IMS
Institute of Microelectronic Systems
Leibniz Universität Hannover

Leibniz Universität Hannover

The PRAISE Approach for Accelerated Transient Analysis Applied to Wire Models

BMAS'09: D. Zaum, S. Hoelldampf, I. Neumann, S. Schmidt, M. Olbrich and E. Barke

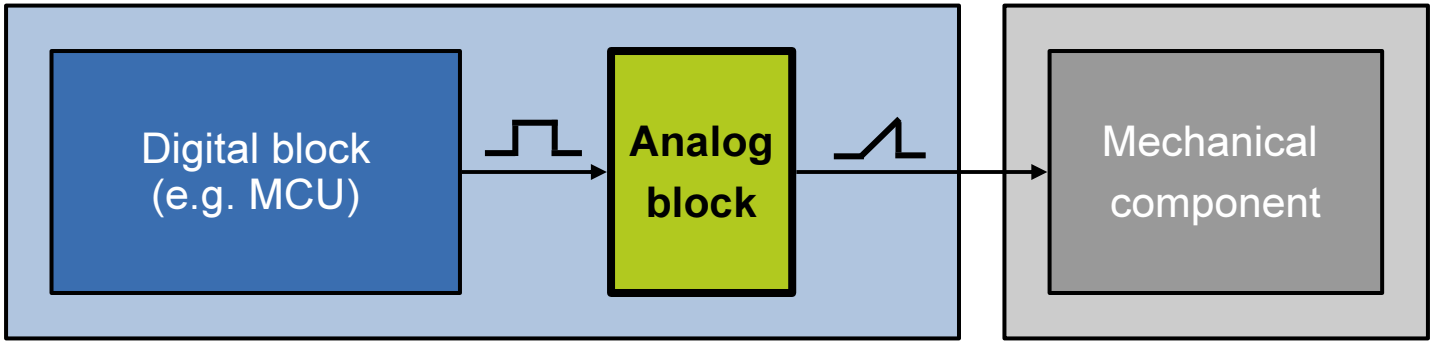
Outline

- Introduction
- The PRAISE Approach
- Implementation
- Application to Wire Models
- Conclusion and Outlook

Motivation

- Increasingly complex automotive electronic devices
- Accelerated system-level verification required
- PRAISE Approach
  - Mixed-signal transient simulation of analog subcircuits
  - Precomputation of behavioral models previous to the simulation
  - Fast evaluation of exponential terms during simulation
  - No explicit numerical integration required
  - SystemC-Interface for effortless integration

Definition of Automotive Circuits



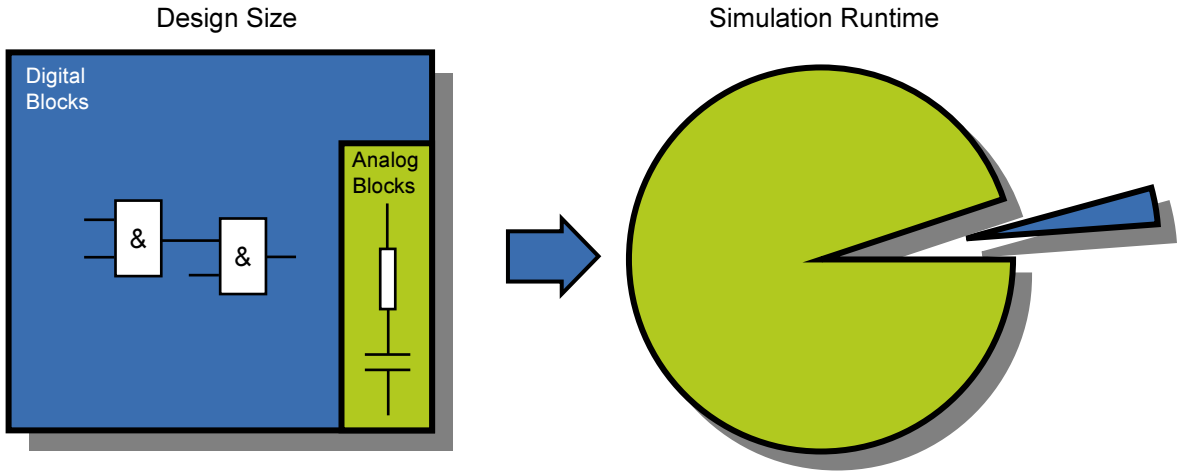
- Analog power electronic blocks as interface between
  - Digital blocks
  - Mechanical components

- Digital signals can be modeled as piecewise-constant inputs of the analog blocks

Bottleneck of automotive system-level verification

- Poor simulation performance of mixed-signal systems
  - Fast digital simulation, but analog simulation suffers from runtime issues
- Slowdown of system simulation due to simulator coupling
- More than 90 % of simulation time spent on analog blocks

Empirical data in an automotive context provided by industry partners

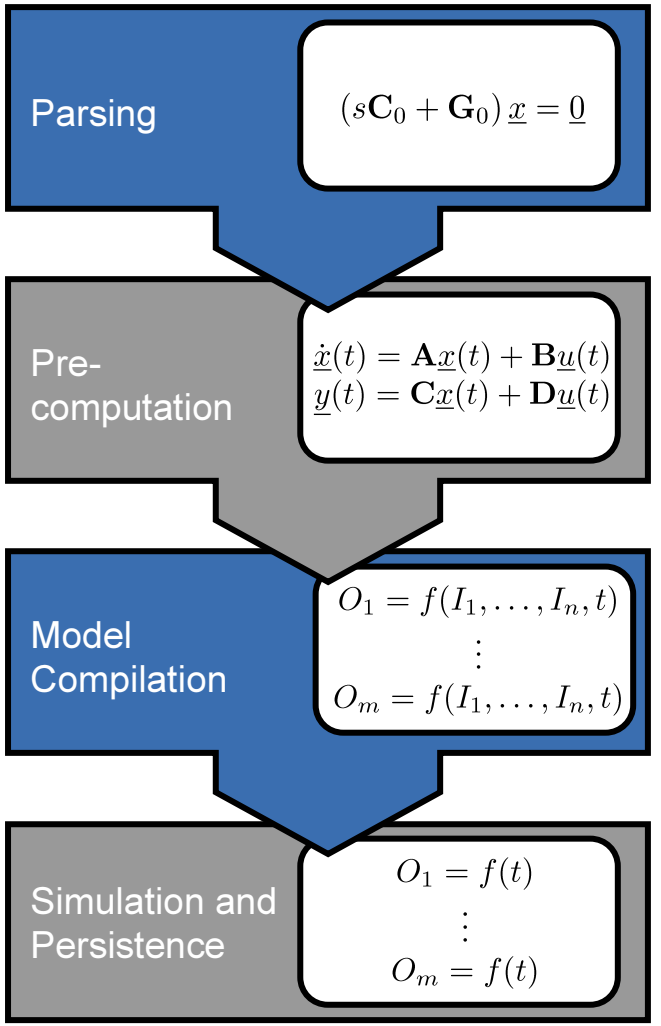


Outline

- Introduction
- The PRAISE Approach
- Implementation
- Application to Wire Models
- Conclusion and Outlook

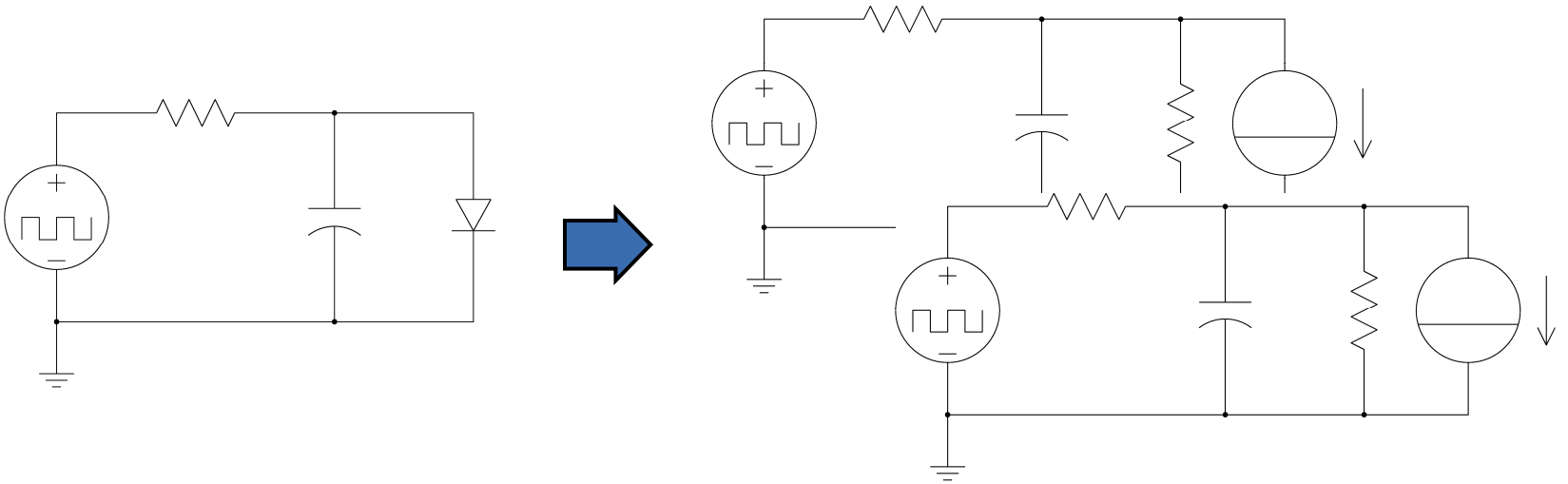
Simulation Flow

- Parsing
  - Read netlist
  - Perform MNA
- Precomputation
  - Solve circuit equations
  - Transformation to state space
- Model Compilation
  - Compute output functions
- Simulation and Persistence
  - Introduce inputs
  - Save precomputed circuit models
  - SystemC-Interface



Nonlinear Circuits

- Library of elements tailored to the automotive domain
- Using piecewise linear models for nonlinear elements
- Circuit simulation using different states depending on used models



Model Generation

Precomputation

- Modified Nodal Analysis (MNA)

$$(s\mathbf{C}_0 + \mathbf{G}_0)\,\underline{x} = \underline{0}$$

Elimination of excessive variables

$$s\begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{G}_{11} & \mathbf{G}_{12} \\ \mathbf{G}_{21} & \mathbf{G}_{22} \end{bmatrix}\begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \end{bmatrix} = \underline{0}$$

$$\mathbf{G}_{21}\underline{x}_1 + \mathbf{G}_{22}\underline{x}_2 = \underline{0} \Longleftrightarrow \underline{x}_2 = -\mathbf{G}_{22}^{-1}\mathbf{G}_{21}\underline{x}_1$$

Generation of transfer functions

$$\mathbf{H}(s) = \frac{n_1 s^3 + n_2 s^2 + n_3 s + n_4}{s^4 + d_1 s^3 + d_2 s^2 + d_3 s + d_4}$$

State space representation from system equations

- Minimal state space realization

$$\begin{aligned} \dot{\underline{x}}_1(t) &= \mathbf{A}\underline{x}_1(t) + \mathbf{B}\underline{u}(t) \\ \underline{y}(t) &= \mathbf{C}\underline{x}_1(t) + \mathbf{D}\underline{u}(t) \end{aligned}$$

State space transformation

State Space Simulation

Precomputation

- Minimal state space realization

$$\begin{aligned} \dot{\underline{x}}(t) &= \mathbf{A}\underline{x}(t) + \mathbf{B}\underline{u}(t) \\ \underline{y}(t) &= \mathbf{C}\underline{x}(t) + \mathbf{D}\underline{u}(t) \end{aligned}$$

- Assumption: $\underline{x}(t) = \mathbf{T}\tilde{\underline{x}}(t)$

$$\begin{aligned} \dot{\tilde{\underline{x}}}(t) &= \tilde{\mathbf{A}}\tilde{\underline{x}}(t) + \tilde{\mathbf{B}}\underline{u}(t) \\ \underline{y}(t) &= \tilde{\mathbf{C}}\tilde{\underline{x}}(t) + \mathbf{D}\underline{u}(t) \end{aligned}$$

$$\begin{aligned} \mathbf{T} &= \text{Eigenvectors}\,(\mathbf{A}) \\ \tilde{\mathbf{A}} &= \mathbf{T}^{-1}\mathbf{A}\mathbf{T} = \text{diag}\,(\lambda_1, \ldots, \lambda_n) \\ \tilde{\mathbf{B}} &= \mathbf{T}^{-1}\mathbf{B} \\ \tilde{\mathbf{C}} &= \mathbf{C}\mathbf{T} \end{aligned}$$

n independent differential equations

- Initial values

$$\underline{x}(0) = \mathbf{T}\tilde{\underline{x}}(0) = \mathbf{T}\begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \quad \Longleftrightarrow \quad \tilde{\underline{x}}(0) = \mathbf{T}^{-1}\underline{x}(0)$$

Output Functions

Simulation

- Time domain solution of state space equations

$$\underline{\tilde{x}}(t) = e^{\tilde{\mathbf{A}}t}\underline{\tilde{x}}_0 + \int_0^t e^{\tilde{\mathbf{A}}(t-\tau)}\tilde{\mathbf{B}}\underline{u}(\tau)d\tau$$

$$\underline{y}(t) = \tilde{\mathbf{C}}e^{\tilde{\mathbf{A}}t}\underline{\tilde{x}}_0 + \tilde{\mathbf{C}}\int_0^t e^{\tilde{\mathbf{A}}(t-\tau)}\tilde{\mathbf{B}}\underline{u}(\tau)d\tau + \mathbf{D}\underline{u}(t)$$

- Assumption: $\underline{u}(t) = \text{const},\ 0 \le t < T$
- Simplified time domain solution
  - Only valid if $\tilde{\mathbf{A}}$ is diagonal

$$\underline{\tilde{x}}(t) = e^{\tilde{\mathbf{A}}t}\left(\underline{\tilde{x}}_0 + \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{B}}\underline{u}(t)\right) - \tilde{\mathbf{A}}^{-1}\tilde{\mathbf{B}}\underline{u}(t)$$

Initial state

Inputs

Matrix exponential function

$$\underline{y}(t) = \tilde{\mathbf{C}}\underline{\tilde{x}}(t) + \mathbf{D}\underline{u}(t)$$

Matrix Exponential

- Even the analytical expression of a 2x2 matrix is complex:

$$e^{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} = \begin{bmatrix} \frac{e^{\frac{a+d}{2}}\left(\sqrt{4bc+(a-d)^2}\cosh\left[\frac{1}{2}\sqrt{4bc+(a-d)^2}\right]+(a-d)\sinh\left[\frac{1}{2}\sqrt{4bc+(a-d)^2}\right]\right)}{\sqrt{4bc+(a-d)^2}} & \frac{2be^{\frac{a+d}{2}}\sinh\left[\frac{1}{2}\sqrt{4bc+(a-d)^2}\right]}{\sqrt{4bc+(a-d)^2}} \\ \frac{2ce^{\frac{a+d}{2}}\sinh\left[\frac{1}{2}\sqrt{4bc+(a-d)^2}\right]}{\sqrt{4bc+(a-d)^2}} & \frac{e^{\frac{a+d}{2}}\left(\sqrt{4bc+(a-d)^2}\cosh\left[\frac{1}{2}\sqrt{4bc+(a-d)^2}\right]+(-a+d)\sinh\left[\frac{1}{2}\sqrt{4bc+(a-d)^2}\right]\right)}{\sqrt{4bc+(a-d)^2}} \end{bmatrix}$$

- Matrix exponential defined by a power series

$$e^{\mathbf{A}t} = \sum_{k=0}^{\infty} \frac{\mathbf{A}^k t^k}{k!} = \mathbf{I} + \mathbf{A}t + \frac{\mathbf{A}^2 t^2}{2!} + \cdots$$

Numerical computation difficult

- Our approach: Simplified solution for diagonal matrices

$$e^{(\operatorname{diag}(\lambda_i))} = e^{\begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix}} = \begin{bmatrix} e^{\lambda_1} & & 0 \\ & \ddots & \\ 0 & & e^{\lambda_n} \end{bmatrix}$$

Numerical computation trivial

Outline

- Introduction
- The PRAISE Approach
- Implementation
- Application to Wire Models
- Conclusion and Outlook

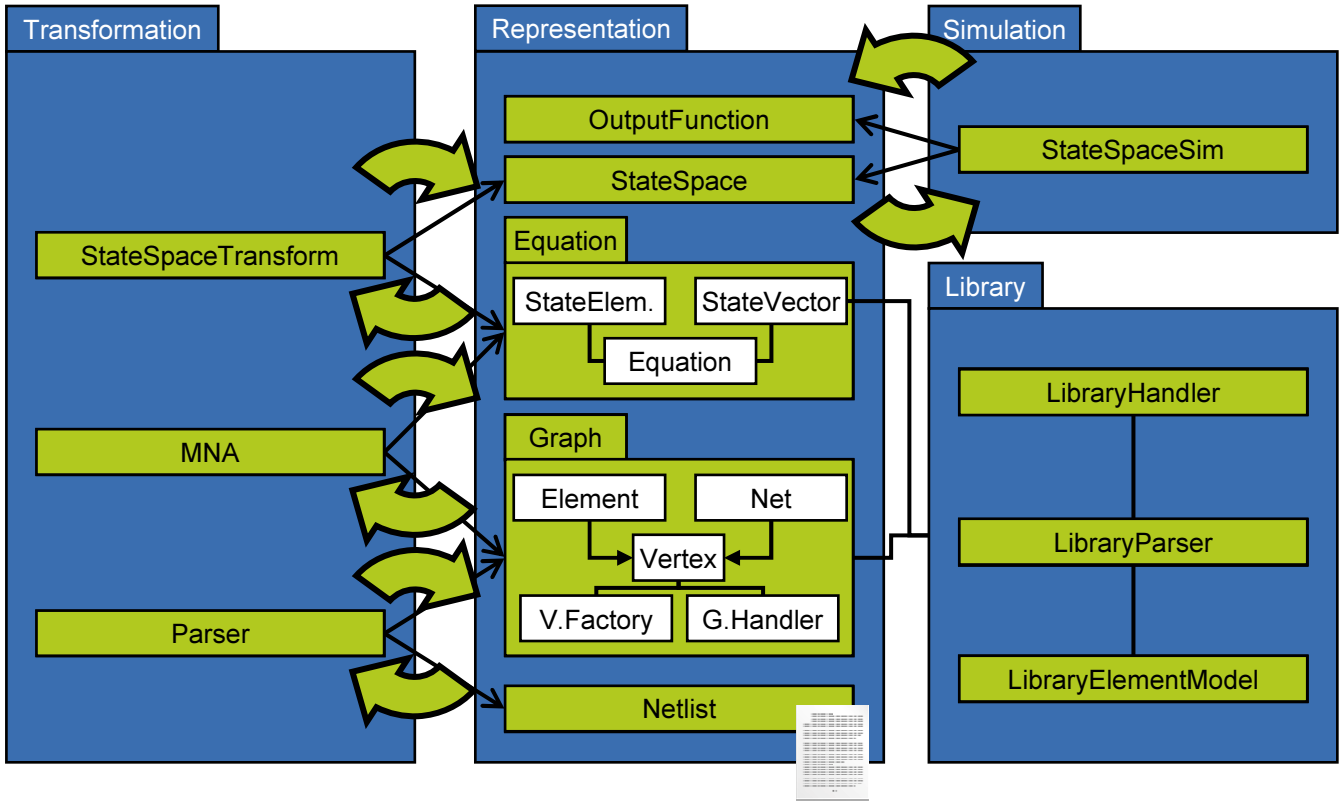
Implementation Details

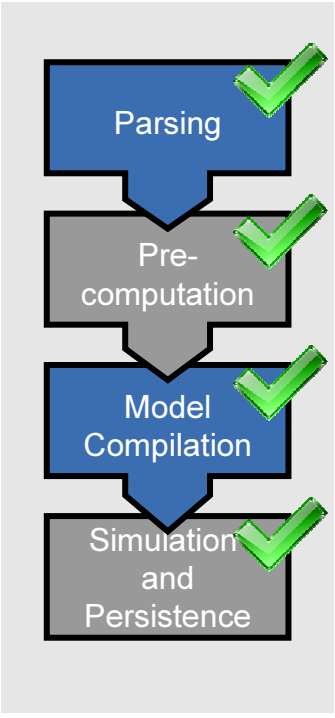
- Two different implementations exist:

- Prototype
  - Evaluation of methodologies
  - Precomputation: Mathematica and Analog Insydes Toolbox
  - Simulation: MATLAB

- High-performance
  - Object-oriented software design in C++
  - Using open source libraries only
  - Providing statically linked binary
  - SystemC-Interface

C++ Software Architecture





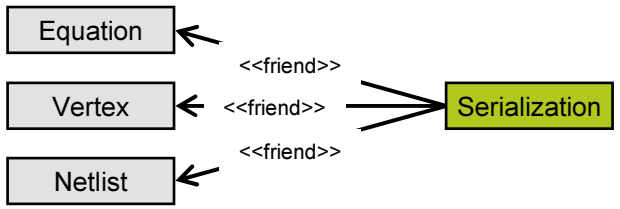
Persistence of Circuit Models

- Storage of precomputed circuit (wire) models required
- Development of an own file format imaginable
  - Conversion between file format and data structures
  - Implementation is time-consuming and error-prone
- Alternative: Serialization
  - Converting a C++ object into a sequence of data bits
  - Identical copies of the original object can be created
  - Pointers and references difficult to handle

Serialization: Advantages

- The data contained in the model can be easily adjusted
- Different versions of a circuit (wire) model possible, e.g. with and without topology
- New data types relatively easy to add
- No netlist parsing required (speed)

- Technical details
  - Our implementation uses the Boost library with STL and pointer/reference support
  - Single class handles all objects

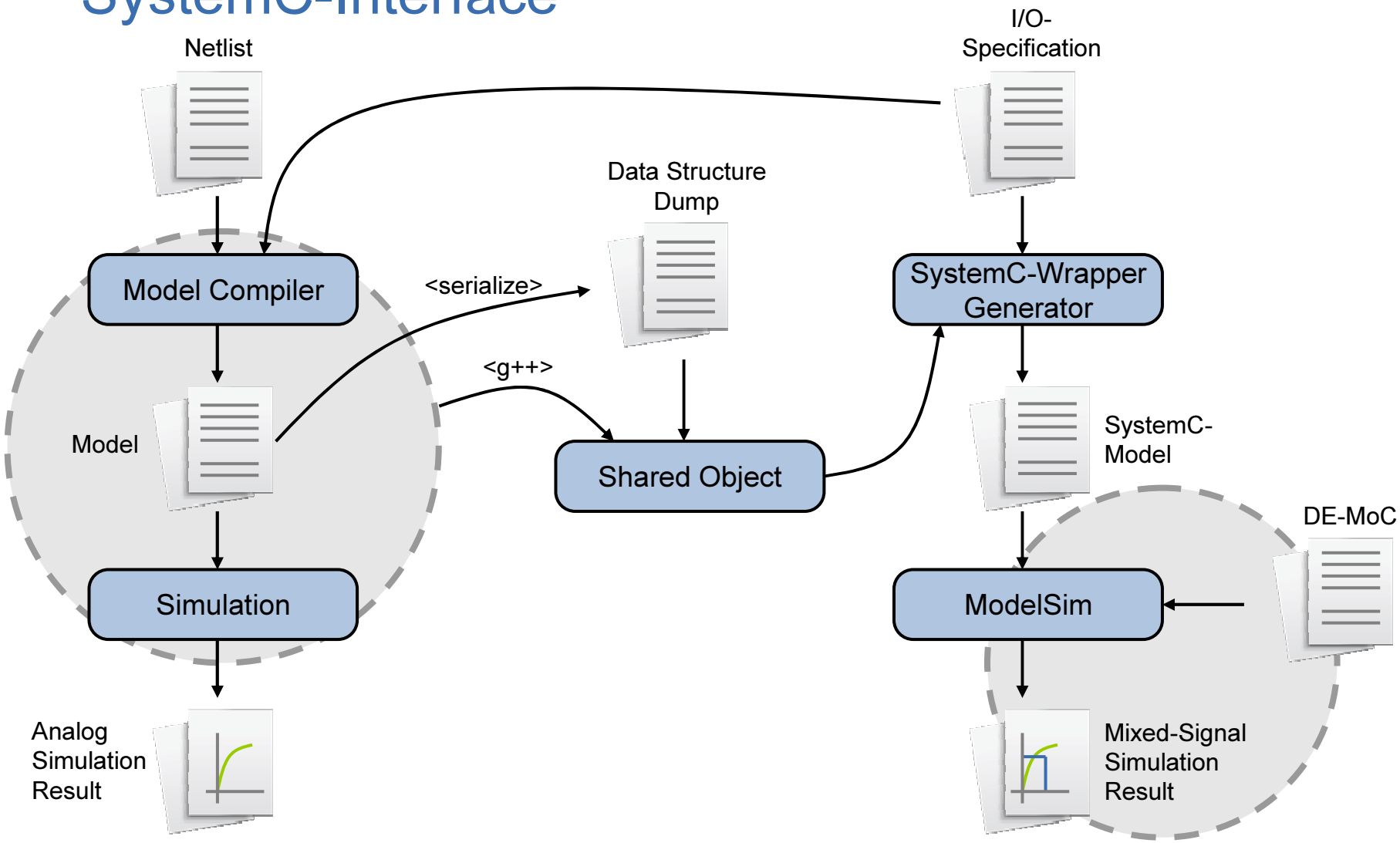


SystemC Simulation Kernel

- I/O-Specification provided by an XML-based data format
- Declarations of inputs and outputs
- Step clock for external synchronization

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE praiseSpec SYSTEM "praise.dtd">
<module name="valveControl">
    <inputs>
        <input name="V,n_1,n_2" type="double"/>
        <input name="I,n_15,n_21" type="double"/>
    </inputs>
    <outputs>
        <output name="V,n_12,n_38" type="double"/>
        <output name="V,n_27,n_39" type="double"/>
    </outputs>
    <stepclk name="slck">
</module>
```

SystemC-Interface



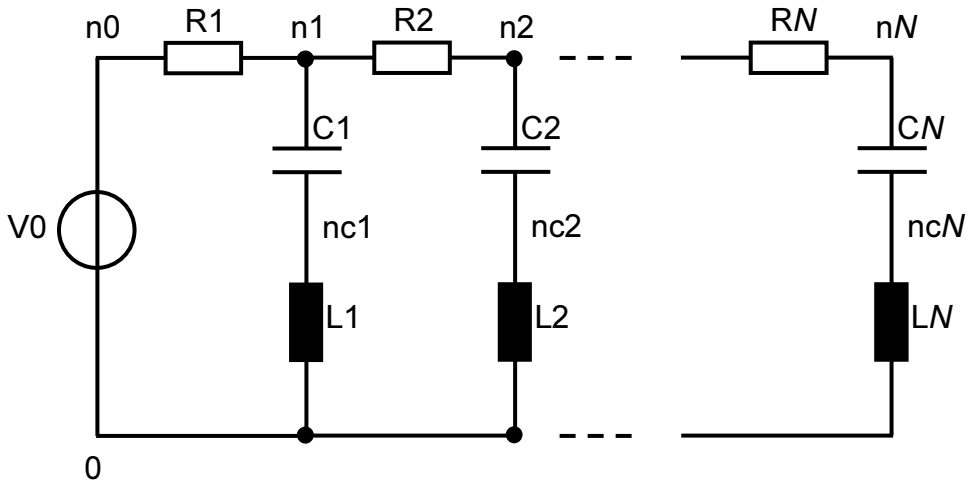
Outline

- Introduction
- The PRAISE Approach
- Implementation
- Application to Wire Models
- Conclusion and Outlook

Wire Models in Terms of PRAISE

- Modeling wires as linear networks

- Example RCL circuits
  - Circuit size depending on the parameter *N*
  - Linear circuit with *(3N+1)* elements
  - *(2N)*th order (*N* capacitors and *N* inductors)



C++ Serialization: Runtime and File Sizes

Circuit Size *N*	Plain [byte]	Zipped [byte]	Read [s]	Write [s]
100	46*k*	8.6*k*	0.0	0.01
200	92*k*	17*k*	0.01	0.01
250	115*k*	21*k*	0.01	0.01
300	139*k*	25*k*	0.01	0.01
400	185*k*	32*k*	0.01	0.01
500	231*k*	40*k*	0.02	0.01
1000	463*k*	77*k*	0.04	0.02

C++ Model Generation: RC Runtime

Circuit Size *N*	Parsing [s]	MNA [s]	Eigenvectors [s]	StateSpaceTr. [s]	Total [s]
100	0.01	0.01	0.05	0.08	0.13
200	0.01	0.02	0.43	0.63	0.72
250	0.01	0.04	0.84	1.38	1.50
300	0.01	0.05	1.52	2.74	2.92
400	0.02	0.09	4.11	7.10	7.27
500	0.03	0.14	8.24	13.61	13.94
1000	0.10	0.56	49.42	97.09	98.01

C++ Model Generation: RCL Runtime

Circuit Size *N*	Parsing [s]	MNA [s]	Eigenvectors [s]	StateSpaceTr. [s]	Total [s]
100	0.00	0.05	0.32	0.68	0.86
200	0.01	0.18	2.99	7.33	7.85
250	0.02	0.27	6.10	13.71	14.29
300	0.04	0.44	9.65	24.19	24.85
400	0.05	0.73	23.72	58.56	59.66
500	0.06	1.14	48.27	118.05	119.81
1000	0.29	6.32	432.25	1015.44	1020.09

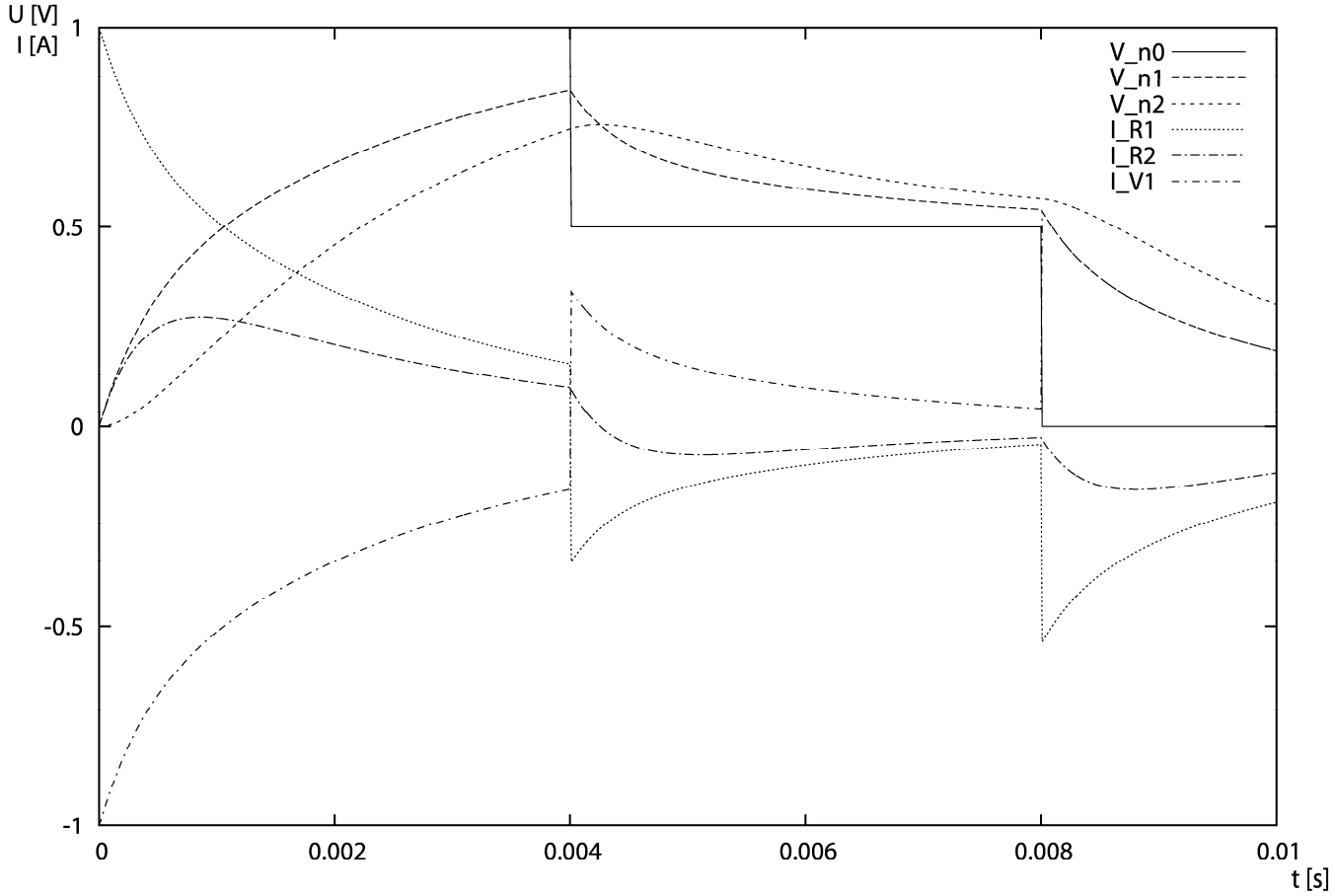
x8

x9

Feasible for mostly linear (automotive) circuits

Simulation Results

- RC-network with 2 capacitors

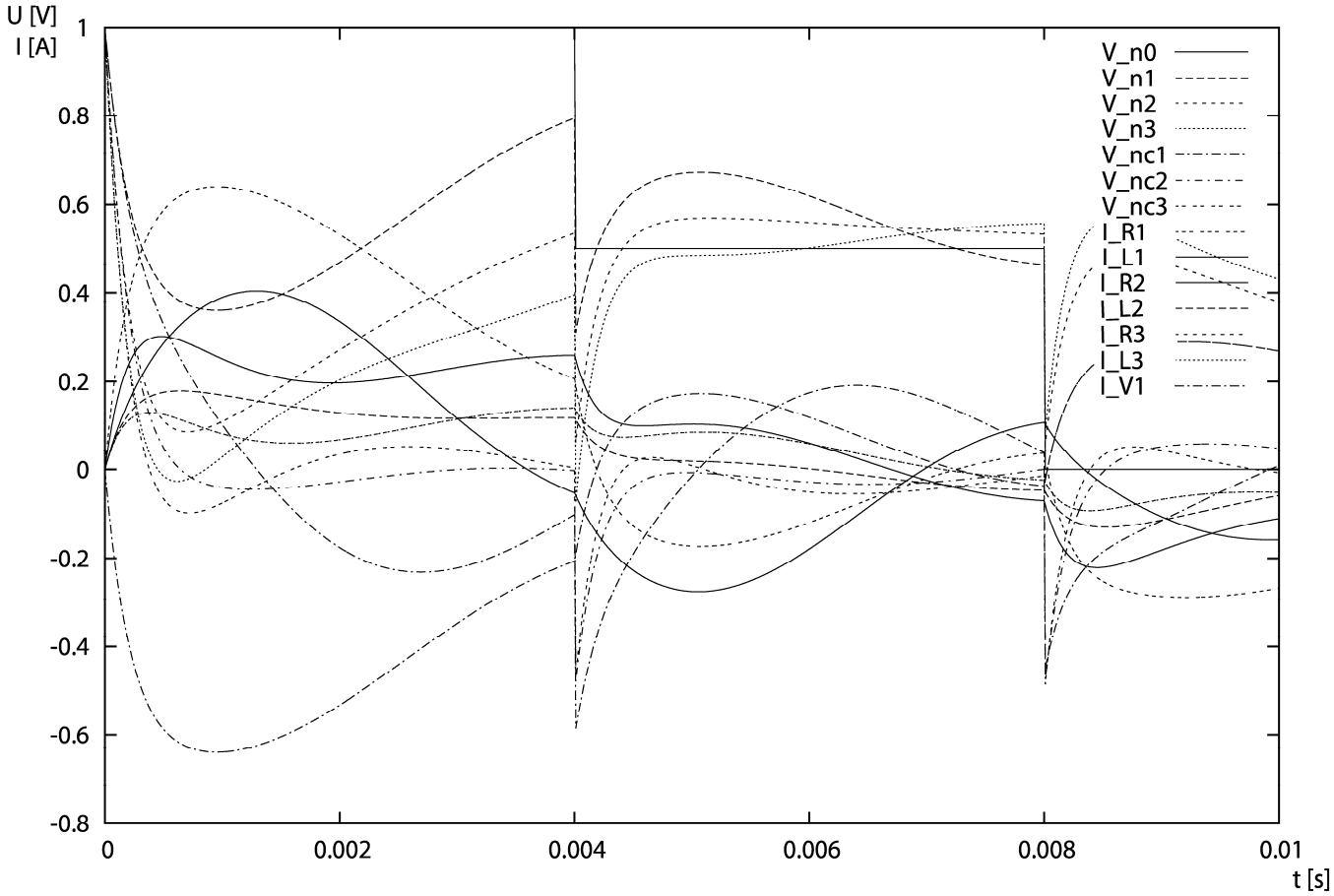


Prototype Simulation: RC Runtime

Circuit Size *N*	Model Generation [s]	Simulation [s]	Total [s]
1	0.93	0.40	1.33
2	1.04	0.69	1.73
3	1.32	0.83	2.15
4	1.60	1.02	2.62
5	2.11	1.28	3.39
6	2.47	1.58	4.05

Simulation Results

- RCL-network with 3 capacitors and 3 inductors



C++ Simulation: Runtime Issues?

- Performance issues due to the exponential function `exp()`?
- Runtime comparison
  - Read value from an array initialized with random values
  - Apply operation
  - Write result to another array

$100 \cdot 10^6$ Operations (Data type `double`)	Addition + [s]	Multiplication * [s]	`exp()` [s]	Exp. / Mul.
x86_64-redhat-linux AMD Opteron 8220, 2.8 GHz	1.51	1.08	2.65	2.5
i386-mingw32 (Cross) Intel Xeon "Irwindale", 3.6 GHz	2.58	2.42	118.41	48.9
i386-redhat-linux Intel Atom N270, 1.6 GHz	2.37	1.92	39.30	20.8
i386-mingw32 (Cross) Intel Atom N270, 1.6 GHz	2.53	2.39	129.14	54.0

Insignificant performance loss on recent systems

Conclusion and Outlook

- PRAISE Approach for accelerated transient analysis

- Precomputation: Model generation
- Simulation: Fast evaluation of exponential terms

- Prototype using Computer Algebra Systems
- C++ high-performance implementation with SystemC-Interface

- Circuit simulation using piecewise linear models
- Model library tailored to the automotive domain

IMS
Institute of Microelectronic Systems
Leibniz Universität Hannover

Leibniz Universität Hannover

Thank you for your time and attention.