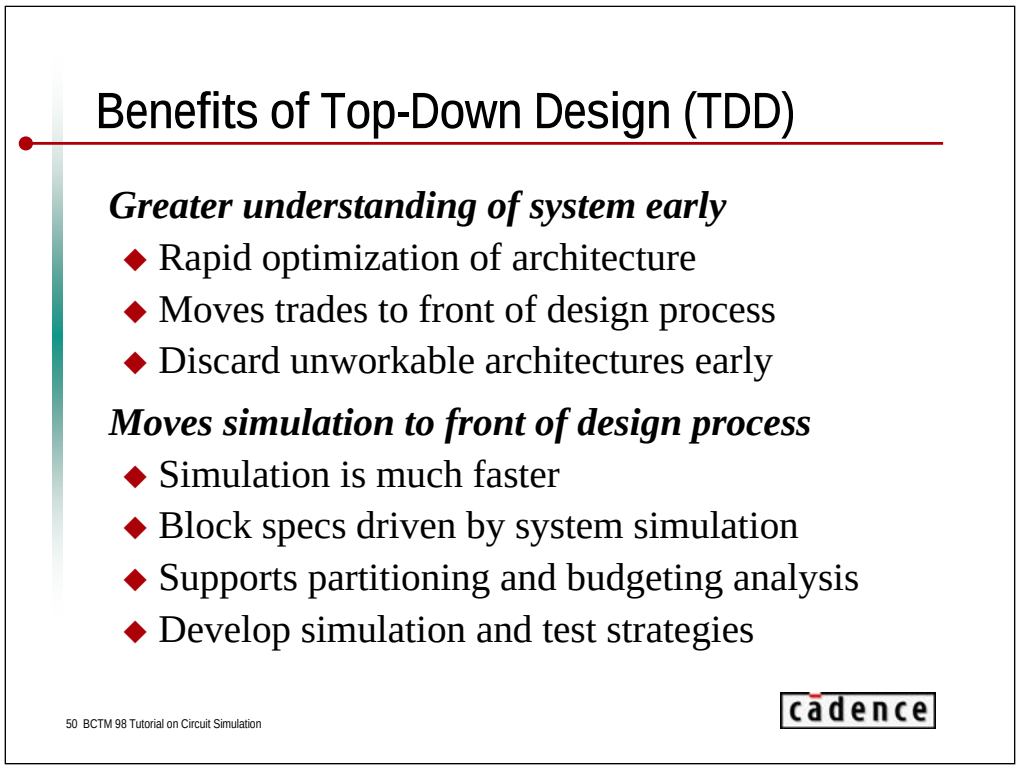
At DAC this year, Ron Collett of Collett International presented study of 21 chip designs from 14 leading semiconductor companies that showed that the relative productivity of the top company was 14 times that of the least productive. Productivity was measured in terms of normalized transistors designed in per person per week. Productivity is most affected by three factors, the people, the methodology, and the tools. Optimal productivity is achieved when all three of these factors are working together to support each other. Thus, the best tool is not necessarily be the fastest tool. Instead, it is the tool that best fits into and supports the methodology. And the best methodology is not the one that reduces the time required for a particular step in the design, it is the one that reduces the time to complete the entire design. To significantly improve productivity, it may be necessary to change both the design methodology and the tools, and then train the people to use both such that they achieve maximum benefit.

With proper methodology and tools, it is currently possible to consistently produce complex mixed-signal chips in 3 months with no respins. This is the level of productivity of Cadence's mixed-signal design center, which uses a top-down design methodology built on Cadence's mixed-signal design tool suite.



The Secret to High Productivity

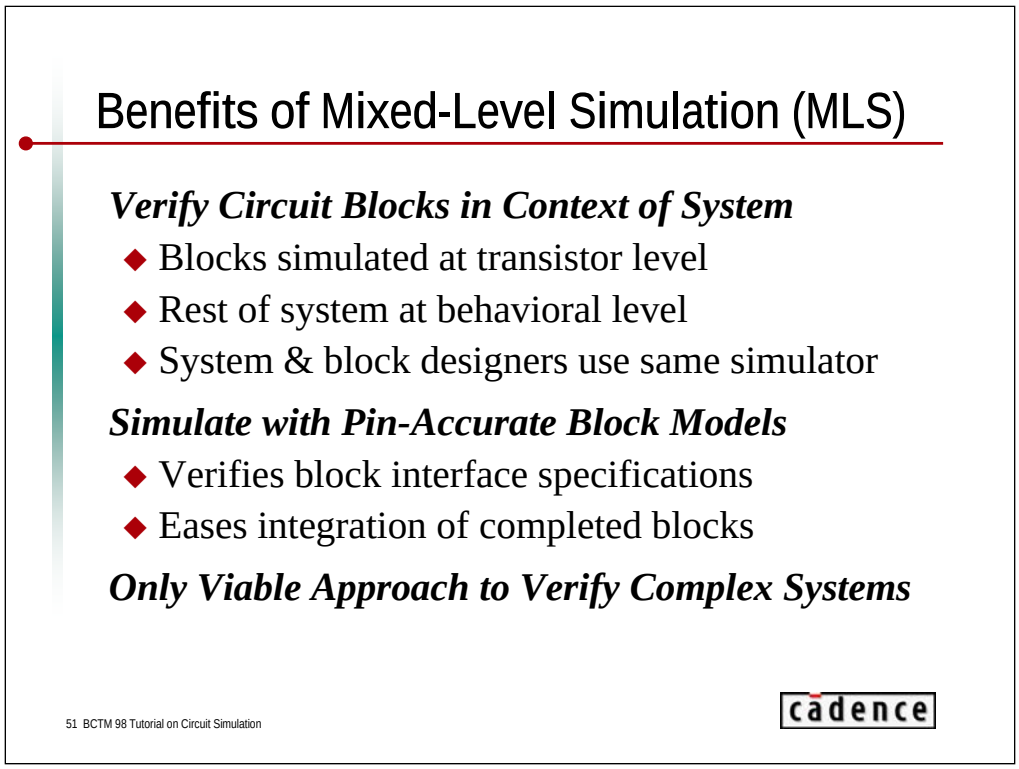
With the increasing complexity of designs, it is becoming extremely important to move to a top-down design methodology. In top-down design, more time is spent exploring, designing, and verifying the system at the architectural level before the detailed design of the individual blocks is begun. However another important aspect is mixed-level simulation, which provides the infrastructure to to support the design process through out the entire life of the design. Mixed-level simulation is the ability to support both high-level modeling and simulation as well as transistor- or SPICE-level simulation simultaneously. During the design process, only high-level simulation is used until the design progresses to the point where the partitioning into blocks is complete, and both the behavior and the interface of each block is defined. At this point, the architecture would be described with interface-accurate block models. At this point the mixed-level simulation becomes critical. As the blocks are designed to the transistor level, they can be substituted for the corresponding behavioral model and simulated with the rest of the system. This allows the block to be verified in the context of the whole system. This approach allows a design to be partitioned into blocks with each block being designed in parallel with confidence that the blocks will work properly when they are integrated together.



Benefits of Top-Down Design (TDD)

The basic principle of top-down design is to design and verify the system at an abstract level before beginning design at the next level down. It is appropriate whenever there is sufficient complexity at the system level. Employing top-down design gives designers a better understanding of the system and allow them to make design tradeoffs in the system architecture early in the design process. Thus, it reduces the chance that blocks will have to be redesigned because they were originally designed with incorrect assumptions. It also naturally supports concurrent design once the system has been specified because the block designers can work relatively autonomously and the system engineer can work to develop the simulation and test plan.

An important step in the top-down design process it to take the system architecture to the point where the details of the interfaces between the blocks are specified and verified. This means that system simulation should be performed with pin-accurate block models. This helps guide the structure of the design and verifies the interfaces between the blocks. It is also a the first step towards mixed-level simulation which will allow the blocks to be verified in the context of the system.

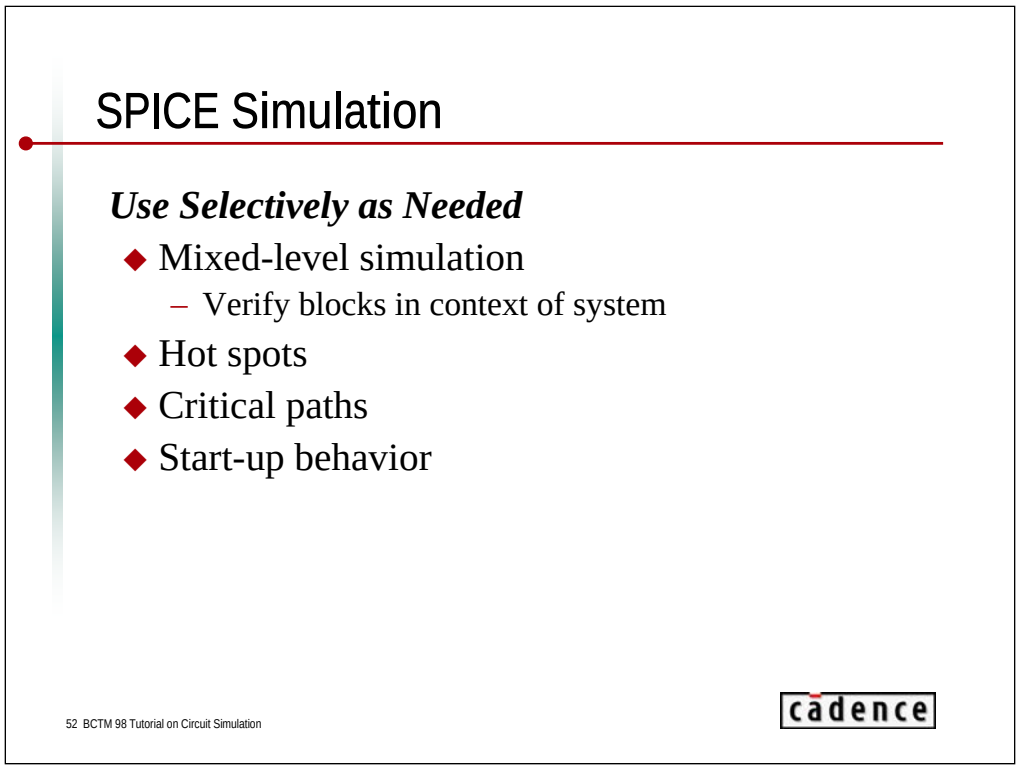


Mixed-Level Simulation

Mixed-level simulation is used during top-down design to verify large complex mixed-signal systems, and it is the only feasible approach currently available. Some propose to use either timing simulators (sometimes referred to as fast or reduced accuracy circuit simulators) or real circuit simulators running on parallel processors. However, both approaches defer system-level verification until the whole system is available at transistor level, and neither provide the performance nor the generality needed to verify most mixed-signal systems.

In mixed-level simulation, the system, described at a high level, acts as a test-bench for the block, which is described at the transistor level. Thus, the block is verified in the context of the system, and it is easy to see the effect of imperfections in the block on the performance of the system. Mixed-level simulation requires that both the system and the block designers use the same simulator and that it be well suited for both system- and transistor-level simulation.

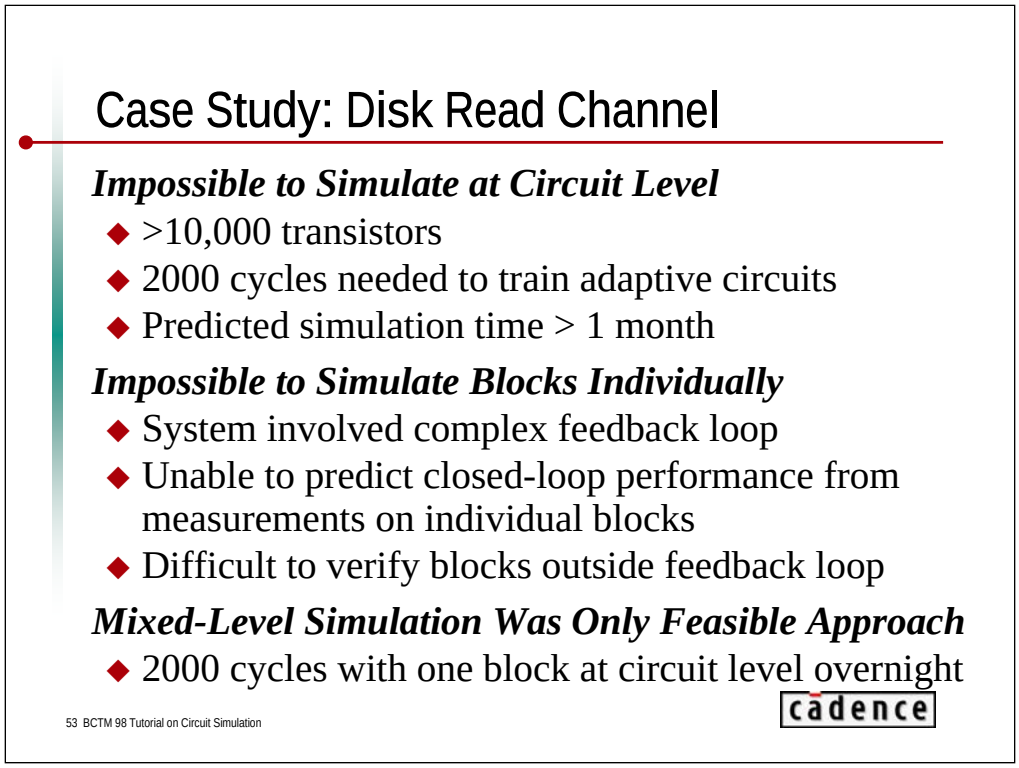
Mixed-level simulation allows a natural sharing of information between the system and block designers. When the system level model is passed to the block designer, the the behavioral model of a block becomes an an executable specification and the description of the system becomes an executable test bench for the block. When the transistor level design of the block is complete, it is easily included in the system level simulation by the system designer.



SPICE-Level Simulation

Circuits are getting more complex in two different ways at the same time. First, circuits are becoming larger. Consider wireless products. 30 years ago a typical transceiver contained between 5 and 10 transistors whereas it is common for a modern cell phone to contain between 1M and 10M transistors. Second, the operation of the circuits are becoming more complex. 20 years ago integrated circuits generally consisted of simple functional blocks such as opamps and gates. Verification typically required simulating the block for two or three cycles. Today, mixed-signal chips implement complex algorithms that require simulations to last thousands of cycles. Examples include PLLs, ΣΔ converters, and magnetic storage PRML channels. Simulation complexity is increasing at a blistering pace, and verifying modern complex circuits completely at the transistor-level simply takes too much time, even for 'fast' simulators such as reduced accuracy simulators or those that employ parallel processing.

In a top-down design process, SPICE-level simulation is used judiciously in order to get the benefits without incurring the costs. All blocks are simulated at the transistor level in the context of the system in order to verify its functionality and interface. Hot spots, or areas of special concern, are identified up front and simulated at the transistor level. The performance of the circuit is verified by simulating just the signal path or key pieces of it at the transistor level. Finally, if start-up behavior is a concern, it is also simulated at the transistor level.

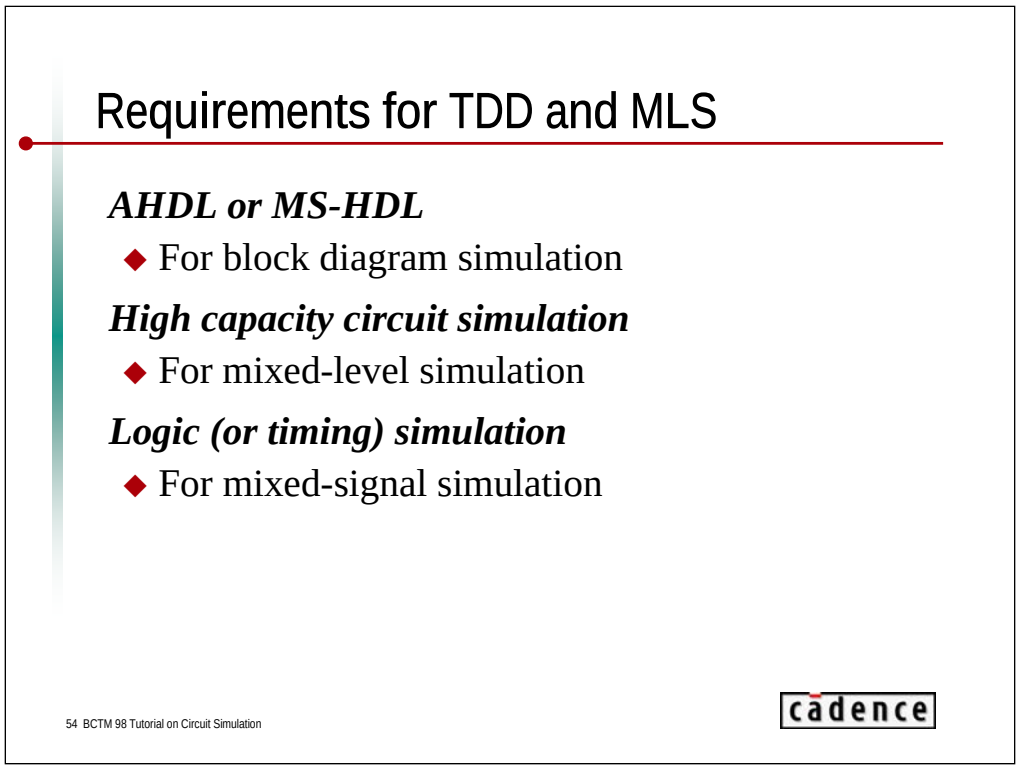


Case Study: Disk Read Channel

Though this example is several years old, it is representative of the type of circuit complexity that is becoming mainstream today. It is a PRML channel chip that it difficult to simulate for two reasons. First, it is a relatively large circuit that involves both analog and digital sections that are closely coupled. Second, the architecture involves complex feedback loops and adaptive circuits and it takes many many cycles for these circuits to settle. The combination of many transistors and many cycles combines with the result being a simulation that is so expensive as to be impractical. In this case, the expected simulation time was predicted to be greater than a month.

The traditional approach to simulating a complex circuit like this would be to simulate the blocks individually. Of course this verifies that the blocks work individually, but not together. In addition, for this circuit it is difficult to verify the blocks when outside the system, and it is difficult to predict the performance of the system just knowing the performance of the individual blocks.

When the architecture was simulated at a high level with each block represented by a pin-accurate behavioral model, the simulation time was less than 10 minutes. Then, when a single block was run at the transistor level, the simulation ran overnight. And even though the full system was never simulated at the transistor level, it worked first time because this methodology does verify the blocks in the context of the system and it verifies the interfaces between the blocks.



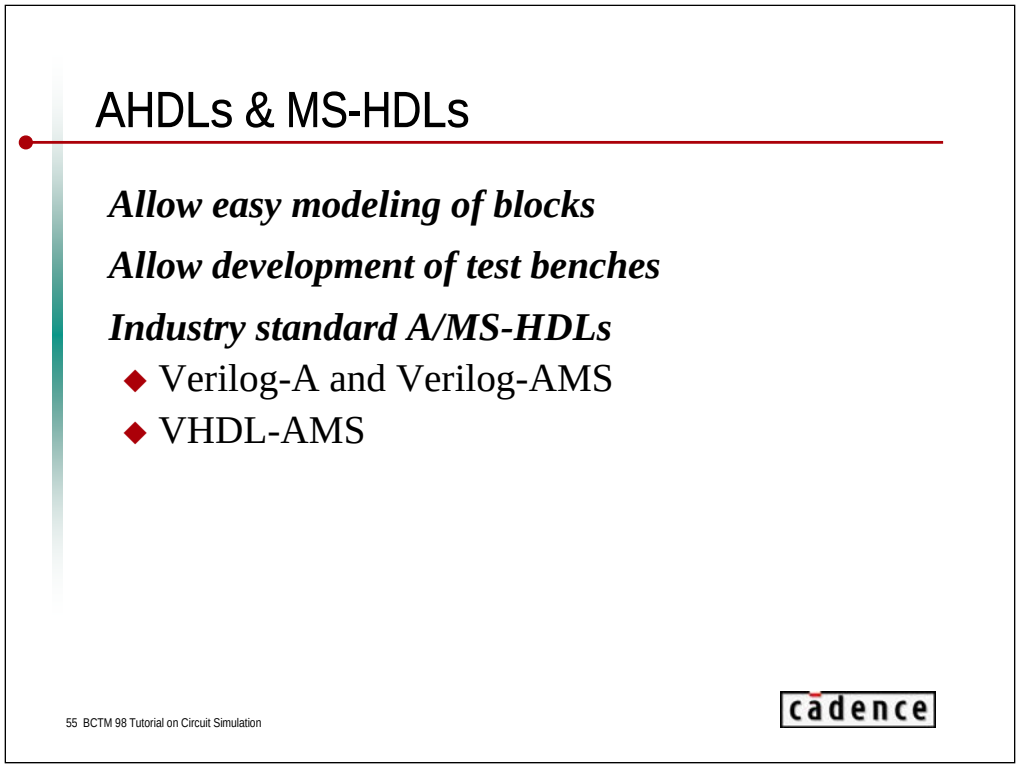
Requirements for Top-Down Design and Mixed-Level Simulation

To support top-down design and mixed-level simulation of mixed-signal circuits requires a simulator that provides an AHDL or MS-HDL, high capacity SPICE simulation, and logic simulation.

An AHDL or MS-HDL is needed to support top-down design and mixed-level simulation. Top-down design improves the designers' ability to efficiently design complex circuits while mixed-level simulation improves the designers' ability to verify complex circuits.

High capacity high performance SPICE simulation is required in order to support mixed-level simulation. Without high capacity SPICE simulation the size of the blocks simulated at transistor level can be limited, which forces an inappropriate or inconvientent partitioning of the blocks.

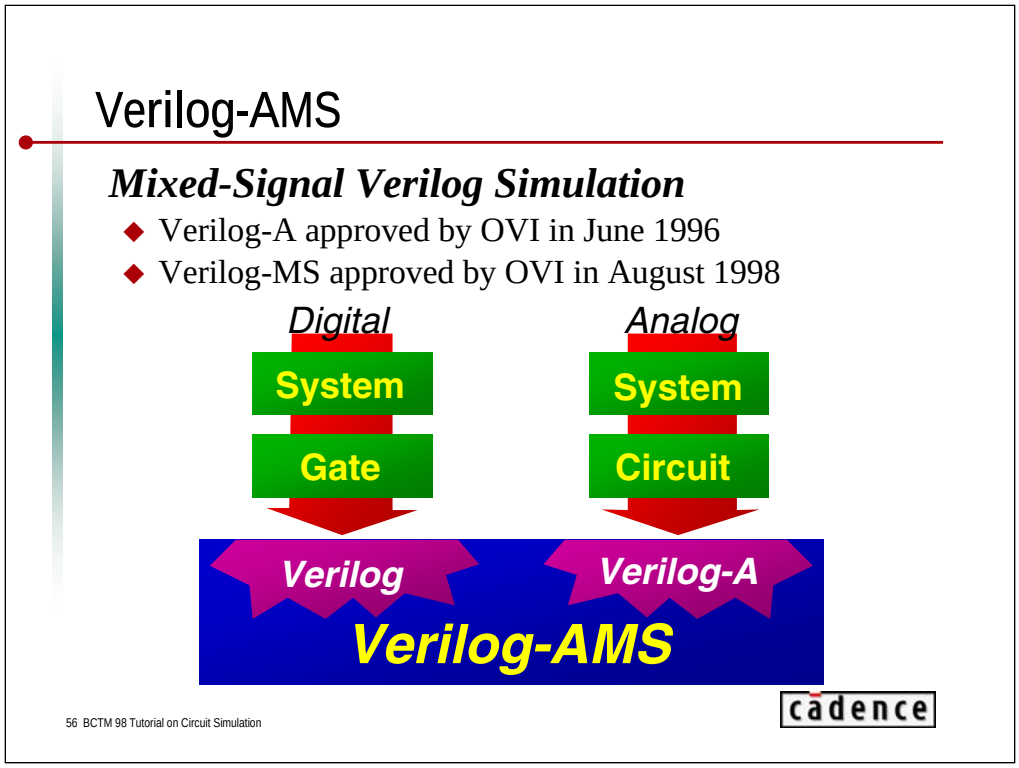
And of course, logic simulation is required to support mixed-signal simulation.



AHDLs and MS-HDLs

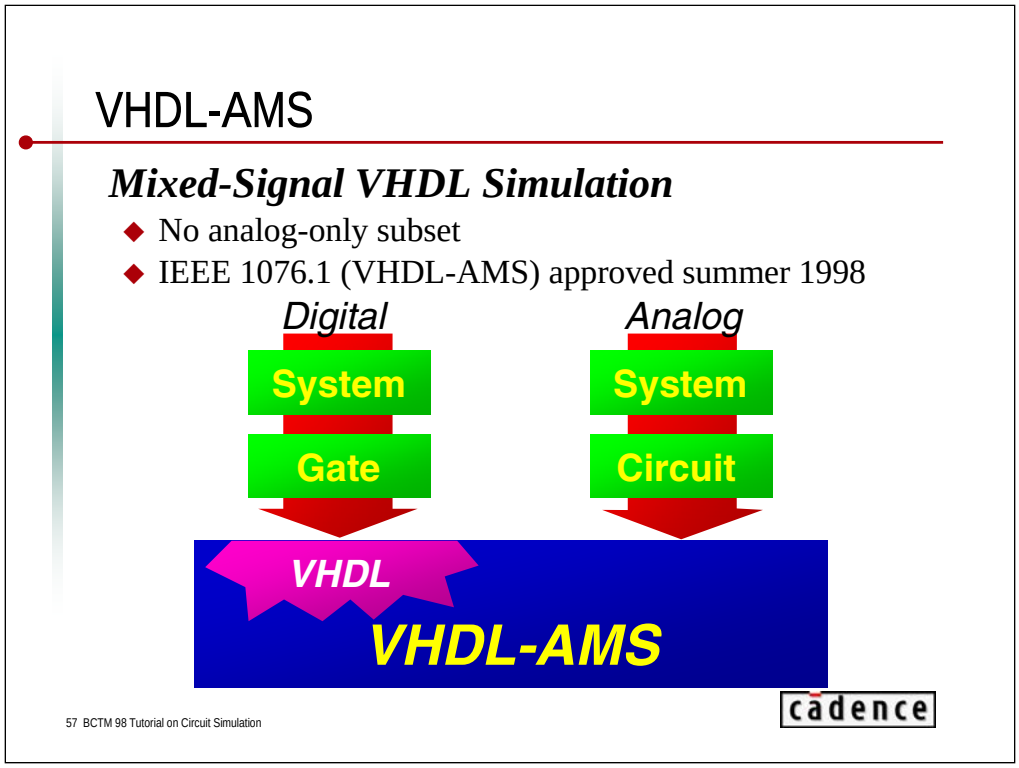
AHDLs and MS-HDLs are used to model many different types of systems, as well as use it for applications such as complex test fixture modeling. They can also provide some means of IP security, as companies can hand designs to outside sources without disclosing their transistor level data.

Recently analog and mixed-signal extensions have been developed for both Verilog and VHDL.



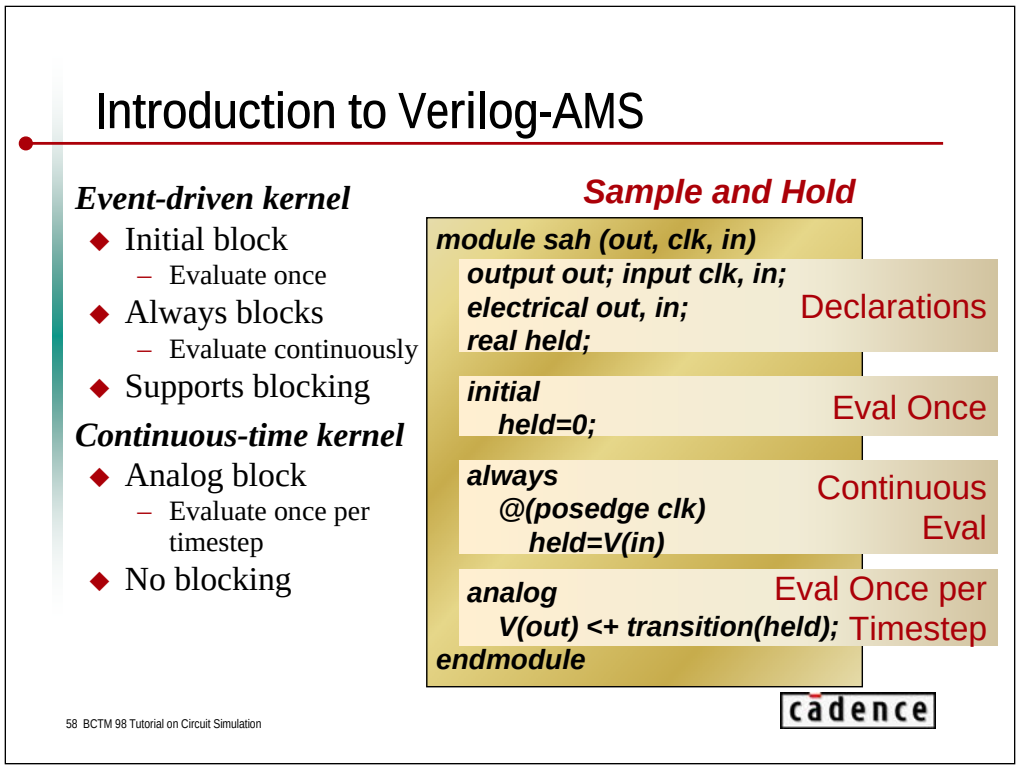
Verilog-AMS

Verilog-AMS is analog and mixed-signal extensions to Verilog-HDL. Verilog-HDL is a digital event-driven simulator. As a first step, Verilog-A was developed as an analog only language. It supports continuous time simulation. It is has a syntax patterned after Verilog-HDL, but is completely distinct. Several commercial versions of Verilog-A are currently available. Verilog-AMS is the merge of Verilog-HDL and Verilog-A. It is a single language that supports the description of both analog and digital circuits and behavior. Verilog-AMS also provides additional capability not found in either Verilog-HDL or Verilog-A. In particular, Verilog-AMS adds event-driven analog, automatic interface element insertion, and automatic back-annotation of parasitics.



VHDL-AMS

VHDL-AMS is analog and mixed-signal extensions to VHDL. VHDL is a digital event-driven simulator. Unlike with Verilog, there is no analog-only subset for VHDL. VHDL-AMS is the addition of analog capabilities to VHDL. It is a single language that supports the description of both analog and digital circuits and behavior. Unlike Verilog-AMS, VHDL-AMS does not provide automatic interface element insertion or automatic back-annotation of parasitics.



Introduction to Verilog-AMS

Verilog-AMS descriptions support both event-driven and continuous-time behavior. Event-driven behavior is given in the *initial* and *always* blocks. Initial blocks are executed only once, whereas always blocks are evaluated repeatedly. Certain statements in these blocks are blocking statements. That means that execution of the block is suspended until the statement finishes.

Continuous-time behavior is given in the *analog* block. Analog blocks are evaluated once per time-step. In the analog block, statements do not block.

Signal-Flow and Conservative Models

Signal-Flow Model

- Model relates potentials only
- Useful for abstract models
  - Top-level models in top-down design

Conservative Models

- Model relates potentials and flows
- Device modeling and loading at interfaces

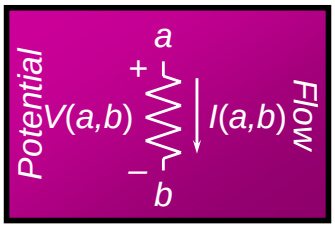
Both are Compatible in Verilog-AMS

- Freely interconnect
- Written in same style

cadence

Potentials and Flows

Resistor

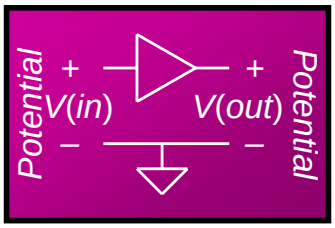



```
module resistor (a, b)
   electrical a, b;
   parameter r = 1;

   analog
      V(a,b) <+ r*I(a,b);
endmodule
```

Conservative Model (potential & flow)

Amplifier




```
module amp (out, in)
   output out; input in;
   voltage out, in;
   parameter a = 1;

   analog
      V(out) <+ a*V(in);
endmodule
```

Signal-Flow Model (potential only)

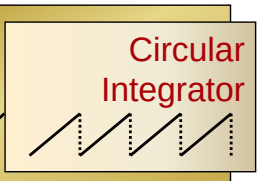
Analog Operators

- Time integration, differentiation, & delay
- Laplace & Z filters
- Transition & slew filters

VCO

```
module vco (out, in)
   voltage out, in;
   parameter k = 1;
   real phase;

   analog begin
      phase = idtmod(k*V(in), 0, 1);
      V(out) <+ cos(2*M_PI*phase);
   end
endmodule
```



cadence

Variables and Parameters

Variables retain their value

- Can be used as state variables

Parameters support range limits

- User specified upper and lower bounds
- Inclusive or exclusive bounds
- Bounds may be functions of other parameters

Types

- Reals, Integers, Strings

cadence

Events

Events

Name	*Generates events at …*
digital signal	At signal transitions
cross()	At analog signal crossings
timer()	Periodically or at specific times
initial_step	At beginning of simulation
final_step	At end of simulation

@ blocks

- Blocks of code executed upon an event

cadence

Example: Sampler

Sampler

```
module sampler (out, in)
   voltage out, in;
   output out; input in;
   parameter Tstart = 0;
   parameter Tp = 1 from (0:inf);
   parameter tt = T/10 from [0:T];
   real hold;

   analog begin
      @(initial_step or timer(Tstart, T))
         hold = V(in);

      V(out) <+ transition(hold, 0, tt);
   end
endmodule
```

Parameters with Limits

State Variable

Event Block

cadence

Mixed-Signal Simulation

Behavior MS Modeling

- Can read analog signals in digital block
- Can read digital signals in analog block
- Generate events from analog signals using *cross*()

Structural MS Modeling

- Analog/digital port type mismatches
  - Automatically inserts user-specified interface module to resolve mismatch
- Signal-flow/conservative ports are compatible

cadence

Other Capabilities

Multidisciplinary modeling

- Modeling non-electrical systems
- User defined disciplines

Ideal switch modeling

Relay

```
module relay (a, b, closed)
   electrical a, b;
   input closed;

   analog begin
      @(closed) discontinuity(0);
      if (closed) V(a,b) <+ 0;
   end
endmodule
```

Switch Branch

cadence

Example: Phase/Frequency Detector

```
module pfd_cp (out, ref, vco)
   electrical out;
   output out; input ref, vco;
   parameter Iout = 100u;
   integer state;

   always begin
      @(posedge ref)
         if (state > –1)  state = state – 1;
      @(posedge vco)
         if (state < 1)  state = state + 1;
   end

   analog
      I(out) <+ transition(Iout*state);
endmodule
```


cadence

Example: N-Bit Analog-to-Digital Converter

```
module adc (out, in, conv)
   parameter bits=8, Vmax=1.0;
   voltage in; input in, conv; output [0:bits-1] out;
   real sample; integer I;

   always begin
      @(posedge conv) begin
         sample = 2*V(in);
         for (I=1; I <= bits; I=I+1) begin
            out[I] = (sample > Vmax);
            if (sample > Vmax)
               sample = sample – Vmax;
            sample = 2*sample;
         end
      end
   end
endmodule
```

Outline

Traditional SPICE Simulation

- DC
- AC and Noise
- Transient
- Fourier

Timing Simulation

AHDLs and MSHDLs

RF Simulation (tomorrow) ←

cadence