SAND2022-16939C

# Xyce and support for modern PDKs

Eric Keiter

Sandia National Laboratories

AWG/MOS-AK Panel Discussion

Dec. 7, 2022

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

# Outline

- Xyce open source circuit simulator overview

- Parallelism
  - Talk is *not* focused on solvers, but parallel issues impact parser and setup

- PDK compatibility
  - Device models
  - Analysis options
  - Expression support
  - Language syntax
  - Parser performance

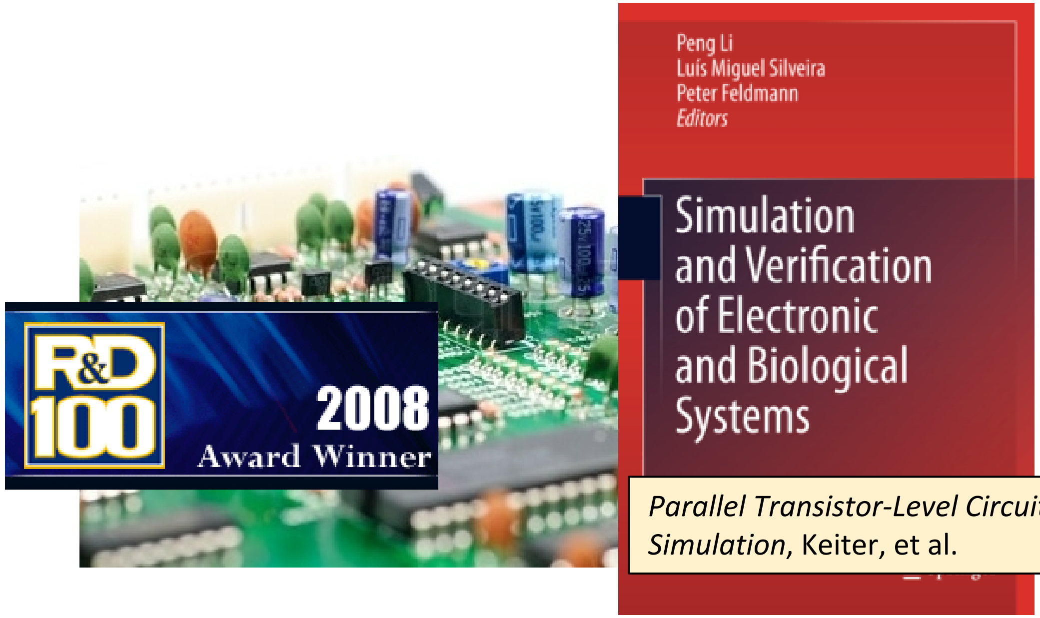
- Xyce status and plans
  - Current parser
  - XDM file translator
  - Replacing the Xyce parser w/ modern parse framework (using XDM grammars)

**https://xyce.sandia.gov**
**https://github.com/xyce**

# The Xyce Analog Circuit Simulator

- SPICE-style simulator includes many industry models
- **Serial** and **Distributed Memory Parallel** (MPI-based)
- Unique solver algorithms
- XDM netlist translator
  - Hspice-to-Xyce
  - Spectre-to-Xyce (new)
- Python model interface (new)
- Xyce at Sandia: `https://xyce.sandia.gov`
  - **Binary executables** for Windows, MacOS and Red Hat Enterprise Linux 7
  - **Xyce** release source code, **build instructions** and more
- Xyce at GitHub: `https://github.com/xyce`
  - For the latest **stable changes** to the **source code**

- **Open Source, GPLv3**
  - Since September of 2013 (Xyce 6.0)
- Xyce Release 7.6
  - Nov, 2022; 32nd major release
  - **>9,100** registrants on **`xyce.sandia.gov`** since 2013
  - Also numerous clones on github




*Parallel Transistor-Level Circuit Simulation*, Keiter, et al.

# Why Open Source?

- Foster external collaboration
- Feedback from wider community
- Taxpayer funded, so encouraged to open source
- Some of our funding requires it

- First open source release, v6.0
- November 5, 2013.
- GPL license v3.0
- Source and binary downloads available
- Most recent release (v7.6) ~Nov 2022.
- Next release (v7.7) ~May 2023.

**https://xyce.sandia.gov**

**https://github.com/xyce**

# Xyce Capabilities

**Typical**

- DC, Transient, AC, Noise
  - .DC, .TRAN, .NOISE, .AC (and .STEP)
- Post Processing:
  - Fourier transform of transient output (.FOUR)
  - Post-simulation calculation of simulation metrics (.MEASURE)
- Output (.PRINT)
  - Text Files (tab or comma delimited)
  - Probe
  - Gnuplot, TecPlot, RAW
- Analog Behavioral Modeling
- Verilog-A model compiler
- Expressions, functions, parameterizations…

**Others**

Harmonic Balance Analysis (.HB)

- Steady state solution of nonlinear circuits in the frequency domain

Random Sampling Analysis

- Executes the primary analysis (.DC, .AC, .TRAN, etc.) inside a loop over randomly distributed parameters

Sensitivities

- Computes sensitivities for a user-specified objective function with respect to a user-specified list of circuit parameters ($\partial O/\partial p$…)
- Works with DC, AC or Transient analysis
- E.g., an output voltage's dependence on a capacitance

Polynomial Chaos methods

- Quadrature
- Regression

# Xyce Simulation Flow

## Parsing

- Convert netlist file syntax to equivalent devices and network/circuit connectivity
- Distribute devices over multiple processors
- Determine global ordering and communication

## Device Evaluation

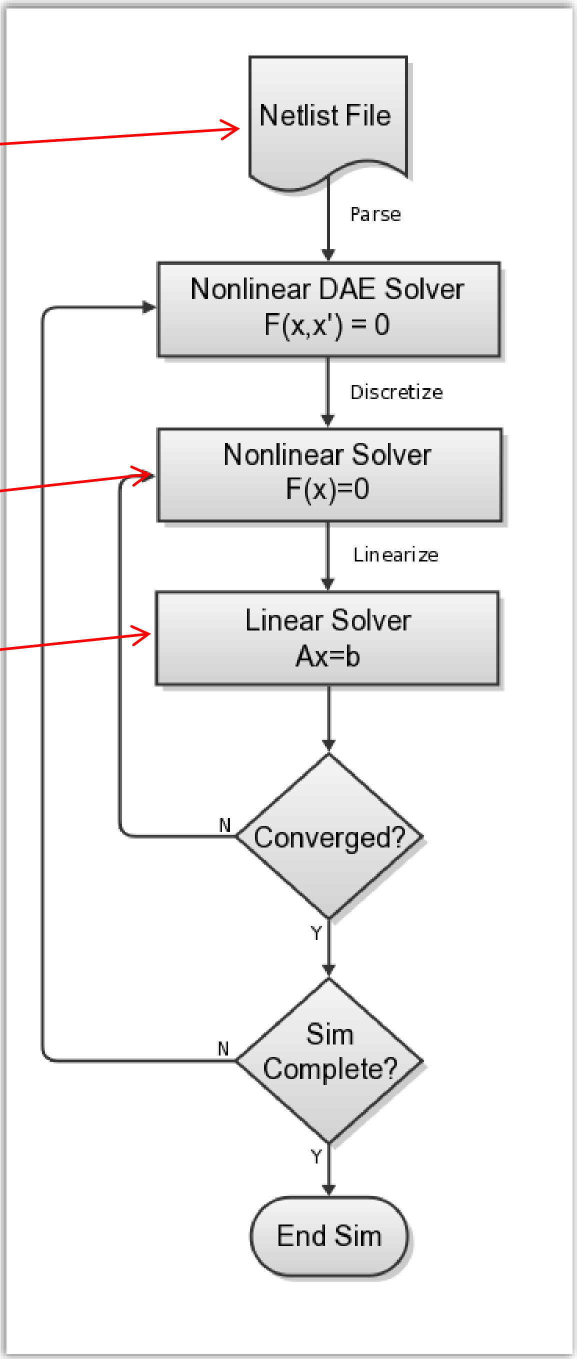
- Loop through all devices for state evaluation and matrix loading

## Linear Solve

- Sparse linear algebra and solvers used to solve linearized system
- Direct solvers more robust, often the choice for commercial tools
- Iterative solvers have potential for better scalability, depends on the preconditioner

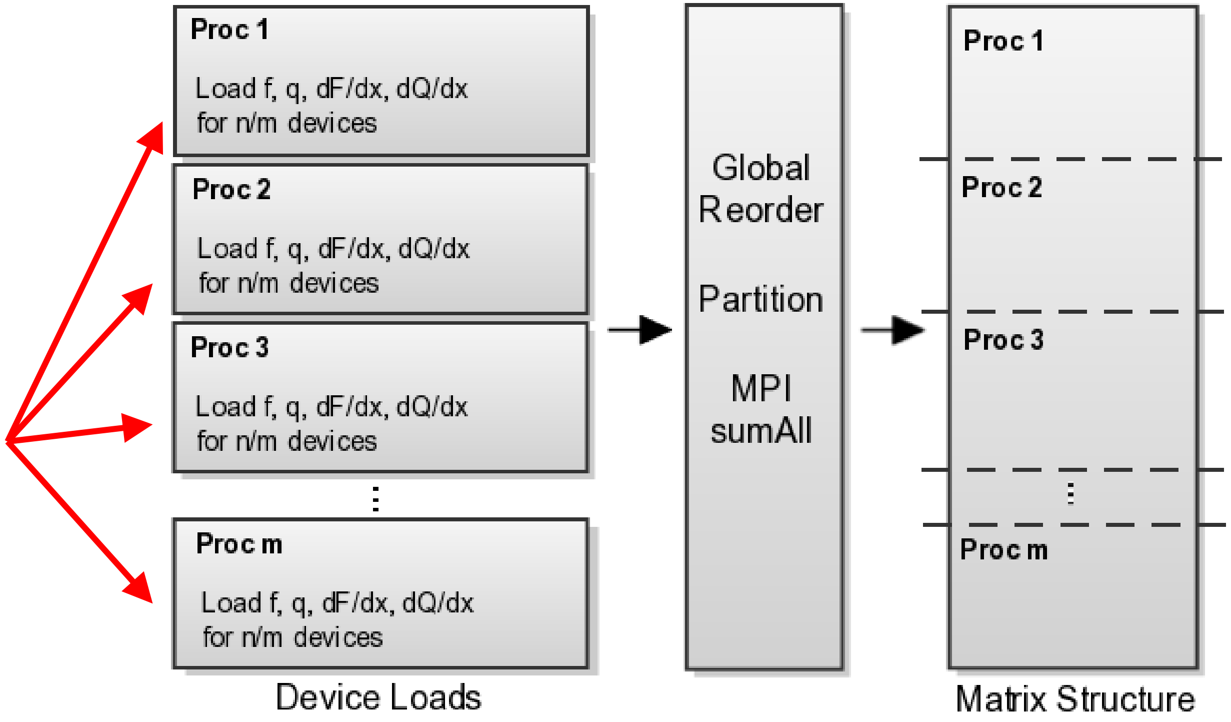
## Advanced Analysis Methods

- Sampling: Monte Carlo, LHS (DAKOTA)
- Optimization

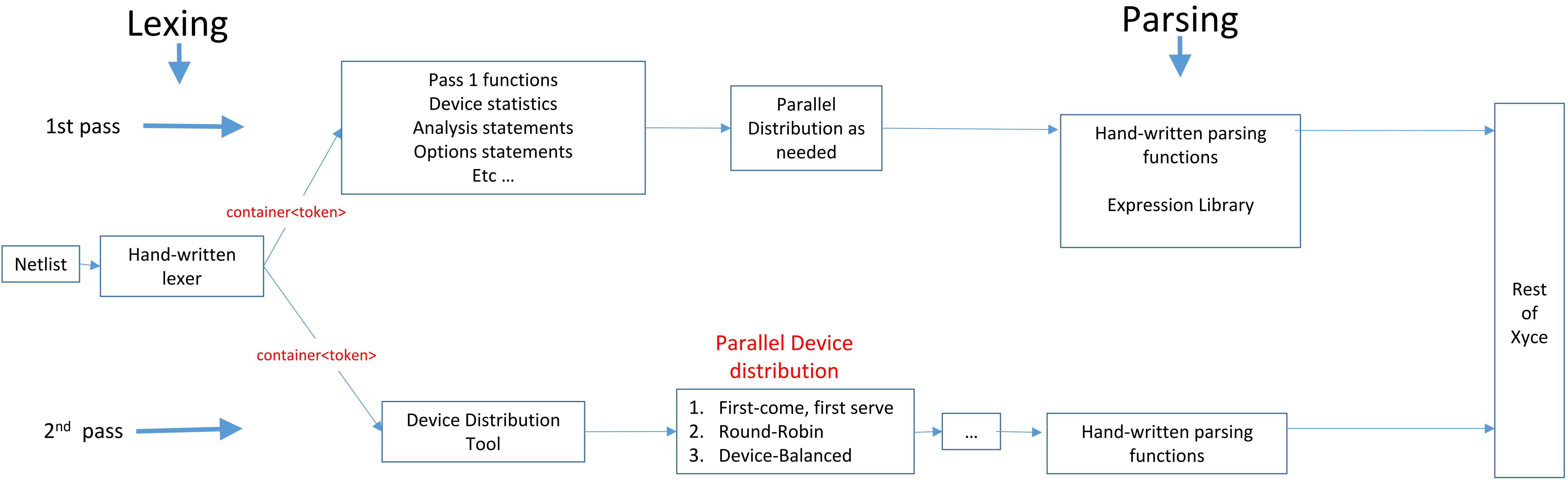


# Two Xyce parallel distributions: device evaluation and matrix solve

- Multiple objectives for load balancing the solver loop
  - Device Loads : The partitioning of devices over processes will impact device evaluation and matrix loads
  - Matrix Structure : Graph structure is static throughout analysis, repartitioning matrix necessary for generating effective preconditioners
- Device Loads
  - Each device type can have a vastly different "cost" for evaluation
  - Memory for each device is considered separate
  - Ghost node distribution can be irregular
  - **Device parallel distribution starts in the parser**
- Matrix Structure
  - Use graph structure to determine best preconditioners / solvers



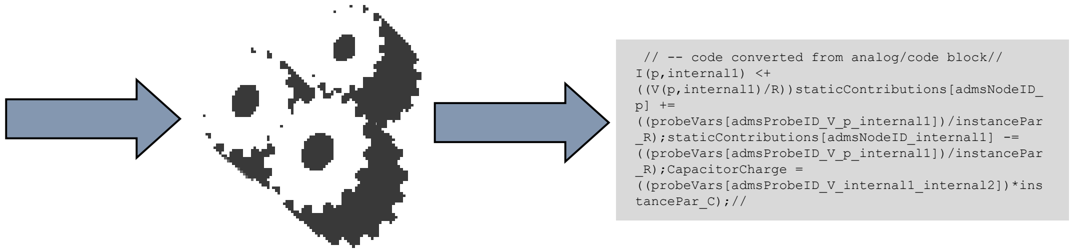
# Xyce Parser Flow




- Parsing happens in 2 passes
- First pass for gathering information (needed by second pass) and parsing that doesn't need specific parallel distribution strategy (broadcast)
- Second pass is mainly for distributing device instances.
- Both passes have a lex and parse phase.
- In second pass, the parallel distribution happens *between* lex and parse.
- Planned refactor: replace hand-written lex and parse functions with modern lex/parse framework
- Use the grammars developed for the XDM tool (Spectre, Hspice, etc)

# Xyce PDK compatibility

- In practice, PDK compatibility means netlist compatibility with commercial simulators
- Xyce syntax compatibility
  - Xyce native parser improvements close to ngspice/Hspice
  - Xyce Data Model (XDM)
    - Available as part of Xyce code releases and also on github: https://github.com/Xyce/XDM
    - Converts Hspice or Spectre format files to Xyce format
- Expression library
  - Completely rewritten to support GF 12/14.
  - Modern parser design
  - Much faster, better scalability
- Verilog-A model compiler (ADMS = automatic device model synthesizer)

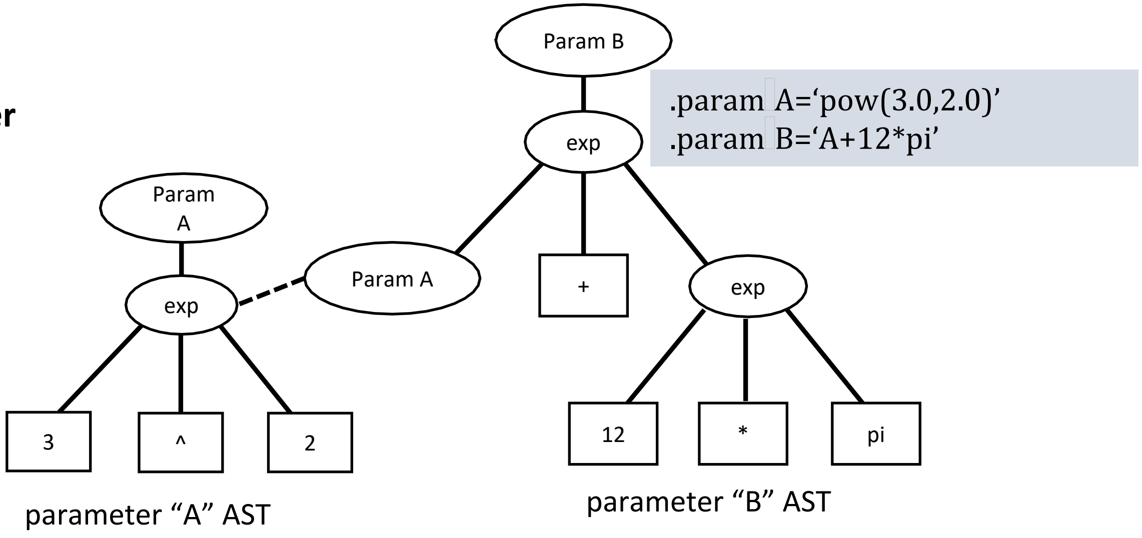

C++ code snippet
(actual Xyce file is 1500 lines)

- Support for industry standard compact models: BSIM-CMG, UTSOI, BSIM4, etc.

| PDK | Xyce demonstrated |
|---|---|
| GF 65nm | ✓ |
| GF 55nm | ✓ |
| GF 45nm | ✓ |
| GF 14nm | ✓ |
| GF 12nm | ✓ |
| ST 28nm | ✓ |
| TSMC 130 | ✓ |
| TSMC 65 | ✓ |
| PTM 45nm | ✓ |
| Sky130 | ✓ |

# PDK Compatibility: Expression performance

- **New expression library:** Xyce has had an old expression library for many years, that contained a large amount of technical debt. Recently, with the 12nm GF PDK, we encountered an issue that couldn't patched, so we wrote a new expression library.
  - **With the new library the 12nm GF PDK parses successfully.**
  - **Fixed at least 20 long-standing expression issues in our internal issue tracker**
  - **Part of Xyce 7.2 (Nov, 2020)**
- The 12nm GF PDK was that it had expressions with many levels of nesting.
- Old library handled external dependencies via string substitution (bad!)
- In the new library, this doesn't happen



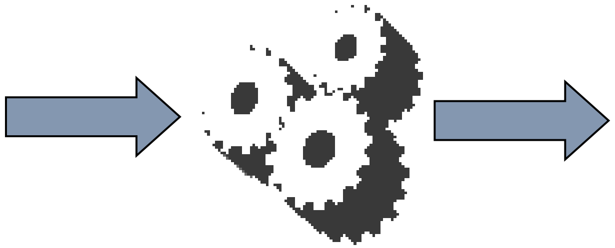

| 12nm GF Circuit | Simulation Time Xyce v7.1 | Simulation Time Xyce v7.2 | Simulation Speedup |
|---|---|---|---|
| UW VCO | ∞ sec | 20 sec | ∞ |

- **Improved parameter searches:** Extensive use of parameters, through .PARAM statements, was identified as a performance bottleneck
  - **Replaced hidden linked list structure with hash table**
  - **This improved the performance on internal GF45 circuits**
  - **Part of Xyce 7.2 (Nov, 2020)**

| Circuit | Simulation Time Xyce v7.1 | Simulation Time Xyce v7.2 | Simulation Speedup |
|---|---|---|---|
| 2 Clock Cycles | 459 sec | 60 sec | ~8x |
| 10 Clock Cycles | 1025 sec | 361 sec | ~3x |

# PDK Compatibility: ADMS-Xyce model compiler

- ADMS = *Automatic Device Model Synthesizer*
- Verilog-A: industry standard format for new models, including (relevant to DARPA):
  - BSIM-CMG (FinFETs) – needed by process nodes <= 14nm.
  - UTSOI – needed by ST28nm PDK.
- Automatically translates ***Verilog-A*** to Xyce-compliant C/C++ code
- Automatic differentiation (AD) was recently rewritten for better performance
- Can be invoked dynamically
- New replacement compiler under development

Run admsXyce

```
 // -- code converted from analog/code block// I(p,internal1) <+
((V(p,internal1)/R))staticContributions[admsNodeID_p] +=
((probeVars[admsProbeID_V_p_internal1])/instancePar_R);staticContribution
s[admsNodeID_internal1] -=
((probeVars[admsProbeID_V_p_internal1])/instancePar_R);CapacitorCharge =
((probeVars[admsProbeID_V_internal1_internal2])*instancePar_C);//
```

C++ code snippet
(actual Xyce file is 1500 lines)

New AD performance improvements

| Circuit | Model | AD residual | New AD residual | Residual speedup | AD total | New AD total | Total speedup |
|---|---|---|---|---|---|---|---|
| CMG inverter | BSIM CMG | 5.5 sec | 1.13 sec | 4.88x | 5.9 sec | 1.5 sec | 3.93x |
| CMG testcase | BSIM CMG | 71 sec | 14 sec | 5.1x | 74 sec | 17 sec | 4.35x |
| “Perry’s Circuit” | VBIC | ~70 hours | ~6.5 hours | 10x | ~77 hours | 13 hours | 5.9x |

# Notes about device model compatibility

- Support for industry standard models is mandatory
- Si2/CMC pushing standardization
- However, for older models (some of which pre-date this effort) standards are not always clear

- Recent examples (for us):
  - Spice3 diode not the same as many simulators' diodes (sidewall capacitances)
  - Berkeley BSIM3 not the same as many simulators (geometrical parameters)
  - Berkeley BSIM4 not the same
  - etc.

# Recent new Xyce compatibility improvements (not exhaustive)

- Done recently
  - Support for multipliers on all device models
  - Support for subcircuit multipliers
  - Support for .DATA
  - Many expression operators: int(x), limit(x,y), sign(x,y), etc.
  - Many .MEASURE features
  - Support for .LIB
  - Support for relative paths for .include and .lib
  - Support for undelimited expressions
  - Parameter precedence (if more than one param has same name, how to choose)
  - ”atto” suffix. In Hspice the “a” suffix means 1e-18. In others, it means “amps”.

- In progress
  - .IF/.ELSE/.ELSEIF/.ENDIF
  - Reading .VEC files
  - Reading SPEF files
  - Supporting “$” as comment delimiter
  - .AUTOSTOP
  - etc

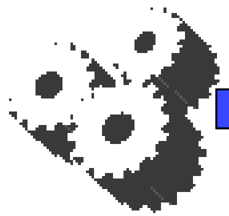
# Tool Compatibility: Xyce Data Model (XDM)

- First released as part of Xyce 7.0 (April, 2020)
- For modern PDK files, file format is either Hspice or Spectre
- Pspice-to-Xyce input file translation complete
- Hspice-to-Xyce input file translation complete
- Spectre-to-Xyce file translation in progresss
- XDM is a stand-alone file translator, but ***eventually will replace Xyce parser*** (see next slides)

- Available as part of Xyce code releases and also on github: **https://github.com/Xyce/XDM**

```
…
V5 (net3 0) vsource dc=1 type=dc

V7 (net1 0) vsource type=exp val0=0.0 \
val1=1.0 td1=100u tau1=20u td2=600u \        tau2=40u

V1 (net7 0) vsource dc=0 mag=0 phase=0 \ type=sine
delay=0 sinedc=0 ampl=1 \
freq=1K damp=0

V9 (net9 0) vsource type=sine sinedc=1 \
ampl=1 freq=1K

V11 (net11 0) vsource type=pulse val0=1\
val1=10 period=400u delay=50u \       rise=50.0u fall=70.0u
width=200u
```

**Spectre netlist**

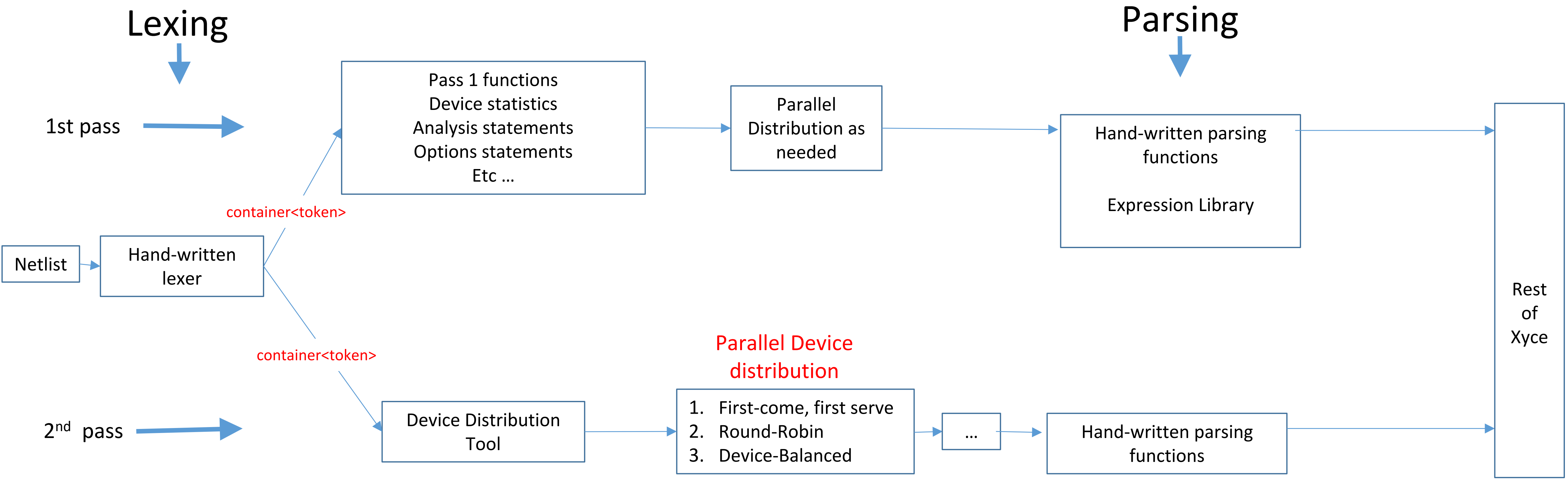


**XDM**

```
…
VV5 net3 0 DC 1
VV7 net1 0 EXP(0.0 1.0 100u 20u 600u 40u)
VV1 net7 0 SIN(0 1 1K 0 0 0)
VV9 net9 0 SIN(1 1 1K 0 0 0)
VV11 net11 0 PULSE(1 10 50u 50.0u 70.0u 200u 400u)
RR7 net1 V8 R=1K
…
```

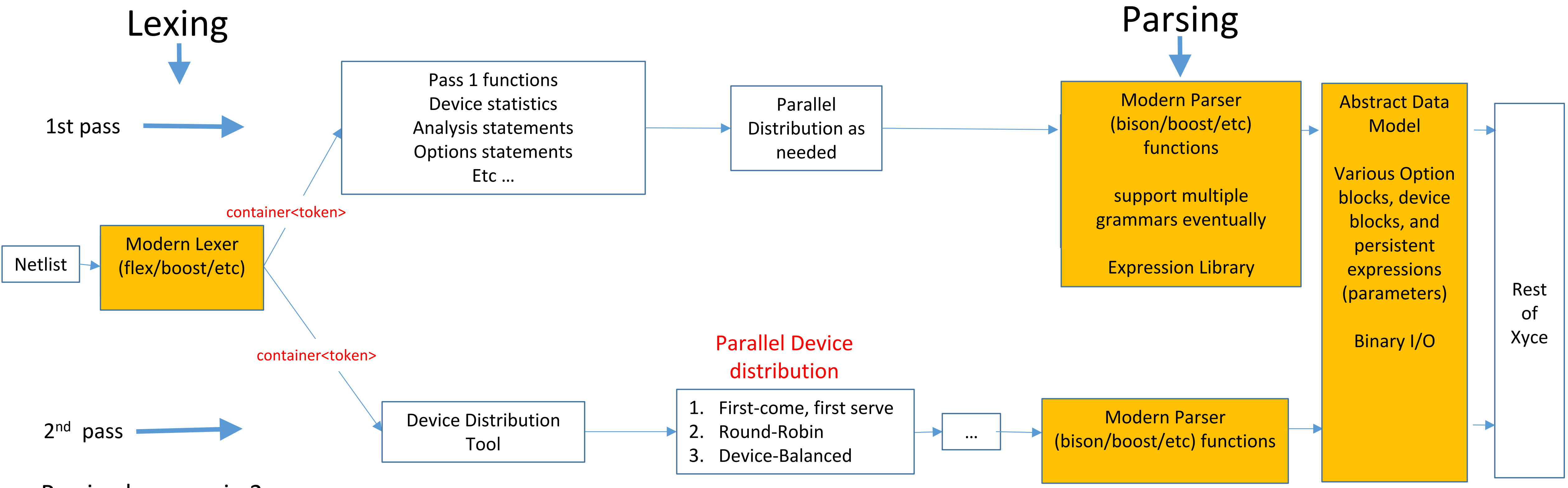
**Xyce netlist**

# Xyce Parser Flow




- Parsing happens in 2 passes
- First pass for gathering information (needed by second pass) and parsing that doesn't need specific parallel distribution strategy (broadcast)
- Second pass is mainly for distributing device instances.
- Both passes have a lex and parse phase.
- In second pass, the parallel distribution happens *between* lex and parse.
- Planned refactor: replace hand-written lex and parse functions with modern lex/parse framework
- Use the grammars developed for the XDM tool (Spectre, Hspice, etc)

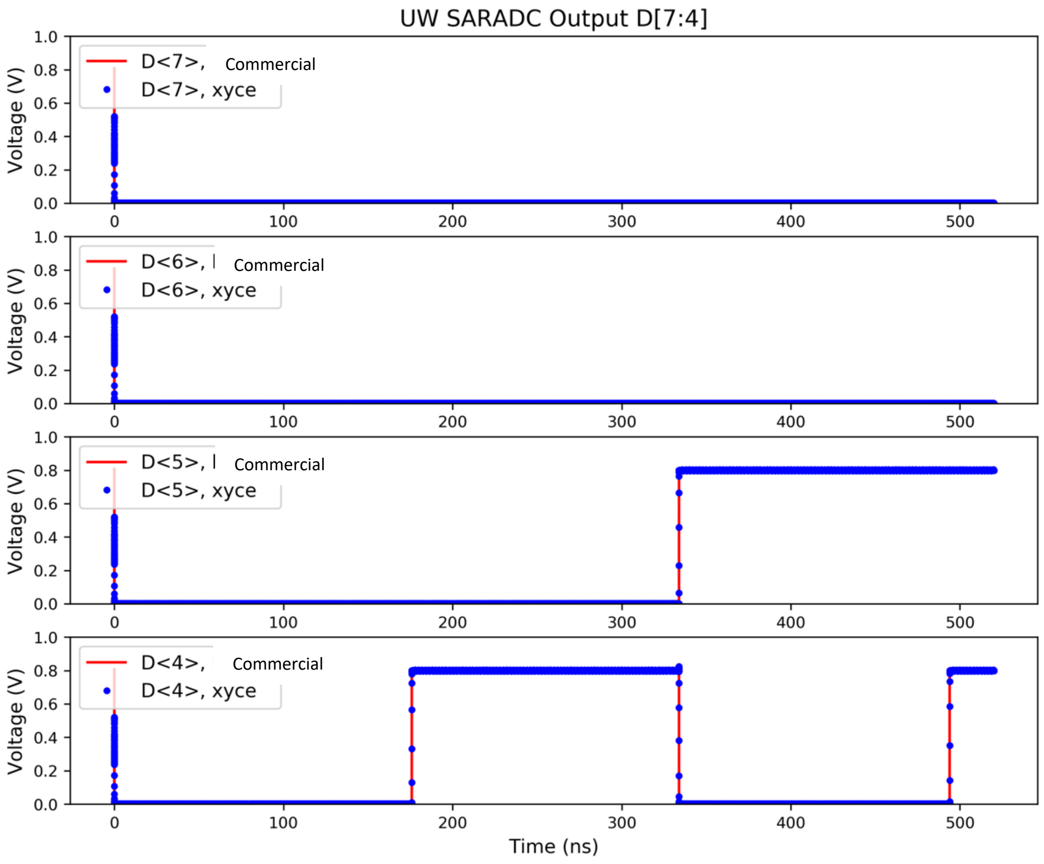
# Xyce Parser Flow

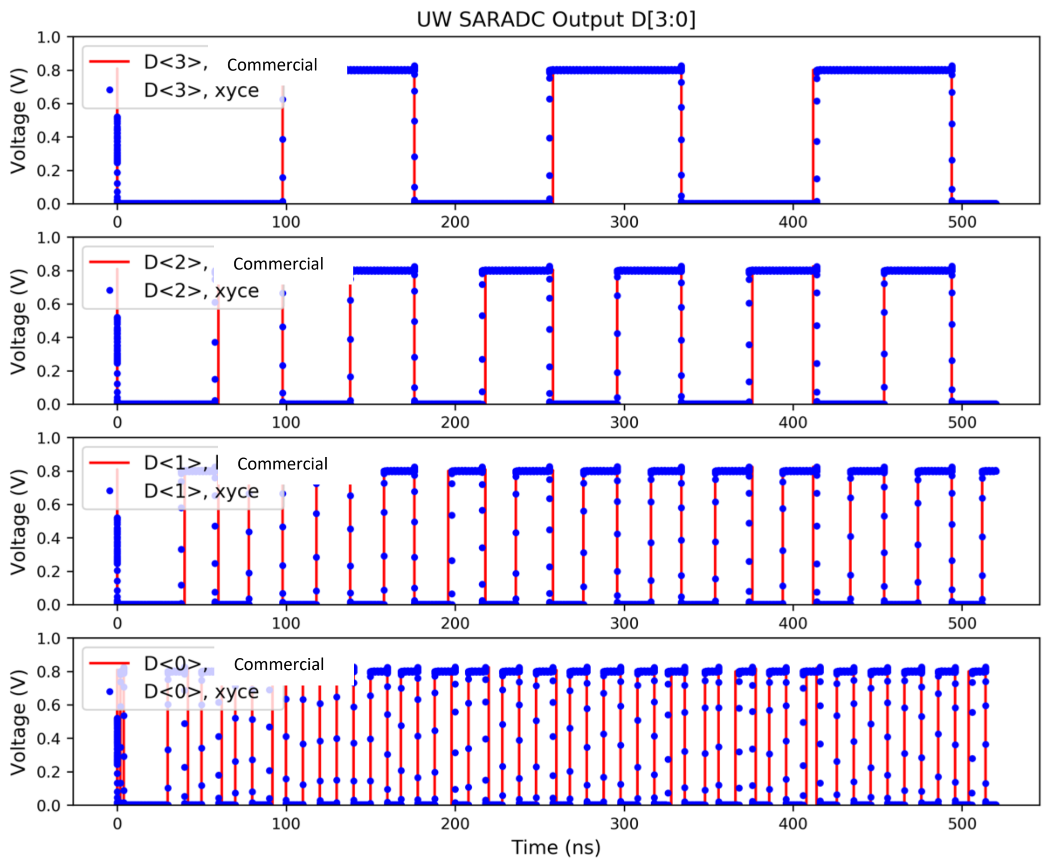



- Parsing happens in 2 passes
- First pass is for gathering information (needed by second pass) and also for parsing stuff that doesn't need specific parallel distribution strategy
- Second pass is mainly for distributing device instances.
- Both passes have a lex and parse phase.
- In second pass, the parallel distribution happens *between* lex and parse.
- Planned refactor: replace hand-written lex and parse functions with modern lex/parse framework

# PDK Compatibility example: UW SAR ADC Circuit example (GF 12nm)

- Circuit is from an external group (non-Sandia)
- Xyce results match commercial simulator
- XDM+Xyce (version > 7.2) now supports the Global Foundries 12nm PDK



# Xyce Team Acknowledgements

- Eric R. Keiter
- Thomas V. Russo
- Richard L. Schiek
- Heidi K. Thornquist
- Ting Mei
- Jason C. Verley
- Karthik V. Aadithya
- Joshua D. Schickling
- Paul Kuberry
- Gary Templet
- Garrick Ng
- …and many others

Contact:

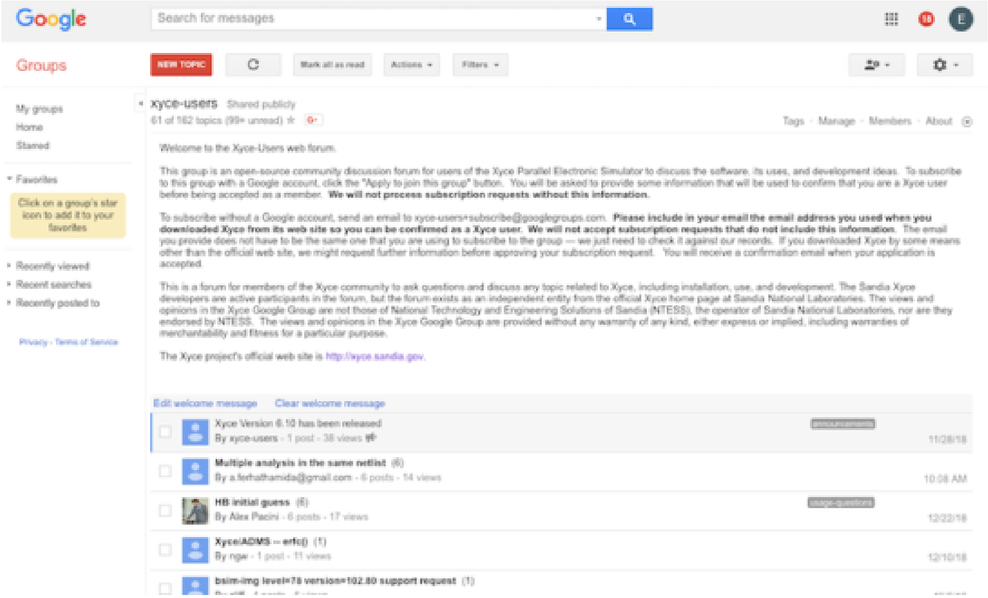
**https://github.com/xyce**

**https://xyce.sandia.gov**

**xyce@sandia.gov**

Google Group Forum:

**https://groups.google.com/group/xyce-users**

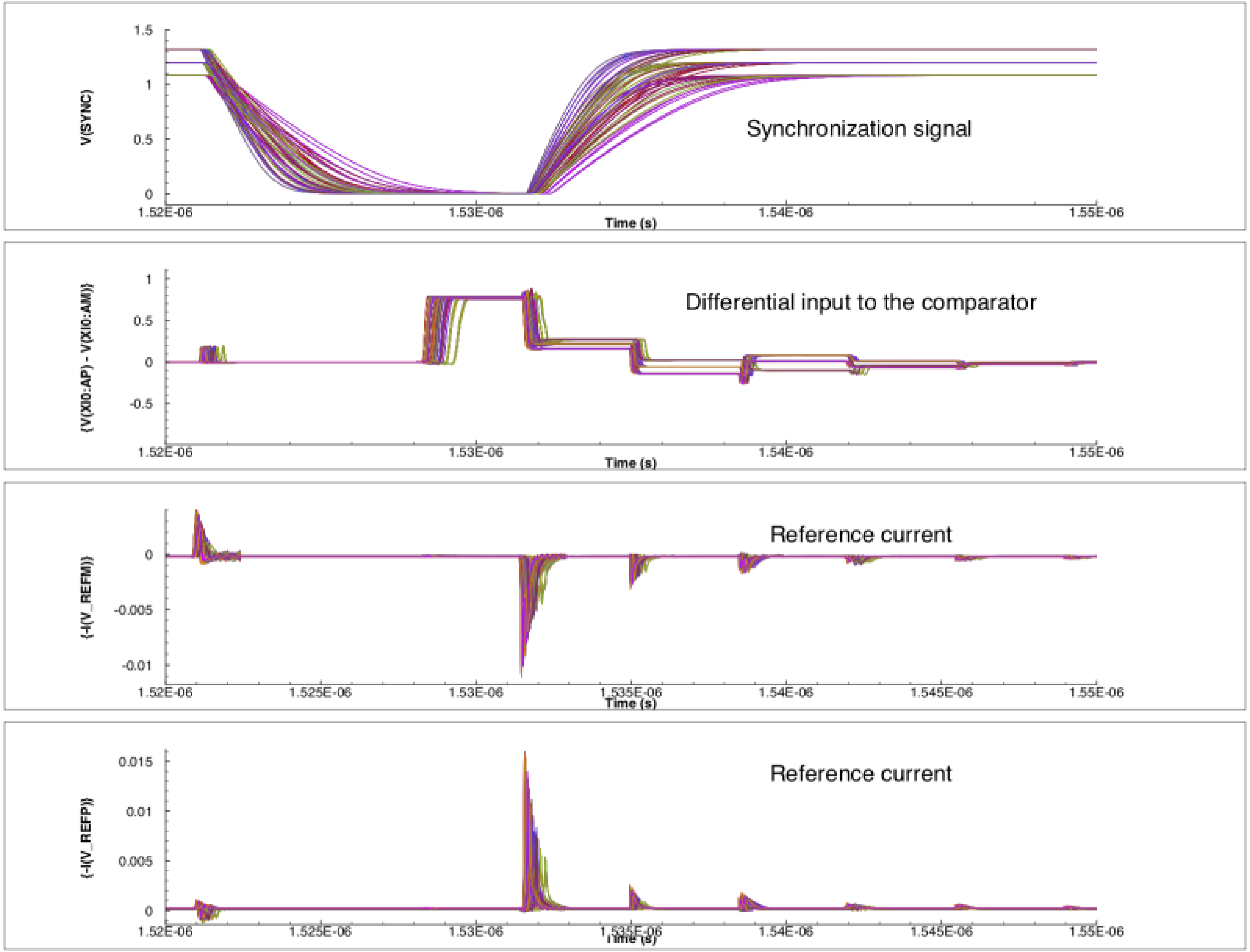


# Extra Slides

X

# PDK Compatibility example: SAR ADC Circuit example (GF 65nm)

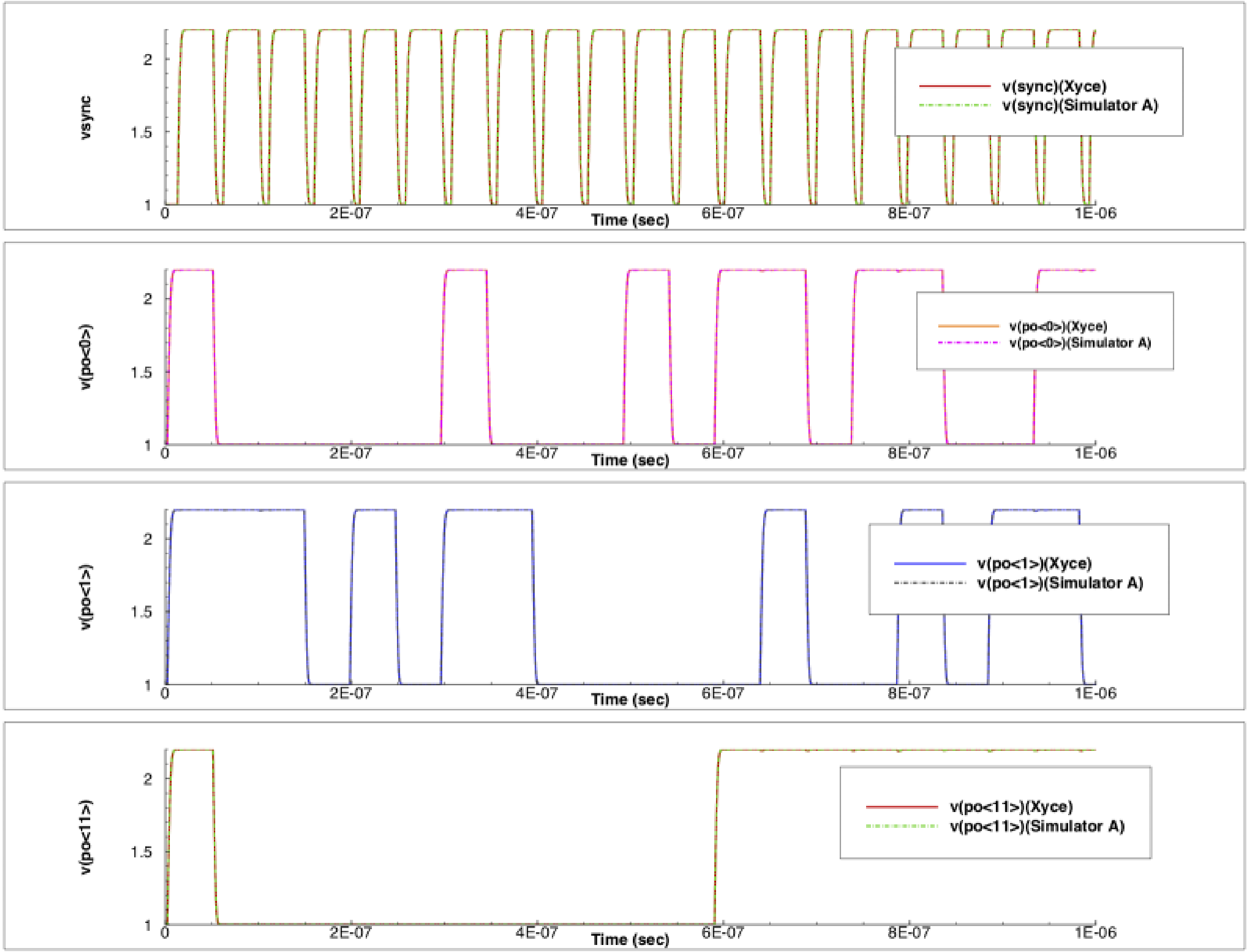
- **SAR ADC** = successive approximation register analog-to-digital converter
- Developed under the POSH program by Bindu Madhavan and Edward Lee
  - working with the POSH group from USC.
- GF 65nm
- 400 corner study



# PDK Compatibility example:
# SAR ADC Circuit example (GF 65nm)

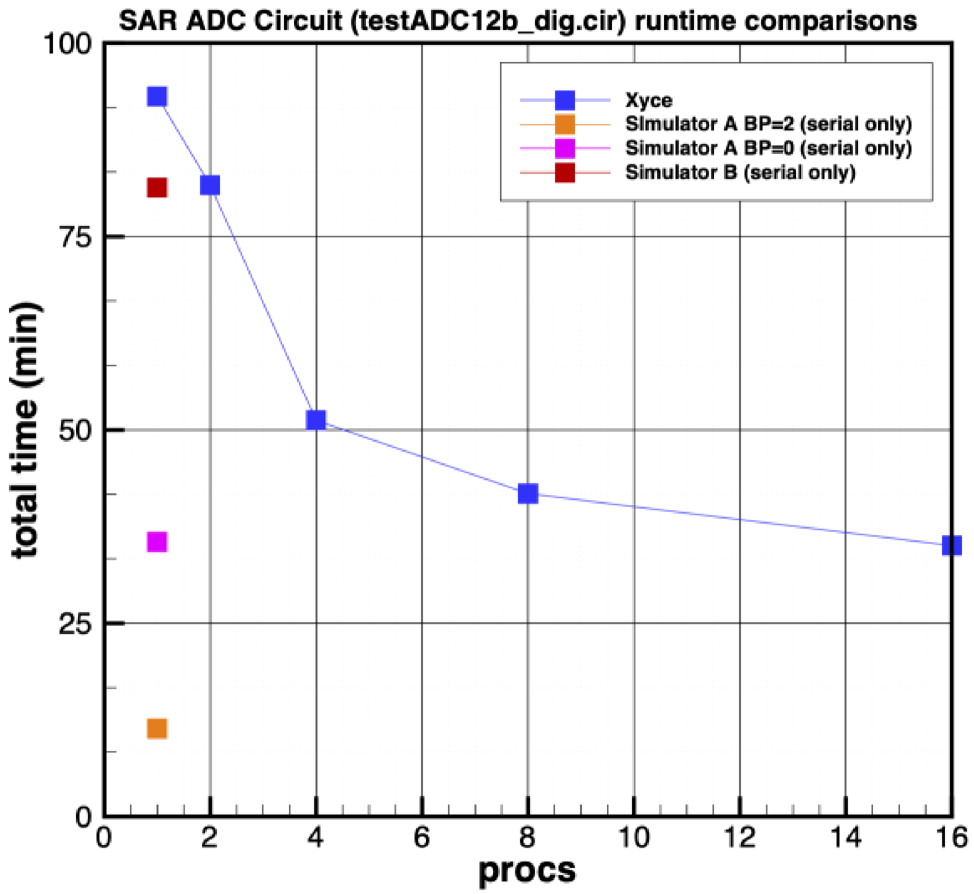
- **Results match well.  RMS Errors small**

- RMS relative error in v(sync) is 0.0434905680015374%
- RMS relative error in v(po<0>) is 0.0232474456164593%
- RMS relative error in v(po<1>) is 0.023581461963474%
- RMS relative error in v(po<2>) is 0.02583082511786%
- RMS relative error in v(po<3>) is 0.0240096727254828%
- RMS relative error in v(po<4>) is 0.0166525520072121%
- RMS relative error in v(po<5>) is 0.00929693070847055%
- RMS relative error in v(po<6>) is 0.0309201017241085%
- RMS relative error in v(po<7>) is 0.0230237794341722%
- RMS relative error in v(po<8>) is 0.0259005260949305%
- RMS relative error in v(po<9>) is 0.0175662606806119%
- RMS relative error in v(po<10>) is 0.00940986678122403%
- RMS relative error in v(po<11>) is 0.00976999004888706%



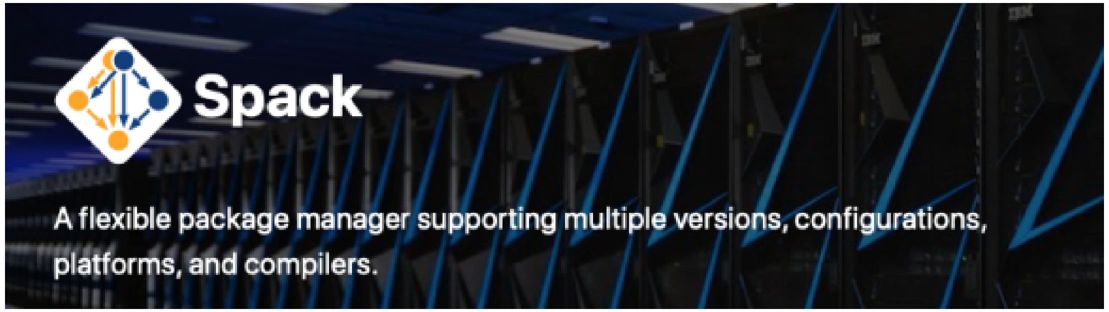
# PDK Compatibility:
# SAR ADC Circuit example (GF 65nm) Simulation timings

- GF 65nm
- Recent efficiency improvements to Xyce have brought it close to “Simulator B” for one processor.
- Still work to do to catch “Simulator A”.
- Some of the difference is due to BYPASS, which is present in ”Simulator A”, but not Xyce or “Simulator B”.



# Xyce Distributions

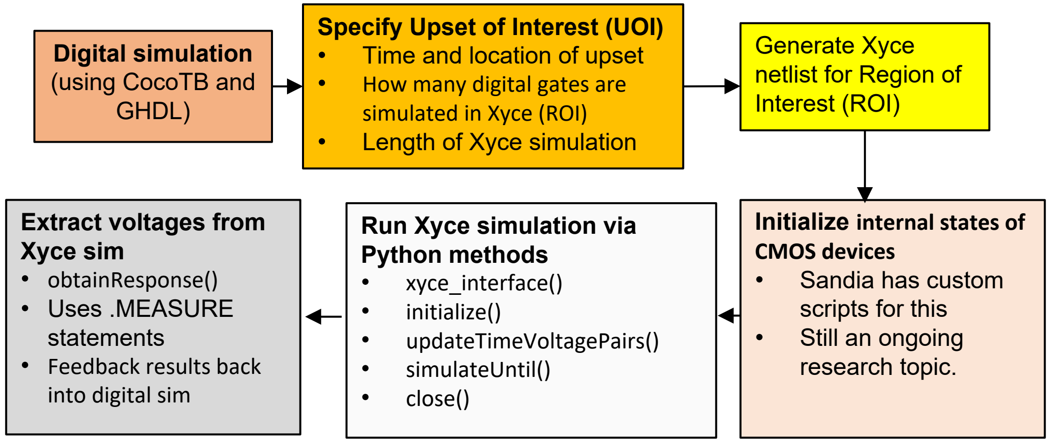
- Binary installers (serial and parallel)
  - RHEL 7
  - MacOS
  - Windows (serial only)
  - http://xyce.sandia.gov
- Source code
  - http://xyce.sandia.gov
  - http://github.com/Xyce
- New: Installing Xyce via Spack, “a package manager for supercomputers, Linux, and MacOS”
  - Use this to install Xyce with the python model interpreter (***Xyce-PyMi***) enabled.

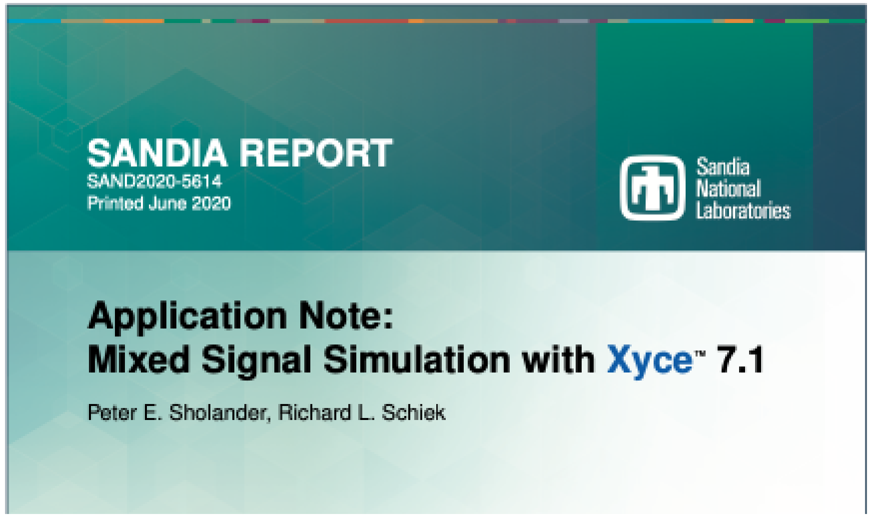



https://spack.io

# Xyce Mixed-Signal API

**Example Sandia Mixed-signal Use-case**





SANDIA REPORT
SAND2020-5614
Printed June 2020
Sandia National Laboratories
Application Note:
Mixed Signal Simulation with Xyce™ 7.1
Peter E. Sholander, Richard L. Schiek

https://xyce.sandia.gov/files/xyce/AppNote-MixedSignal.pdf

**Xyce Supports mixed signal by being callable as a library**

Mixed Signal Simulation with Xyce:

- Both Python and C/C++ interfaces available
- Supported on RHEL6 and RHEL7
- Used by internal Sandia projects
  - See SAND2018-14109

- Coupling Examples:
  - Pyghdl (VHDL)
  - GHDL (VHDL)
  - Icarus (Verilog)
  - Yale simulator (Prsim)
  - Amstaff from Synopsys

# Surrogate Modeling:
# ML-based Python device models in Xyce (Xyce-PyMI)

**Goal:** To develop *platform-independent interpreter(s) for ML-based surrogate models*

**Approach:**
- Using Pybind11 to enable Xyce to call the Python interpreter
- Leveraging Xyce GeneralExternal interface (C++)

**Benefit:**
- Calling Python classes allows for enabling various ML models based on TensorFlow, PyTorch, etc…

**Distribution:**
- Available in Xyce development branch (on github.com), also in Xyce v7.3 release.

Xyce pybind11

>> Xyce-PyMi example.cir

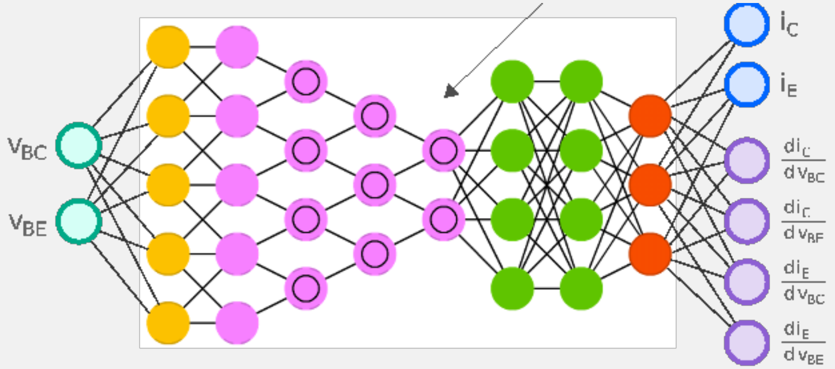
Xyce-PyMi
(Xyce + **Py**thon **M**odel **I**nterpreter)

- Full Xyce functionality + devices / subcircuits / circuits defined in Python
- Python class defines how F, Q, B, dF/dX, and dQ/dX vectors/matrices are populated for Xyce DAE equation:
  - residual = f(x,t) + dq(x,t)/dt - b(t)

```
* example.cir (Example of easy inclusion in a netlist)
YGENEXT devicename terminal1 terminal2
+ SPARAMS={NAME=MODULENAME VALUE=PythonDevice.py}
```

```
# PythonDevice.py
import numpy as np
from BaseDevice import BaseDevice
class Device(BaseDevice):
   def computeXyceVectors(…):
      # definition of device in Python goes here
```

PyTorch TensorFlow NumPy

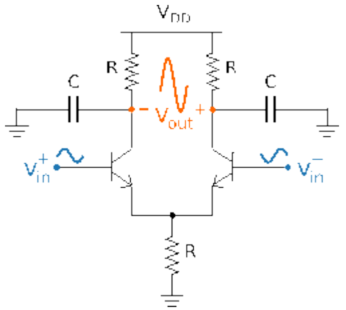


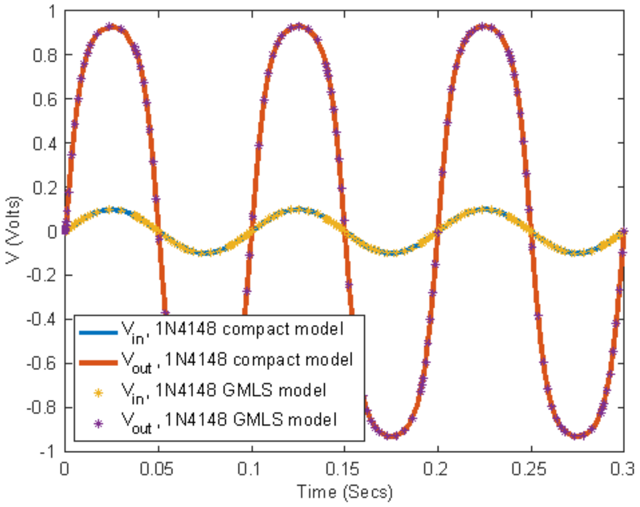

"75% of ML developers and data scientists use Python"
- State of the Developer Nation (Slashdata.co 2020)

# Surrogate Modeling: Compact model examples

## Operational Amplifier with BJTs

```
**************
* Netlist for Operational Amplifier
**************
VDD    1   0  DC   2.5
R1 1 4 1e4
R2 1 5 1e4
R3 6 0 5e3
C1 4 0 5e-12
C2 5 0 5e-12
YGENEXT pyQ1 4 7 6
+ SPARAMS={NAME=MODULENAME,DATAFILE
        VALUE=/models/gmls_bjt_2N2222.py,../data/2N2222_alan.01.dat}
RQ1 7 2 50
YGENEXT pyQ2 5 8  6
+ SPARAMS={NAME=MODULENAME,DATAFILE
        VALUE=/models/gmls_bjt_2N2222.py,../data/2N2222_alan.01.dat}
RQ2 8 3 50
Em_plus 2 0 VALUE={1+50e-3*sin(2*pi*10*time)}
Em_minus 3 0 VALUE={1-50e-3*sin(2*pi*10*time)}
```

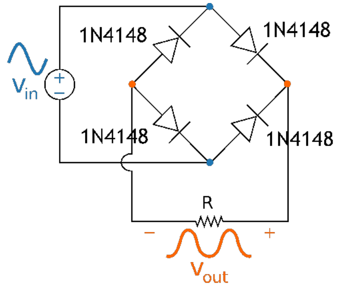


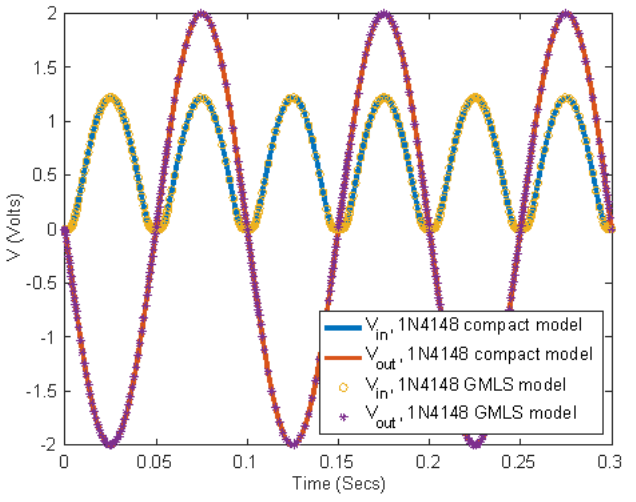




* Runs GMLS on data generated from synthetic MMBT2222, NPN, Fairchild

## Fast switching 1N4148 diode in bridge rectifier

```
**************
* Netlist for Bridge Rectifier
**************
V3 1 2 SIN (0 2 10)
R3 3 0 10M
R4 3 4 100K
YGENEXT pyd3 1 4
+ SPARAMS={NAME=MODULENAME,DATAFILE
VALUE=../models/gmls_diode_1N4148.py,../data/1N4148_synthetic.dat}
YGENEXT pyd1 3 1
+ SPARAMS={NAME=MODULENAME,DATAFILE
VALUE=../models/gmls_diode_1N4148.py,../data/1N4148_synthetic.dat}
YGENEXT pyd4 3 2
+ SPARAMS={NAME=MODULENAME,DATAFILE
VALUE=../models/gmls_diode_1N4148.py,../data/1N4148_synthetic.dat}
YGENEXT pyd2 2 4
+ SPARAMS={NAME=MODULENAME,DATAFILE
VALUE=../models/gmls_diode_1N4148.py,../data/1N4148_synthetic.dat}
.TRAN 0 0.3s
.options timeint reltol=1.0e-4
.PRINT TRAN V(1) V(2) V(3) V(4) V(2,1) V(4,3)
.END
```



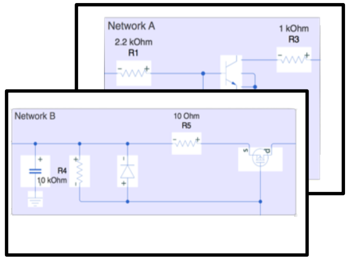

* Runs GMLS on data generated from synthetic 1N4148

# Surrogate Modeling: Circuit examples

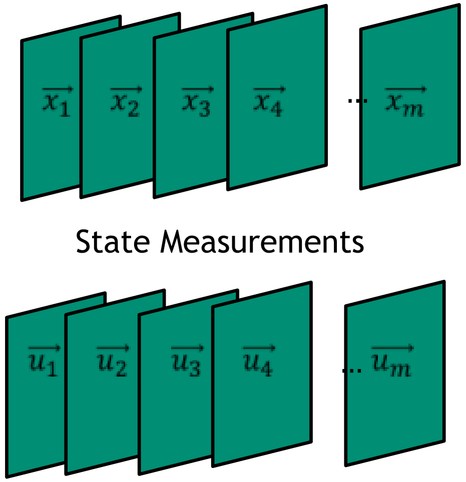
PI: Edgar Galvan

- **Dynamic Mode Decomposition (DMD)**

Circuit Simulations




Snapshots




Control/Input Measurements

System Identification

$$X = [\vec{x}_1, \vec{x}_2, \vec{x}_3, \dots, \vec{x}_{m-1}]$$

$$X' = [\vec{x}_2, \vec{x}_3, \vec{x}_4 \dots, \vec{x}_m]$$

$$U = [\vec{u}_2, \vec{u}_3, \vec{u}_4 \dots, \vec{u}_m]$$

Snapshot Matrices

$$\vec{x}_{k+1} = A\vec{x}_k + B\vec{u}_k$$

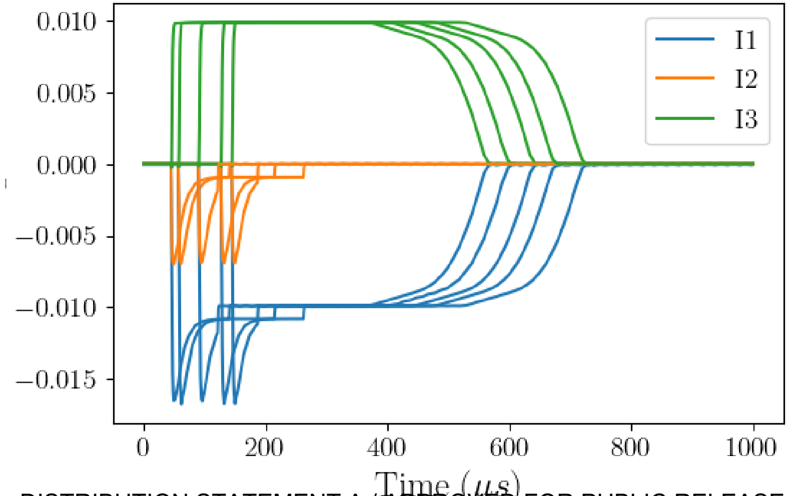
$$[A, B] = X' \begin{bmatrix} X \\ U \end{bmatrix}^{\dagger}$$

Identify Discrete System

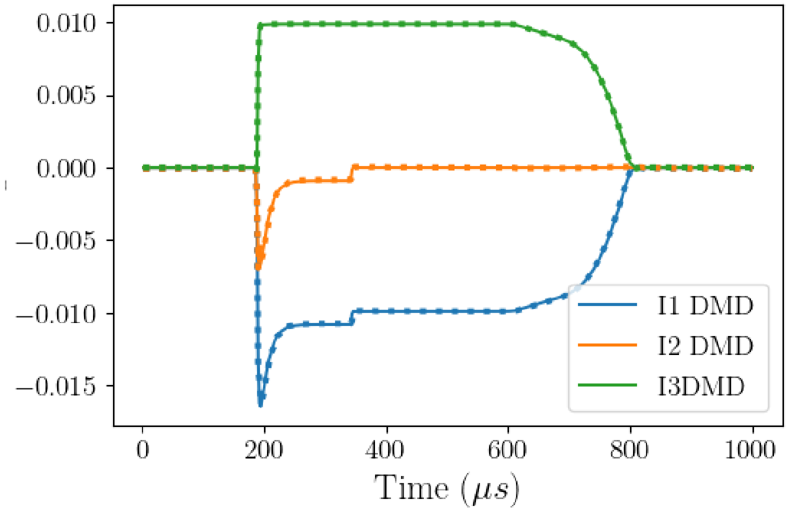
- **Power Amplifier Circuit (PAC)** loosely inspired by circuits at Sandia. The primary function of the circuit is to drive a load at some voltage when a logic HI signal is received.
- The DMD abstraction is trained using several trajectories where the input is a logic HI signal with varying maximum voltages, pulse duration and start time.

† denotes Moore-Penrose Inverse

Training Trajectories




Validation Trajectories



# Schematic editing with Xyce

- Xyce is the simulator (like HSPICE, SmartSpice, Spectre, Eldo…), so Sandia does not provide a Schematic GUI. ***However***:

**Qucs-S**
- Open Source (GPL2)
- https://ra3xdh.github.io
- https://github.com/ra3xdh/qucs_s

**Typhoon HIL**
- Free (not open-source)
- https://www.typhoon-hil.com/products/xyce-integration/
- Electrical power/distribution focus