Xyce: an open source SPICE engine

Eric Keiter, Jason Verley, Tom Russo, Heidi Thornquist, Sivasankaran Rajamanickam, Ting Mei, Rich Schiek, Pete Sholander, Dave Baur
Sandia National Laboratories

Jaijeet Roychowdhury, Tianshi Wang
UC-Berkeley

MOS-AK Workshop
Berkeley, CA
Dec. 12, 2014
Outline

Xyce intro
Xyce requirements
Xyce design
  Object-oriented (C++)
  Spice-compatible
  But different than SPICE!
  Parallel implementation
  Analysis types (DC, Tran, AC, HB, MPDE, UQ)
  Open-Source (as of v6.0)

ADMS model compiler
ModSpec-Xyce
Xyce supports most SPICE models (BSIM, EKV, etc)
Parallel Circuit Simulator

- Xyce: Massively Parallel circuit simulator:
  - Distributed Memory Parallel (MPI-based)
  - Unique solver algorithms
  - SPICE “Compatible”
  - Industry standard models (BSIM, PSP, EKV, VBIC, etc)
  - ADMS model compiler

- Analysis types
  - DC, TRAN, AC
  - Harmonic Balance (HB)
  - Multi-time PDE (MPDE)
  - Model order reduction (MOR)
  - Direct and Adjoint sensitivity analysis
  - Uncertainty quantification (UQ) via Dakota.

- Sandia-specific models
  - Prompt Photocurrent
  - Prompt Neutron
  - Thermal

- Other, non-traditional models
  - Neuron/synapse
  - Reaction network
  - TCAD (PDE-based)

- Xyce Release 6.2
  - Open Source!
  - GPL v3 license

http://xyce.sandia.gov

Next release (v6.3) ~May 2015
Project Motivation: Why Xyce?

- Comprehensive Test Ban Treaty (CTBT), 1993
- Advanced Simulation & Computing (ASC), 1995
- Qualification Alternatives to SPR (QASPR), 2005
  - Offset lack of NW testing
  - Help qualify NW systems

- Unique Requirements ➔ Differentiating capabilities
  - Unique models: Radiation Effects
  - High fidelity: SPICE-level or higher
  - Large capacity: Massively-parallel
  - IP: Sandia owns it

- Xyce is not an acronym
  - Rhymes with SPICE
  - Sounds cool.
Why Sandia?

Rich history in the development and maturation of high performance computing hardware and software technology

Gordon Bell Prize

R&D 100
Parallel Software

R&D 100
Dense Solvers

Gordon Bell Prize

R&D 100
Storage

Gordon Bell Prize

R&D 100
Signal Processing

World Record Teraflops

World Record
281 GFlops

World Record
143 GFlops

Patent
Meshing

Patent
Paving

Patent
Meshing

Patent
Decomposition

Karp Challenge

R&D 100
3D-Touch

R&D 100
Fernbach
Award

SC96 Gold Medal

Data Mining

Networking

Patent

Mannheim

SuParCup

R&D 100
Trilinos R&D 100
Xyce

R&D 100
Allocator

R&D 100
Aztec

R&D 100
Salvo

Patent

Burton

Aztec

Patent

Decomposition

R&D 100

3D-Touch

Eric Keiter, Sandia National Laboratories
2014 NEEDS Workshop, Berkeley, CA
Why Open Source?

- First open source release, v6.0
- November 5, 2013.
- GPL license v3.0
- Source and binary downloads available
  - xyce.sandia.gov

- Foster external collaboration
- Feedback from wider community
- Taxpayer funded, so encouraged to open source.
Open-Source Capabilities

- Numerical libraries have been open-source for years:
  - Trilinos: trilinos.sandia.gov
  - Dakota: dakota.sandia.gov

- Xyce open-source is new:
  - Xyce: xyce.sandia.gov

Dakota capabilities:
- LHS sampling (parallel)
- Importance sampling
- Failure analysis methods
- Stochastic collocation
- Polynomial chaos
- Model calibration/extraction
- Gradient-based optimization

Optimization and Uncertainty Quantification (UQ) Toolbox

Circuit Simulator Application

Solver Library

Eric Keiter, Sandia National Laboratories
2014 MOS-AK Workshop, Berkeley, CA
Open Source Releases

Release every 6 months.

Xyce v6.0 (first open source) October, 2013

Xyce v6.1 April 2014
- Parallel HB, MPDE and AC analysis.
- BSIM-CMG model
- New digital models
- Dynamically linkable devices (beta implementation)

Xyce v6.2 October, 2014
- Calculation of transient direct sensitivities.
- Support for lead current calculation and output for Harmonic Balance (HB) analysis.
- Improved interpolation for both the Trapezoid and Gear time integrators.
- Improved results output (.PRINT) capabilities.
- New models for a lumped transmission line and a digital latch.
- Fixes for nearly 60 bugs and enhancement requests.

Xyce v 6.3 ~May 2015
Near term new features

Some of these will be in 6.3, some 6.4.

Xyce v6.3 ~May 2015
- Multi-tone HB (unlimited # of tones)
- Small-signal noise analysis
- Auto-detection/application of MOR to linear subcircuits
- BSIM6 model
- MEXTRAM model

Xyce v6.4 ~October/November 2015
- Adjoint transient parameter sensitivities
- New direct linear solver (BASKER*) [alternative to KLU]
  - serial version (completed)
  - threaded version (in progress, done by fall 2015)

BASKER = BASic Sparse KERnels
Longer term activity

Releases TBD
still research phase

Flexible support for threading
(Most of Xyce’s parallelism based on message-passing)
Need to accommodate modern computing architectures
Cuda, TBB, etc
Kokkos/Tpetra and other threaded-enabled parts of trilinos.

Intrusive optimization and UQ methods
ROL ("Rapid Optimization Library") library from Trilinos
Dakota
Xyce History of Linear Solvers

- Initially (circa 1999), Xyce used available PDE-based preconditioning techniques
  - Incomplete LU factorization
  - Limited scaling / robustness
- For small scale circuits, the Dulmage-Mendelsohn permutation (BTF) was leveraged in KLU (2004)
- In 2008, BTF structure was leveraged to create a new preconditioned iterative method
  - Great for older CMOS memory circuits
  - Circuits with parasitics are more challenging
  - Modern MOS models (w/gate leakage) break it.
- In 2010, initial development of ShyLU, a “hybrid-hybrid” sparse linear solver package
  - Improve robustness

<table>
<thead>
<tr>
<th>Preconditioning Method</th>
<th>Residual</th>
<th>GMRES iters</th>
<th>Solver Time (s.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local AMD ILUT</td>
<td>3.43e-01</td>
<td>500</td>
<td>302.573</td>
</tr>
<tr>
<td>ParMETIS</td>
<td>(FAIL)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BTF</td>
<td>3.47e-10</td>
<td>3</td>
<td>0.139</td>
</tr>
<tr>
<td>Block Jacobi Hypergraph</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

W. Bomhof and H.A. van der Vorst [NLAA, 2000]

26x speedup on 16 cores
Hybrid-Hybrid solver result: 19x Speedup for Challenging IC

Necessary for efficient simulation of a CMOS logic circuit:
1.6M total devices, ~2M unknowns
Xyce w/ KLU solver takes ~ 2 weeks, w/ ShyLU solver takes ~ 1 day

TABLE III
COMPARISON OF TOTAL LINEAR SOLVE TIME (SEC.) OF VARIOUS SPARSE DIRECT SOLVERS FOR OUR TEST CIRCUITS; (-) INDICATES SIMULATION FAILED TO COMPLETE.

<table>
<thead>
<tr>
<th></th>
<th>ckt1</th>
<th>ckt2</th>
<th>ckt3</th>
<th>ckt4</th>
<th>ckt5</th>
</tr>
</thead>
<tbody>
<tr>
<td>KLU</td>
<td>80.8</td>
<td>162.2</td>
<td>9381.3</td>
<td>7060.8</td>
<td>14222.7</td>
</tr>
<tr>
<td>PARDISO (16)</td>
<td>128.6</td>
<td>105.3</td>
<td>715.0</td>
<td>6690.5</td>
<td>-</td>
</tr>
<tr>
<td>SuperLU</td>
<td>-</td>
<td>10294.1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SuperLU_Dist (16)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Strong scaling of Xyce's simulation time and ShyLU linear solve time for different configurations of MPI Tasks X Threads per node.

Why not use SPICE?

SPICE: the original open-source simulator

- de-facto standard
- However: difficult to modify legacy serial code to use message passing

More generally, to be useful: modular, well-structured, flexible

- separated numerics, algorithms, models, I/O
- simple, clean interfaces
- short, easy to read, easy to modify
```c
#define SENSDEBUG
    printf("vd = \%7e \n", vd);
#endif /* SENSDEBUG */
    goto next1;

} else if (ckt->CKTmode & MODEINITJCT) &&
    (ckt->CKTmode & MODETRANOP)
    && (ckt->CKTmode & MODEUIC) ) { 
    vd=here->DIOinitCond;
} else if ( (ckt->CKTmode & MODEINITJCT) && here->DIOoff) { 
    vd=0;
} else if ( ckt->CKTmode & MODEINITJCT) { 
    vd=here->DIOtvCrit;
} else if ( ckt->CKTmode & MODEINITFIX && here->DIOoff) { 
    vd=0;
} else {
    #ifndef PREDICTOR
    if (ckt->CKTmode & MODEINITPRED) {
```
Why Modularity is Necessary

Key to managing complexity

• limits what each developer needs to know about the system
• the “API” encapsulates rest of the simulator
• Eg, to implement BSIM6.4, you don't need to know all about:
  • Parallel implementation
  • trapezoidal, Gear, etc. algorithms
  • shooting, harmonic balance, sensitivity, ...
  • Newton-Raphson, homotopy, ...

Errors and mistakes reduced
Analysis Algorithms: Structure

- dense & sparse matrix solution routines
- Newton-Raphson and Homotopy
- integration methods (BE, TRAP, Gear, etc)
- Krylov subspace, TBR methods; eigenanalysis, SVD routines

(arrows mean “needed for”)

DC

AC

stationary noise

transient

2, 3, etc. time scale multi-time/envelope, including autonomous (time/freq domain formulations)

One-tone steady state including autonomous

Multi-tone steady state including autonomous

Cyclostationary/phase noise

LTV/nonlinear macromodelling algorithms, including autonomous

- 2014 MOS-AK Workshop, Berkeley, CA
Xyce device hierarchy

- Object-oriented design
- Devices derived from base objects
- Hidden from devices:
  - Parallelism details
  - Analysis details
- Each device responsible for providing $f(x,t)$ and $q(x,t)$ terms

$$\frac{d}{dt} q(\vec{x}(t)) + f(\vec{x}(t), \vec{u}(t)) = \vec{0}$$
Xyce Analysis classes

With proper code design, analysis types *should* be independent of device models

Reverse should also be true!

Main difference between each of these is how $f$ and $q$ are handled.

$$
\frac{d}{dt} q(\vec{x}(t)) + f(\vec{x}(t), \vec{u}(t)) = \vec{0}
$$
Harmonic Balance

- Expands state variables as a Fourier series; solves for the Fourier coefficients
  -Insensitive to widely spaced spectral components
  -Excellent for dealing with complex high-frequency passive (linear) components
  -Directly captures the large-signal quasi-periodic steady-state
- Standard set of circuit equations:
  \[ g(x(t)) + \frac{d}{dt} q(x(t)) = u(t) \]

  Approximate solution with Fourier expansion:
  \[ x(t) = \sum_{h=-M}^{M} X_h \exp(j\omega_h t) \]

  Frequency domain equation:
  \[ G(X) + \Omega Q(X) = U \]

- Block linear system:
  - N=original system size
  - M=number of Fourier terms
  - Total size = N*M

- New for Xyce v6.1: Parallel HB
- New for Xyce v6.3:
  - Multi-tone HB
  - Unlimited number of tones
  - Various stability enhancements
DCOP Non-linear solution techniques

- DCOP solution can be difficult
  - No initial guess!
- Some circuits have multiple DC solns.
- Need to find stable solution.
- DCOP is initial condition to most analysis methods:
  - TRAN, AC, HB, etc

- Homotopy:
  - Relax stiffness
  - continuation params
  - Obtain solution via continuation

- Common examples (Xyce & SPICE)
  - Voltage source stepping
  - GMIN stepping

- Less common (Xyce):
  - Pseudo-transient
  - MOSFET homotopy (see right)

MOSFET continuation example

Model Order Reduction (MOR)

- Model reduction: in a rigorous manner, generate a system of the same form, but smaller dimension, with input-output behavior approximately the same.
- Very important for large linear networks, such as post-layout parasitics.
- Several projection methods are implemented in Xyce.
- Projection squashes matrices to smaller size

\[
Q^T \quad A \quad Q \quad \rightarrow \quad \hat{A}
\]

- Several MOR methods for linear systems implemented in Xyce
- New auto-detection code being implemented to automatically detect and reduce large linear subcircuits. (should be in Xyce v6.3)
Uncertainty Quantification (UQ)

- Xyce has a number of capabilities to support uncertainty quantification (UQ)
- Mostly these come via the DAKOTA framework.

- General idea: given uncertainty on parameter inputs, how to estimate that on circuit outputs.
- Most simulators support “brute force” approaches based on sampling.
- Many much more sophisticated methods exist, including stochastic collocation methods, etc.

Eric Keiter, Sandia National Laboratories
2014 MOS-AK Workshop, Berkeley, CA
Transient Direct Sensitivities

- New in Xyce v6.2
- Solves for \( \frac{dO}{dp} \), where \( O \)=objective function, \( p \)=parameter.
- Can be applied to *any* device model parameter.
- In general, direct form solves

\[
\frac{dO}{dp} = \frac{dO}{dx} \left[ \left( \frac{dF}{dx} \right)^{-1} \frac{dF}{dp} \right] + \frac{dO}{dp}
\]

- For transient, the linear system being solved is specific to the integration method. For backward Euler, this gives:

\[
\left[ \frac{1}{h} \frac{dq}{dx} + \frac{dj}{dx} \right] \frac{dx}{dp_{n+1}} = -\frac{1}{h} \left[ \frac{dq}{dp_{n+1}} - \frac{dq}{dp_n} \right] - \frac{dj}{dp} + \frac{db}{dp} + \frac{1}{h} \left[ \frac{dq}{dx} \right] \frac{dx}{dp_n}
\]

- The RHS \( dq/dp, dj/dp, db/dp \) terms come from **compact device models**.
- Xyce will compute these automatically; method depends on device implementation
  - Hand-coded derivatives in simple devices (resistors, etc)
  - Automatic-differentiation (Sacado) in devices that have been instrumented for it
  - Finite difference derivatives if neither are available.

- **ADMS** Back end currently doesn’t instrument for AD parameter derivatives, but this is planned for this year.
Transient Direct Sensitivities

Example circuit.
Simple diode, driven by sinewave source.

```
 transient diode circuit sensitivity calculation
 R 1 2 0.0001
 V2 2 0 0.0 SIN(0 5 100K)
 V1 3 0 0.0

 D2 1 3 DZR
 .MODEL DZR D( level=2 IS = 1E-14 RS = 10.8 N = 1 TT = 0
 + CJO = 1P VJ = 1 M = .5 EG = 1.11
 + XTI = 3 KF = 0 AF = 1 FC = .5
 + BV = 7.255 IBV = .001 tbv1 = 0.00013 tbv2 = -5e-8 )

 .SENS objfunc=(I(V1))

 .options SENSITIVITY adjoint=0 direct=1
 .options timeint method=gear
 .TRAN 0 2e-5
 .print tran v(2) I(V1)
 .print sens v(2)
 .options device temp=25
 .END
```

Figure 4.9. Transient Diode Circuit

Input Voltage
Finite Difference
Transient Direct

<table>
<thead>
<tr>
<th>V(2A)</th>
<th>dI/d(VJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1E-06</td>
<td>-2E-06</td>
</tr>
<tr>
<td>2E-06</td>
<td>0</td>
</tr>
<tr>
<td>3E-06</td>
<td>2E-06</td>
</tr>
<tr>
<td>4E-06</td>
<td>0</td>
</tr>
<tr>
<td>5E-06</td>
<td>-2E-06</td>
</tr>
</tbody>
</table>

Figure 4.10. Transient Diode Sensitivity Example Netlist
ADMS-Xyce

Verilog-A interface, via ADMS model compiler

- VBIC, Mextram, EKV, HiCUM, etc.

Verilog-A: industry standard format for new models

ADMS translates Verilog-A to compilable C/C++ code;
API automatically handles data structures, matrices, tedious details.

```
VBIC TRANSISTOR
in
VERILOG-A

Run admsXyce

Ready-to-compile C++ code

ADMSvbic.h
ADMSvbic.C
```
ADMS-Xyce

ADMS-converted models in Xyce:

- VBIC (the first model we did)
- EKV (can't distribute thanks to NDA, so it's not in Xyce open-source)
- PSP (version 103)
- BSIM-CMG 107 (only one of the many variants)
- FBH HBT_X version 2.1
- FBH HBT_X version 2.3

- BSIM6 (development branch)
- MEXTRAM (pending)
- MVS  Silicon virtual source model
What is unique about Xyce's use of ADMS?

- Sacado AD library: By using Sacado, we are able to do away with an enormous amount of "admst" code for computing derivatives (compare with ng-spice's or qucs ADMS back-ends). It is difficult to overstate how much back-end work this saves.

- Have added "$limit" support (simple mod to ADMS required to let "$limit" be recognized as a legal function, but otherwise all the work is in the back-end).

- Had to add some "attributes" for parameters and modules to provide certain metadata (descriptions, units, "model" vs. "instance", level number, whether it's a Q, M, Y device, etc.).
ADMS Challenges

- ADMS back-ends are written in ADMST, a sort of XSLT superset.
- Xyce's linear system differs from SPICE so generating code for voltage limiting ($\text{limit}$) is not as simple as generating a function call.
- ADMS has NO support for current branches. It generates datastructures representing the Jacobian resulting when all variables are nodal voltages, but you must do all the work yourself if you throw a branch current in the mix.
- Node collapse: since this has to happen at the time the device is instantiated (rather than the time its equations are evaluated), it is necessary to generate (separately from the rest of the evaluation) the parts of code needed to compute all the variables required to determine whether or not to collapse.
- Need to extend ADMS back-end to produce parameter sensitivities with Sacado.
- Dynamic linking: this has been done, but is still immature.
ADMS Challenges

- not well suited for multi-physics domains
  - even for electrical domain, doesn't support structures like internal current unknowns

- no longer open-source (*No longer true! See QUCS talk*)
  - ADMS's website: “Noovela Consulting offers - at interesting rate - support to implement your compact models into spice simulators”
  - future support uncertain

- written in XSLT
  - difficult to work with or maintain
  - e.g. took 6-7 years by Tom Russo (working off and on on it) even with ADMS already existing
Model Development: Verilog-A → ModSpec → Xyce

Verilog-A model

→

ModSpec Model (C++ API)

→

.compile standalone

.so libraries (dynamically loadable)

→

use model in Commercial Simulators

→

Test immediately (standalone)

→

model supported in Xyce

→

robust/efficient model deployed

Already tested in MDE and Xyce

High probability of success

binary release possible (IP protection)

fast/efficient

speed near native implementation (compiled C++ code)

via ModSpec C++ API support (easy: example provided for Xyce)

Eric Keiter, Sandia National Laboratories
2014 MOS-AK Workshop, Berkeley, CA
Xyce Summary

• Mature code; under development for 14 years
  • Large regression test suite
  • Regular, formal code releases
• Under active development
• Open source GPL v3.
• Modular object-oriented design
  • DAE form: \( \frac{d}{dt} q(x, t) + f(x, t) = b(t) \)
  • Quick, independent development of models and algorithms
• Model support
  • Legacy spice models
  • Non-electrical: neuron, reaction networks
  • CMC models
  • ADMS integration
  • Mod-Spec integration
Acknowledgements

Xyce team
xyce.sandia.gov
- Scott Hutchinson
- Tom Russo
- Heidi Thornquist
- Jason Verley
- Rich Schiek
- Ting Mei
- Dave Baur
- Sivasankaran Rajamanickam

Trilinos team
trilinos.sandia.gov

Dakota team
dakota.sandia.gov
Thanks!
Publications / Presentations

Publications

• "A Hybrid Approach for Parallel Transistor-Level Full-Chip Circuit Simulation", VECPAR 2014
• “Electrical Modeling and Simulation for Stockpile Stewardship”, ACM XRDS, 2013
• “ShyLU: A Hybrid-Hybrid Solver for Multicore Platforms”, IPDPS 2012
• “Parallel Transistor-Level Circuit Simulation”, Simulation and Verification of Electronic and Biological Systems, Springer, 2011
• “A Parallel Preconditioning Strategy for Efficient Transistor-Level Circuit Simulation”, ICCAD 2009

Presentations

• “Sparse Matrix Techniques for Next-Generation Parallel Transistor-Level Circuit Simulation”
  Heidi K. Thornquist, Parallel Matrix Algorithms and Applications 2012
• “Partitioning for Hybrid Solvers: ShyLU and HIPS”
  Erik G. Boman, Siva Rajamanickam, and Jeremie Gaidamour, Copper Mtn. 2012
• “Efficient Preconditioners for Large-Scale Parallel Circuit Simulation”
  Heidi K. Thornquist, SIAM Computational Science & Engineering 2011
• “Advances in Parallel Transistor-Level Circuit Simulation”
  Heidi K. Thornquist, Scientific Computing in Electrical Engineering 2010
• “Large Scale Parallel Circuit Simulation”
  Heidi Thornquist and Eric Keiter, Circuit and Multi-Domain Simulation Workshop, ICCAD 2009