High-Performance Model Compilation for Complex Behavioral Models

September 20th, 2007
Bottom-Up Modeling Flow

- Import of several netlist formats (Spice, Spectre, etc), Cadence Design Framework integration
- Automated setup of symbolic network equations through an extended MNA and symbolic device models
- Efficient symbolic model reduction techniques for nonlinear DAEs with continuous numeric error control
- Algebraic reformulation and restructuring of the DAEs to optimize simulation performance (no impact on the accuracy)
- Export the DAEs to several common AHDLs incl. VHDL-AMS, Verilog-A, MAST, (Modelica)

No model reduction, 100% accuracy, comparable performance
Example $\mu$A741
## Analytic Model of μA741

<table>
<thead>
<tr>
<th>μA741</th>
<th>Dimension</th>
<th>Sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit</td>
<td>62</td>
<td>90.3%</td>
</tr>
<tr>
<td>Model (MNA)</td>
<td>368</td>
<td>98.8%</td>
</tr>
</tbody>
</table>

- **26 Transistors** (Gummel-Poon)
- **No model reduction applied**
- **564 Parameters**
- **115 kB Code Size**
- **1300 Lines of Code VHDL**
- **10 additional internal variables per BJT**

**Notes:**

- 564 Parameters
- 115 kB Code Size
- 1300 Lines of Code VHDL
- 10 additional internal variables per BJT
Motivation

Circuit (Refer.)

Today
- Model (full)
- Model (reduced)

Speed-Up 5x

Target
- Model (full)
- Model (reduced)

0.1 1 10 100

$t_{\text{ref}}$

200x $t_{\text{ref}}$

<10x $t_{\text{ref}}$

0.2

Reduction

Adaptation
Reasons for Performance Problems

• „Intelligence“ of built-in device models missing
• Equation formulation not optimized for numerical methods
• Model compilers do not efficiently handle such complex behavioral models
• Restrictions by using common AHDLs
  – procedural statements in VHDL-AMS
  – simultaneous equation sets in Verilog-A
  – no optimization of Jacobian matrix possible
Model Compilation with Analog Insydes

- Generation of compiled models for Qimonda's inhouse-simulator Titan
- Analog Insydes is used as platform for the model compilation
- Key features:
  - Efficient processing of simultaneous DAE
  - Local solving of sequential equations
  - Data locality (cache)
  - Sparse handling and data structures
  - Direct interface to the simulator kernel

---

No model reduction, 100% accuracy, comparable performance
Example – Sequential Network Equations

Sequential

\[
vd[t] = -\frac{RS ISAC[t]}{AREA} + V$IN[t] - V$OUT[t]
\]
\[
id[t] = AREA \left[ 0.00333167 qvd[t] + IBV \left( -1 + e^{\frac{-0.00181069 qvd[t]}{k}} \right) IS \right] + GMIN vd[t]
\]
\[
cj[t] = AREA CJIF \left[ vd[t] < 0.25, \frac{1}{(1.2 vd[t])^{0.333}}, 2.51926 \left( 0.3335 + 0.666 vd[t] \right) \right]
\]

Simultaneous

\[
I$V[GND][t] + \frac{V$GND[t] - V$OUT[t]}{R_0} + C_0 \left( V$GND'[t] - V$OUT'[t] \right) = 0
\]
\[
I$AC[t] + I$VIN[t] = 0
\]
\[
-I$AC[t] + \frac{-V$GND[t] + V$OUT[t]}{R_0} + \frac{V$OUT[t] - V$OUT2[t]}{R_L} + C_0 \left( -V$GND'[t] + V$OUT'[t] \right) = 0
\]
\[
I$V[OUT][t] + \frac{-V$OUT[t] + V$OUT2[t]}{R_L} = 0
\]
\[
I$AC[t] = id[t] + 1.15 \times 10^{-8} id'[t] + cj[t] vd'[t]
\]
\[
V$IN[t] = VIN
\]
\[
V$OUT[t] = 0
\]
\[
V$GND[t] = 0
\]
Newton’s Method for Sequential DAEs

\[
\begin{bmatrix}
\frac{\partial f_{\text{seq}}}{\partial y} & \frac{\partial f_{\text{seq}}}{\partial x}
\end{bmatrix}
\begin{bmatrix}
\Delta y
\end{bmatrix}
= \begin{bmatrix}
J_{11} & J_{12}
\end{bmatrix}
\begin{bmatrix}
\Delta y
\end{bmatrix}
- \begin{bmatrix}
f_{\text{seq}}(y, x)
f_{\text{sim}}(y, x)
\end{bmatrix}
\]

\[
\Delta y = J_{11}^{-1} \cdot \left( f_{\text{seq}}(y, x) + J_{12} \Delta x \right)
\]

\[
\left( J_{22} - J_{21} \cdot J_{11}^{-1} \cdot J_{12} \right) \cdot \Delta x = -f_{\text{sim}}(y, x) + J_{21} \cdot J_{11}^{-1} \cdot f_{\text{seq}}(y, x)
\]

\[
J^*_{22} \cdot \Delta x = -f_{\text{sim}}(y, x)
\]

1. Calculate \(y_{n+1}\) from \(x_n\) (sequential equations)
2. Update \(J\) with \(y_{n+1}\) and \(x_n\)
3. Calculate Schur complement
   \[
   J^*_{22} = J_{22} - J_{21} \cdot J_{11}^{-1} \cdot J_{12}
   \]
4. Solve
   \[
   J^*_{22} \cdot \Delta x = -f_{\text{sim}}(y, x)
   \]
Methods to Calculate Schur Complement

1) **Symbolic Schur complement**
   - extremely complex expressions, impossible

2) **Numerical calculation**
   - matrix inversion and multiplications necessary

3) **Semi-symbolic Schur complement with roll-out**
   - symbolically calculate the Schur complement with references to the evaluated Jacobian matrix
   - generate an efficient transformation process by elimination of the submatrix $J_{21}$

\[
\tilde{J}_{22} = J_{22} - J_{21} \cdot J_{11}^{-1} \cdot J_{12}
\]
Example Schur Complement

Structural Matrix

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>J[1,4]</th>
<th>J[1,5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>J[2,4]</td>
<td>J[2,5]</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>J[3,4]</td>
<td>J[3,5]</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>J[4,4]</td>
<td>J[4,5]</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>J[5,4]</td>
<td>J[5,5]</td>
</tr>
</tbody>
</table>

Transformation Process

- $J[2,1] = 0$

- $J[3,1] = 0$


...
Optimization Strategies

• **Identification of Sequential Equations (BLT Transf.)**
  Automated recognition of sequential equations from general DAE systems to reduce the number of the simult. equations

• **Common Subexpression Elimination (CSE)**
  CSE on DAE level to reduce redundancy within the function evaluation and improve data locality

• **Loop Invariant Expressions / Preloading**
  Preevaluation of constant and parametric expressions within the initialization phase, recognition of parametric subexpressions

• **Copy / Constant Propagation**
  Reduce memory accesses by propagation of copied or constant values into subsequent expressions
Example - Optimized DAE System

\[ v_d(t) = V_{IN} - \frac{RS I_{AC}(t)}{AREA} - V_{OUT}(t) \]

\[ \text{SeqExpr2}(t) = - \frac{V_{OUT}(t)}{R_0} \]

\[ \text{SeqExpr3}(t) = \frac{V_{OUT}(t)}{R_L} \]

\[ i_d(t) = -\text{AREA} \left( e^{-0.0033167q(BV+vd[t])} \right) \] 
\[ \text{IBV} - \left( -1 + e^{0.00181069qvd[t]} \right) \] 
\[ IS + \text{GMIN} vd[t] \]

\[ c_j(t) = \text{AREA CJO If} \left[ vd[t] < 0.25, \frac{1}{(1-2vd[t])0.333}, 2.51926(0.3335 + 0.666 vd[t]) \right] \]

\[ \text{SeqExpr1}(t) = -C_0 V_{OUT}'[t] \]

\[ I$V_{GND}(t) = -\text{SeqExpr1}(t) - \text{SeqExpr2}(t) \]

\[ -I$AC(t) - \text{SeqExpr1}(t) - \text{SeqExpr2}(t) + \text{SeqExpr3}(t) = 0 \]

\[ I$AC(t) = i_d(t) + 1.15 \times 10^{-8} i_d'(t) + c_j(t) v_d'(t) \]

- Dimension reduced from 8x8 to 2x2
- No redundancy within DAEs (3 common subexpressions eliminated)
Performance Improvement

- opamp741 (ZMS)
- CPU-Time [s]

TML (sparse loading)  ZMS (sim)  ZMS (seq)  ZMS (opt)  Circuit

- Loading  Solving

8x Speed-Up (14x)  2x Slow-Down

MUMPS Solver
ZMS Compiler
Local Solving
Optimizations

6x
50%
Outlook

Analog Insydes

Circuit Netlist → Netlist Import

Symbolic Device Models → Equation Setup

Circuit Equations

Model Reduction → Model Reduction

Simplified Eqs.

Equation Optimization

Model Export

AHDL Model

Compiled Model

Behavioral Model → Model Import

Model Equations

Model Reduction

Model Optimization

Translation, Device Model Compilation

new

improved

missing
Summary

- Model generation based on a **symbolic analysis flow** using the toolbox Analog Insydes presented
- Such complex and high-dimensional AHDL-based models have **insufficient simulation performance**
- **New model compiler** for the Titan simulator developed
- **Efficient processing** of the generated models due to
  - Local solving with semi-symbolic Schur complement
  - Optimization techniques on DAE level
  - Close interaction between model and simulator
- **Performance improvement** by a factor of at least 8
Performance Problem

- Circuit
  - Equations
  - Solver
  - Waveform

- Circuit
  - Equations
  - Model
  - Solver
  - Waveforms

- Circuit simulation
- Symbolic Analysis
- Behavioral simulation

- Equivalent
- Similar
- Identical

- ~200x
Dimension of the Linearized Systems

- Enormous reduction of the model dimension
- Only minor difference between optimized model and circuit simulation
- Local solving and optimizations very efficient

Sim: All model equations simultaneous
Seq: Sequential equations defined within device models
Blt: With identification of sequential equations
Cir: Dimension of the MNA equations in circuit simulation
Jacobian Matrix of Sequential DAEs

- Sequential Variables
- Simultaneous Variables
- Port Volt.

Typically much larger dimension

Solved within the simulator kernel

Very few

- Model Equations
- Nodal Equations (Ports)
- Nodal Equations
- Voltage Equations

Typically much larger dimension

Sequence

Simultaneous

Nodal Equations (Ports)

Model Equations

Typically much larger dimension

Typically much larger dimension

Typically much larger dimension

Typically much larger dimension
1) Transform $S_{11}$ to unity matrix
   $\Rightarrow$ changes within $S_{12}$, fill-ins

2) Eliminate $S_{21}$
   $\Rightarrow$ update of $S_{22}$, fill-ins
A system of differential algebraic equations (DAE)

\[ f(y, y', x, x', t) = 0 \]

with

\[ y \in \mathbb{R}^m \quad \text{sequential variables} \]
\[ x \in \mathbb{R}^n \quad \text{simultaneous variables} \]

\[ f = \{ f_{\text{seq}}, f_{\text{sim}} \} \]

\[ f_{\text{seq}} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \rightarrow \mathbb{R}^m \quad \text{sequential equations} \]
\[ f_{\text{sim}} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \rightarrow \mathbb{R}^n \quad \text{simultaneous equations} \]

is of sequential structure, if

\[ y_i = f_{\text{seq},i} \left( y_1, \ldots, y_{i-1}, y'_1, \ldots, y'_{i-1}, x, x', t \right) \quad \text{for} \quad i = 1 \ldots n \]
## Modeling DAE Systems with Seq. Structure

<table>
<thead>
<tr>
<th><strong>Verilog-A</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequential Equations</strong></td>
</tr>
<tr>
<td><code>// Analog procedural</code></td>
</tr>
<tr>
<td><code>// assignment</code></td>
</tr>
<tr>
<td><code>real vd;</code></td>
</tr>
<tr>
<td><code>vd = -RS*X(I_A))/AREA-</code></td>
</tr>
<tr>
<td><code>X(V_K)+V(A);</code></td>
</tr>
<tr>
<td><strong>Simultaneous Equations</strong></td>
</tr>
<tr>
<td><code>// (Indirect) branch</code></td>
</tr>
<tr>
<td><code>// contribution</code></td>
</tr>
<tr>
<td><code>DAEVa r V_G;</code></td>
</tr>
<tr>
<td><code>X(V_G) &lt;+ X(V_G)+id-0.115E-</code></td>
</tr>
<tr>
<td><code>7*X(id_d1)-</code></td>
</tr>
<tr>
<td><code>X(I_A)+cj*X(vd_d1);</code></td>
</tr>
</tbody>
</table>
# Modeling DAE Systems with Seq. Structure

<table>
<thead>
<tr>
<th>Sequential Equations</th>
<th><strong>Verilog-A</strong></th>
<th><strong>VHDL-AMS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>// Analog procedural</td>
<td>// Analog procedural</td>
<td>-- Simultaneous</td>
</tr>
<tr>
<td>// assignment</td>
<td>real vd;</td>
<td>-- procedural statement</td>
</tr>
<tr>
<td></td>
<td>vd = -RS*X(I_A))/AREA-</td>
<td>vd := -RS*I_A/AREA-V_K+A;</td>
</tr>
<tr>
<td></td>
<td>X(V_K)+V(A);</td>
<td>-- or: Function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>function vd (I_A,...) is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simultaneous Equations</th>
<th><strong>Verilog-A</strong></th>
<th><strong>VHDL-AMS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>// (Indirect) branch</td>
<td>// (Indirect) branch</td>
<td>-- Simple simultaneous</td>
</tr>
<tr>
<td>// contribution</td>
<td>DAEVar V_G;</td>
<td>-- statement</td>
</tr>
<tr>
<td></td>
<td>X(V_G) &lt;+ X(V_G)+id-</td>
<td>QUANTITY V_G : Voltage;</td>
</tr>
<tr>
<td></td>
<td>0.115E-7*X(id_d1)-</td>
<td>I_A-ID-0.115E-7*ID_D1-</td>
</tr>
<tr>
<td></td>
<td>X(I_A)+cj*X(vd_d1);</td>
<td>CJ*VD_D1 == 0.0 Tolerance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Current”;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>
Block Lower-Triangular Matrix

\[
\begin{align*}
\text{Seq. Equations} & : J_{11} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ J_{21} & = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ J_{22} & = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ \end{pmatrix}
\end{pmatrix}
\end{align*}
\]

\[
\text{Sim. Vars.} & : J_{12} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ J_{22} & = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ \end{pmatrix}
\end{pmatrix}
\]
Speed-Up for TML Models

The chart illustrates the CPU-time comparison between different methods for TML model computations:

- **TML (default)**: 49.36 seconds
- **TML (sparse solver)**: 0.70 seconds
- **TML (sparse loading)**: 6.85 seconds

The chart shows a 10x speed-up in CPU time when using sparse loading compared to the default method.

**Legend**

- Loading
- Solving

**Graph Elements**

- **Sparse Solver**: 27.84
- **Convergence**: 64.79
- **Sparse Loading**: 0.64
Sequential Equations (Definition)

\[
x_1 = G_1(y)
\]

\[
x_2 = G_2(y, x_1)
\]

\[
x_3 = G_3(y, x_1, x_2)
\]

\[\ldots\]

\[
x_n = G_1(y, x_1, \ldots, x_{n-1})
\]

\[
F(x, y) = 0
\]

\[
\tilde{F}(\underline{y}) = F(x(y), \underline{y}) = 0
\]

**Advantages:**

- Reduced dimension of Jacobian matrix
- Iterative solution for less variables

\[(n + m) \times (n + m) \rightarrow m \times m\]
Recognition of Sequential Equations

8x8

Identification

Seq

3x3
Simulator Comparison for opamp741

![Diagram showing comparison between different simulators and models for opamp741, with bars representing time steps and iterations.](image-url)